KEDBYTE

SOFTWARE · AI · AUTOMATION

HOW MONEY MOVES

Volume II — The Card

The complete plain-English guide to payments and banking, followed by the engineer's version of the same thing.

WRITTEN BY

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

Five Volumes • Fifty-Four Chapters • First Edition

HOW MONEY MOVES

Volume II — The Card

Author	Shikhar Singh
Designation	Founder & Chairman, KedByte Technologies Private Limited
Published by	KedByte Technologies Private Limited
Imprint	KedByte Press
Edition	First Edition
Subject	Payments, banking systems, financial technology
Extent	Five volumes, fifty-four chapters

Accuracy and currency. This book describes technology that changes. Specifications, prices, version numbers, product names and industry figures were checked at the time of writing and are dated in the text wherever they are liable to change. The chapters on artificial intelligence in particular describe a field that moves quickly; readers are told, in the text, where to re-check a figure before relying on it. Where experts disagree, the book says so rather than choosing a side.

Disclaimer. The information in this book is provided on an "as is" basis, for education. The security chapter is written for defence: it explains how classes of attack work so that systems can be built to resist them, and it deliberately contains no working exploit code. Neither the author nor KedByte Technologies Private Limited shall have any liability to any person or entity with respect to any loss or damage caused, or alleged to have been caused, directly or indirectly by the information contained in this book.

Worked examples. Parts F and G are built around one real network fault and one real version-control session, reproduced exactly as they were observed, including the addresses, the traceroute output and the error messages. Nothing in those traces has been altered or invented.

FOREWORD

From the Founder & Chairman

You tap a card against a machine and a shop lets you leave with the coffee. Somewhere behind that gesture, money you cannot see is supposed to move from a place you have never visited to another place you have never visited, and almost nobody involved can tell you how. Not the person who served you. Often not the person who installed the terminal. Frequently not the people who work at the bank.

That collective not-knowing is expensive. It produces businesses that cannot reconcile their own takings, engineers who store money in floating point, support teams who ask customers for the one reference number that cannot possibly help, and founders who discover the difference between a payment being authorised and a payment being settled on the day it costs them something. None of these are stupidity. They are the predictable result of an industry that documents itself for people who are already inside it.

This book is the outsider's way in, written to be useful to the insider. It begins with what money actually is, which is a record of obligation rather than a substance, and it ends with regulation, dispute and the ugly arithmetic of reconciliation. In between: the card in your pocket and the small computer embedded in it, the machine on the counter, the message formats that have carried the world's card traffic for forty years, the difference between a payment clearing and a payment settling, the rails that move money without cards at all, and what happens when any of it fails.

The method is the same one used in *How Computers Work*, and it matters more than the coverage. Every idea is explained twice. First in plain language, with an everyday comparison and a worked example, such that a twelve-year-old could follow it. Then again in the exact professional vocabulary, with the real field numbers, the real timings and the real institution names that a practitioner would use. Between the two sits a section that most writing on this subject omits entirely: an explicit statement of where the simple version stops being true. A useful simplification told without warning is how people end up confidently wrong, and in payments confidently wrong is a category of loss.

One thing about this book differs from its predecessor, and the reader should know it. How a transistor works will not change. Payments will. Scheme rulebooks are revised annually, regulatory caps move, and migrations between message standards run on live deadlines. The volumes are therefore arranged so that decay is contained: the mechanics that will outlive us sit in the early volumes, and the material that is true as of writing is gathered deliberately at the end, marked as

such. Where a figure is time-bound, this book says so rather than pretending to a permanence it does not have.

At KedByte Technologies Private Limited we build systems that move other people's money, which is a privilege that ought to come with an obligation to explain the machinery. If this book leaves one reader able to follow their own money from a card terminal to a bank ledger, and to say honestly which part of that journey they have verified and which part they have merely assumed, it has done its job.

Start at Chapter 1. It begins long before cards, and for good reason.

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

BEFORE YOU START

How to read this book

The shape of every section

Every section of every chapter is built from the same six blocks, always in the same order. Once you know the shape, you can navigate two thousand pages without ever being lost.

Block	What it is for
PLAIN in simple words	The idea for a complete beginner. No jargon at all.
PLAIN a picture in your head	One everyday comparison, and then a line telling you exactly where that comparison stops being true.
PLAIN a worked example	Concrete numbers, worked step by step.
PLAIN what is really happening inside	The mechanism, still in easy words, but with nothing hand-waved.
TECHNICAL the engineer's version	The correct professional vocabulary, exact units, real specifications, real figures, standard numbers and the commands that observe it.
WORDS remember these	Each term given twice: the plain meaning, then the technical meaning.

Two ways to read it

1. **The single pass.** Read everything, in order, from Chapter 1. This is the intended route and it works.
2. **The double pass.** Read only the blocks marked PLAIN, front to back. You will understand the whole of computing. Then return to the start and read the blocks marked TECHNICAL to be able to hold your own in an engineering conversation.

Conventions used throughout

1. Almost everything is written as short numbered lines, one idea per line, so nothing hides inside a paragraph.
2. *The honest version* marks the places where a simple explanation is not strictly true. The accurate model follows immediately.

3. The book states whether a thing is a **standard** (written down in a specification), a **convention** (everybody just does it this way), or an **implementation detail** (true of one product, not of the idea).
4. In the artificial intelligence chapters, claims are separated into established fact, active research, and marketing claim.
5. Anything that goes stale is dated. Check it again before you rely on it.
6. Every chapter ends with *Common wrong ideas* and a twenty-line summary. If you are short of time, read those two first.

Where to find things

1. **Chapter 1** is the map of the whole book.
2. **Chapter 52** is the reference desk: the numbers, the command cheat sheets and the two diagnostic decision trees.
3. **Chapter 53** is the master timeline and a glossary of more than seven hundred terms, each given in plain words and then technically.

Contents

VOLUME II

The Card

The plastic, the chip that is a computer, the cryptogram it signs, the terminal that reads it, and the token that stands in for it.

CHAPTER

1

What Is Physically on a Card

11.0 What this chapter gives you

1. You will be able to name the six physical forms in which one card carries one piece of information, say which three are obsolete, and explain why all six are still there.
2. You will be able to read a track 2 record character by character, from the start sentinel to the check character, and say what each field is for.
3. You will be able to show that the 79-character and 40-character track limits are not round numbers but what the bit densities allow across 2.80 inches of usable track.
4. You will be able to explain why chip data written onto a blank magnetic stripe produces a card that is declined, and why that is a scheme rule rather than an implementation accident.
5. You will be able to read a three-digit service code and say whether the card is international, whether it carries a chip, and what the terminal should do about a PIN.
6. You will be able to say which of the eight ISO/IEC 7816-2 contacts do anything under EMV, and why C6 is a fossil.
7. You will be able to explain why a terminal accepting six brands runs six transaction state machines behind one tap.
8. You will be able to say why tag `57` is a track 2 record in shape only, and why tag `5F34` matters to anyone reconciling on the account number alone.
9. You will be able to justify the magnetic stripe's survival to 2033 in terms of declined transactions rather than technology, and read the economics off the published timetable.
10. You will be able to describe how a card is laminated, punched, milled and personalised, and say where the boundary falls between plastics manufacturing and cryptographic key management.

Take the card out of your wallet and put it on the table in front of you. It weighs about five grams, and it carries, in six different physical forms, one piece of information that could be written on the back of a receipt: which account to charge.

Six. Not one. The number is stamped into the plastic so it stands proud of the surface. It is printed in ink. It is written magnetically into a strip of iron oxide. It is stored electrically inside a silicon chip. It can be recited over a radio link at 13.56 megahertz. And on many cards it is printed a second time, in small type, on the back.

Three of those six are obsolete. All six are still there.

This chapter is about the object itself: what each layer physically is, which standard governs it, what it holds, and why an industry that is otherwise ruthless about cost has not thrown any of it away.

The plain version

Imagine you run a shop, and you have to leave a note for whoever opens up tomorrow. You do not know who that will be. It might be your grandmother, who only reads handwriting. It might be your colleague, who only checks the noticeboard. It might be the new assistant, who only reads text messages. You cannot teach any of them a new habit before morning, so you write the same note three times: once in pen, once pinned to the board, once as a text. The reader decides which copy they see.

A payment card is that note. It has to be readable by a card machine installed last month in a London coffee shop, by a petrol pump in rural Nebraska that has not been touched since 2009, and by a hotel in a country where the phone line to the bank is unreliable. Those machines cannot be upgraded on your behalf before you next buy a sandwich. So the card carries the same message in several formats at once.

What each copy of the note actually is

The raised numbers on the front. These exist because of a machine most people under forty have never seen. Before card machines had telephone lines, a shop had a metal device with a roller. The shopkeeper laid your card in it, put a paper form on top with carbon paper underneath, and rolled the handle across. The raised digits pressed through the carbon and printed your account number on the form, which went in the post to the bank. That is why the numbers stick out rather than being printed flat: the card was a printing plate.

The brown or black stripe on the back. This is magnetic tape, exactly the same idea as a cassette. It has three parallel lanes running along its length, and a machine reads it by dragging a tiny magnetic sensor along one lane while you swipe. The whole stripe holds less than a short text message. The first lane holds seventy-nine characters, enough for your account number, your name, the expiry date and a few extra digits. The second holds forty and no name. The third holds a hundred and seven and is, in practice, empty.

The gold square. This is not a memory chip. It is a computer, with a processor, memory and a program, powered by the card machine through those gold contacts, which is why nothing happens until you push the card in. Unlike the stripe, the chip does not simply hand over what it knows. The terminal asks it questions and it answers with the results of calculations. That difference is the entire reason chips replaced stripes.

The invisible loop of wire. Somewhere inside the plastic, running around the edge of the card, is a coil of very fine wire you cannot see or feel. When you hold the card near a contactless reader, the reader's own magnetic field induces a current in that loop, and the current powers the same chip. The card has no battery. It borrows power from the reader for the fraction of a second it needs.

The shiny hologram. A shopkeeper in 1990 could not phone the bank for every purchase, and needed a way to tell by eye that the card was real. A hologram was chosen because in 1985 you could not photocopy one, and printing one required equipment a forger with a garage did not have.

The white strip you sign. Once, the shopkeeper compared your signature on the receipt with the one on this strip. That comparison stopped being required years ago, and most people's cards are now unsigned.

A worked example

Suppose your card number is 4111 1111 1111 1111, it expires in December 2026, and your name is on it. Here is what is actually written into the second lane of the magnetic stripe, character for character:

```
;4111111111111111=26122011234567890 12?
```

Read it left to right. The semicolon says "the data starts here". Then sixteen digits of account number. The equals sign says "account number finished". Then `2612`, which is December 2026 written year-first. Then `201`, a three-digit code telling the terminal what the card is allowed to do. Then twelve digits the bank uses for its own purposes. Then a question mark, which says "data ends here", followed by one more character the reader uses to check it did not misread anything.

Thirty-eight characters. That is the whole of what a swipe transmits, and if you can read it you can make a working copy of the card. That single sentence is the reason the chip exists.

The first lane holds a richer version of the same thing, with your name in it:

```
%B4111111111111111^SINGH/SHIKHAR^2612201123456789?
```

The chip, by contrast, holds tens of thousands of characters, most of which it will never show anybody, because most of it is keys.

Where the plain version stops being true

The note-in-three-formats picture is the right shape. Four things about it are wrong, and each of the four matters.

First, the copies are not identical, and the differences are the security. The plain version says the chip and the stripe carry the same message. They carry deliberately different messages. The chip does hold something that looks like the second lane of the stripe, and it even uses the same layout, but one field inside it is set to a different value on purpose. If a criminal reads the chip's copy and writes it onto a blank magnetic stripe, the resulting card will be declined, because the value the issuer expects from a swipe is not the value the chip carries. This is not an accident of implementation. It is a scheme rule, and it is why chip data cannot be trivially downgraded into a stripe clone. Chapter 13 is about the four different check values that make this work.

Second, the chip does not "hold" your account number in the sense the analogy implies. It holds it, yes, and will read it out. But the thing the terminal actually needs from the chip is not stored on it at all: it is computed, freshly, for that transaction, using a secret key that never leaves the silicon. The card is not a container. It is a participant. Treating it as a container is the single most common misunderstanding among people building their first card integration, and it leads directly to designs that try to "read the card and send the data", which is exactly what EMV was built to make impossible.

Third, "obsolete" and "unused" are different words. It is tempting to read the embossing, the hologram, the signature panel and the stripe as vestigial decoration. Three of the four genuinely are. The stripe is not: it is still the fallback when a chip fails to read, still the only thing some unattended machines can handle, and still in daily use in parts of the world. An issuer removing the stripe is not simplifying a card; it is accepting a measurable rate of declined transactions in exchange. That is a commercial decision with a number attached, and it is why removing it has needed a decade-long timetable.

Fourth, the plastic itself is engineered, not just cut to size. The plain version treats the card body as an inert rectangle. It is a laminated stack of polymer sheets built to survive bending, torsion, being sat on, being left on a dashboard, and thousands of passes through a reader, all while keeping a chip module and a wire antenna intact inside it. There are international standards for how far it may bend before it has failed, and how large a burr the punching die may leave on its edge. Card manufacturing is a precision plastics business with a cryptographic key-management business bolted to the end of it.

With those four corrections in place, the technical version will land properly.

The technical version

The card body: ISO/IEC 7810 and what ID-1 means

The format is called ID-1, and it is defined in ISO/IEC 7810, *Identification cards — Physical characteristics*. The nominal dimensions are 85.60 mm by 53.98 mm with a nominal thickness of 0.76 mm. ISO/IEC 7810:2019 states these as tolerance bands rather than nominals: for an unused ID-1 card, width 85.47 mm to 85.72 mm, height 53.92 mm to 54.03 mm, and thickness 0.68 mm to 0.84 mm. Corners are rounded to a radius of 2.88 mm to 3.48 mm. Edge burrs left by the punching die must not stand more than 0.08 mm above the card surface.

The same standard defines the other card sizes you will meet in this industry: ID-000 at 25.10 mm by 15.10 mm maximum, which is the full-size SIM outline; ID-2 at approximately 105 mm by 74 mm; and ID-3 at approximately 125 mm by 88 mm, which is the passport format. All four share the same thickness band.

ID-1 cards are also specified for mechanical behaviour. Under the standard's bending test, an ID-1 card's deformation must fall between 13 mm and 35 mm: too rigid and it cracks around the chip cavity, too limp and it will not feed through a reader. The test procedures themselves live in a separate series, ISO/IEC 10373-1, *Cards and security devices for personal identification — Test methods — Part 1: General characteristics*, and card service life evaluation is covered by ISO/IEC 24789-1:2024 and ISO/IEC 24789-2:2024.

A conventional payment card is not a solid slab. It is a laminated sandwich: overlay, printed front core, an inner inlay carrying the antenna, printed back core, rear overlay. The magnetic stripe sits on or within the rear overlay. The overlays take the abrasion so the print does not.

The magnetic stripe

Two standards families govern the stripe. ISO/IEC 7811 covers the recording technique: what the stripe is, where it is, and how bits are written to it. ISO/IEC 7813 covers what a financial card must put on tracks 1 and 2. Track 3 content is specified separately in ISO/IEC 4909.

ISO/IEC 7811 is published in parts, and the numbering has a history that catches people out:

Part	Title	Status
7811-1	Embossing	Current
7811-2	Magnetic stripe — Low coercivity	Current
7811-3	Location of embossed characters on ID-1 cards	Withdrawn, folded into Part 1
7811-4	Location of read-only magnetic tracks — Tracks 1 and 2	Withdrawn, folded into Part 2
7811-5	Location of read-write magnetic track — Track 3	Withdrawn, folded into Part 2
7811-6	Magnetic stripe — High coercivity	Current
7811-7	Magnetic stripe — High coercivity, high density	Current
7811-8	Magnetic stripe — Coercivity of 51,7 kA/m (650 Oe)	Current

Part	Title	Status
7811-9	Tactile identifier mark	Current

If a supplier quotes you ISO/IEC 7811-4 for track geometry, they are quoting a withdrawn part. The geometry is still correct; the reference is stale.

Where the stripe is

ISO/IEC 7811-2 places the stripe on the back of the card, no closer than 2.92 mm (0.115 in) to the reference edge, with a minimum length of 82.55 mm (3.250 in) and a minimum width of either 6.35 mm (0.25 in) or 10.28 mm (0.405 in) depending on how many tracks are encoded. A stripe 82.55 mm long on a card 85.60 mm wide is effectively edge to edge.

The three tracks are parallel bands measured from the top reference edge. The figures as originally published in ISO/IEC 7811-4 and 7811-5 are:

Track	Encoded data lies between
Track 1	5.66 mm (0.223 in) and 8.46 mm (0.333 in)
Track 2	8.97 mm (0.353 in) and 11.76 mm (0.463 in)
Track 3	12.52 mm (0.493 in) and 15.32 mm (0.603 in)

Each band is therefore about 2.80 mm wide, with a guard gap of half a millimetre or so between them. Longitudinally, the centreline of the first data bit — the start sentinel — sits 7.44 mm $\pm$ 0.50 mm (0.293 in $\pm$ 0.020 in) from one end of the card, and the last data bit must not come closer than 6.93 mm (0.273 in) to the other. That leaves roughly 71 mm, or 2.80 inches, of usable track. Remember that figure; it is what makes the character limits come out where they do.

How bits are written

The recording method is two-frequency coherent-phase encoding, universally abbreviated F2F and also known as Aiken biphase. There is a flux transition at every bit-cell boundary; a cell containing a `1` has an additional transition in the middle of the cell and a cell containing a `0` does not. ISO/IEC 7811-2 states it plainly: a flux transition occurring between clock transitions signifies a one, and its absence signifies a zero.

That is the property which made magnetic stripes work at all in the 1960s. Because every bit cell begins with a transition, the reader recovers the clock from the data itself, so it does not care how fast you swipe, only that you swipe at a roughly constant speed. A self-clocking code was the price of building a reader with no motor and no precision transport.

Average bit densities are:

Track	Density	Bits per character	Character set	Maximum characters
Track 1	8.27 bits/mm (210 bpi)	7	64 alphanumeric	79
Track 2	2.95 bits/mm (75 bpi)	5	16 numeric	40

Track	Density	Bits per character	Character set	Maximum characters
Track 3	8.27 bits/mm (210 bpi)	5	16 numeric	107

Now the arithmetic. Track 1 at 210 bits per inch across 2.80 inches of usable length yields about 588 bits, which at 7 bits per character is 84 characters — comfortably above the 79-character maximum. Track 2 at 75 bits per inch across the same length yields about 210 bits, which at 5 bits per character is 42 characters, against a 40-character maximum. The limits in the standard are not arbitrary round numbers; they are what the physics allows with margin.

Track 1's 64-character set is the printable ASCII range from `0x20` to `0x5F` with 32 subtracted from each value, giving six data bits per character. Track 2 and track 3 use a 16-character set: the digits `0` to `9` plus the six symbols `: ; < = > ?`, which is ASCII `0x30` to `0x3F` with 48 subtracted, giving four data bits per character. In both cases the data bits are followed by one odd parity bit: ISO/IEC 7811-2 requires that the total number of one-bits recorded for each character, including the parity bit, be odd. Bits within a character are recorded least significant first.

After the end sentinel comes a longitudinal redundancy check character. The rule is column parity: taken across every character on the track including the LRC itself, the count of one-bits in each corresponding bit position must be even. The LRC's own parity bit is still odd parity, computed over the LRC character. Between per-character odd parity and per-column even parity, a single misread bit is always detected and its position can usually be located.

Coercivity

Coercivity is the field strength needed to flip the magnetisation of the stripe's oxide particles, and it distinguishes a stripe that survives a handbag magnet from one that does not. The ISO standards are explicit about *not* specifying it: ISO/IEC 7811-6 states that coercivity influences many of the quantities specified in the document but is not itself specified. What the standards specify instead is the resulting signal amplitude, which is what a reader actually cares about.

The commercial vocabulary is nonetheless universal. Low coercivity ("LoCo") stripes are typically brown; high coercivity ("HiCo") stripes are typically black. The figures usually quoted are around 300 Oe for LoCo and 2,750 Oe or 4,000 Oe for HiCo, and card manufacturers and reader vendors quote them with striking consistency, but no public specification states them; they are trade convention rather than published requirement, which is exactly what one would expect of a family of standards that declines to specify coercivity at all. What is verifiable is that a specific intermediate grade has a standard of its own: ISO/IEC 7811-8 is titled *Magnetic stripe — Coercivity of 51,7 kA/m (650 Oe)*. Payment cards are HiCo; hotel keys and transit tickets are usually LoCo, being disposable.

What is actually written: ISO/IEC 7813

ISO/IEC 7813 defines the data content of tracks 1 and 2 on a financial transaction card. Track 1 uses a format code; format code B is the one the financial industry uses, and format code A is reserved for the card issuer's proprietary use.

Track 1, format B, in order:

Field	Content	Length
Start sentinel	%	1
Format code	B	1
Primary account number	Digits	Up to 19
Field separator	^	1
Cardholder name	Surname, /, given name	2 to 26
Field separator	^	1
Expiration date	YYMM	4, or ^
Service code	Three digits	3, or ^
Discretionary data	Issuer-defined	Balance
End sentinel	?	1
LRC	Check character	1

Total: 79 characters maximum.

Track 2, in order:

Field	Content	Length
Start sentinel	;	1
Primary account number	Digits	Up to 19
Field separator	=	1
Expiration date	YYMM	4, or =
Service code	Three digits	3, or =
Discretionary data	Issuer-defined	Balance
End sentinel	?	1
LRC	Check digit	1

Total: 40 characters maximum.

Two details are routinely got wrong. First, the expiry date and service code are not unconditionally four and three digits: ISO/IEC 7813 permits a placeholder character in place of either, ^ on track 1 and = on track 2, for cards where the field does not apply. A parser assuming fixed offsets after the PAN will break on such a card. Second, the PAN is variable length up to 19 digits; its structure, and the issuer identification number inside it, are governed by ISO/IEC 7812 and are the subject of the next chapter.

The discretionary data field is where the interesting things hide. On most cards it carries a PIN verification field and a card verification value. The value encoded on the stripe is CVV1 or CVC1, and it is not the number printed on the back. Chapter 13 explains why there are four of these values and what attack each one defeats.

The service code

The service code is three digits, sits immediately after the expiry date on both tracks, and tells the terminal what the issuer permits. It is worth learning because it is also one of the inputs to the card verification value algorithm, which means an issuer cannot change it without invalidating every CVV already in the field.

Value	Position 1: interchange and technology	Position 2: authorisation processing	Position 3: allowed services and PIN
0	—	Normal	No restrictions, PIN required
1	International	—	No restrictions
2	International, integrated circuit card	By issuer, online	Goods and services only
3	—	—	ATM only, PIN required
4	—	By issuer, online, unless explicit bilateral agreement applies	Cash only
5	National	—	Goods and services only, PIN required
6	National, integrated circuit card	—	No restrictions, prompt for PIN if PIN pad present
7	Private	—	Goods and services only, prompt for PIN if PIN pad present
9	Test	—	—

So `201` is an international chip card, normal authorisation processing, no terminal restrictions. `221` is the same card with the issuer requiring online authorisation. `121` is a stripe-only international card. `702` is a private card, goods and services only, prompt for PIN where a pad exists.

A first digit of `2` or `6` is the stripe telling a swipe terminal *there is a chip on this card, use it*. That is the mechanism by which a chip-capable terminal refuses a swipe on a chip card and asks the customer to insert instead. It is also the field a fraudster with a stripe writer changes first, which is precisely why the service code is fed into the CVV computation.

Track 3

Track 3 was designed to be read *and written* by the terminal, so an offline ATM could decrement a balance held on the card itself. Its content is specified in ISO/IEC 4909. It is, in payments, dead: almost no issuer encodes it and almost no terminal reads it. It survives in the standards, and in the physical geometry of the stripe, as a reserved band of plastic that nothing uses.

The chip

The contact chip is governed by ISO/IEC 7816. Part 1 covers physical characteristics, part 2 the dimensions and location of the contacts, part 3 the electrical interface and transmission protocols, and part 4 the organisation, security and commands for interchange. EMV Book 1, *Application Independent ICC to Terminal Interface Requirements*, states directly that the EMV specification is based on the ISO/IEC 7816 series and then narrows it: EMV is a profile of 7816, not a superset.

The contacts

ISO/IEC 7816-2 defines eight contacts, C1 to C8, in two rows of four on the front face of the card. Each must present a conductive rectangle of at least 2.0 mm by 1.7 mm. Horizontally, measured from the left edge of the card, the four columns occupy bands at 19.23 mm to 20.93 mm, 21.77 mm to 23.47 mm,

24.31 mm to 26.01 mm, and 26.85 mm to 28.55 mm. Vertically, the upper row's required conductive area extends to 10.25 mm from the upper edge and the lower row's begins at 12.25 mm.

The assignments, as EMV Book 1 sets them out:

Contact	Signal	EMV use
C1	VCC	Supply voltage
C2	RST	Reset
C3	CLK	Clock
C4	Auxiliary	Not used; need not be physically present
C5	GND	Ground
C6	VPP	Not used; EMV cards shall not require VPP
C7	I/O	Serial input/output, half duplex
C8	Auxiliary	Not used; need not be physically present

Note C6. On the earliest smart cards, VPP supplied the elevated programming voltage needed to write EEPROM. Modern chips generate that voltage internally with a charge pump, so EMV forbids relying on it. If you have wondered why the gold plate has eight pads when only five do anything, that is the answer: two are reserved and one is a fossil. There is a second fossil: before 1990 an alternative contact position was in use, and ISO/IEC 7816-2 still discusses electrical isolation of the zones so a terminal can accept cards built to either convention.

Electrical behaviour

EMV Book 1 specifies that the card shall operate at 5 V $\pm$ 0.5 V DC with a maximum current requirement of 50 mA, and that the terminal shall supply 5 V $\pm$ 0.4 V DC with a steady-state output current in the range 0 to 55 mA. The clock frequency must be in the range 1 MHz to 5 MHz, and once the card has answered to reset the terminal must hold the clock within $\pm$1% of the frequency used during that answer to reset, for the remainder of the card session.

Two transmission protocols are defined: T=0, a character-oriented asynchronous half-duplex protocol, and T=1, a block-oriented asynchronous half-duplex protocol. EMV requires that a card support one of them and that a terminal support both. This asymmetry is deliberate: it lets issuers choose a chip platform without fragmenting the terminal estate.

What the chip holds

Data on an EMV chip is organised as BER-TLV data objects, each identified by a tag. The identity-carrying ones are:

Tag	Name
`4F`	Application Identifier (AID) — card
`50`	Application Label
`57`	Track 2 Equivalent Data
`5A`	Application Primary Account Number (PAN)
`5F20`	Cardholder Name
`5F24`	Application Expiration Date
`5F25`	Application Effective Date
`5F28`	Issuer Country Code

Tag	Name
5F30	Service Code
5F34	Application PAN Sequence Number
84	Dedicated File (DF) Name
87	Application Priority Indicator
8C	Card Risk Management Data Object List 1 (CDOL1)
8D	Card Risk Management Data Object List 2 (CDOL2)
9F1F	Track 1 Discretionary Data
9F20	Track 2 Discretionary Data

Tag 57, Track 2 Equivalent Data, is the one that causes confusion. It has the same field order as the magnetic stripe's track 2 — PAN, separator, expiry date, service code, discretionary data — but it is packed as compressed numeric, two digits per byte, so the separator is the hexadecimal nibble D rather than the character =, and the final nibble is padded with F if the digit count is odd. It is a track 2 record in shape only. Its discretionary data carries the chip's own card verification value, conventionally set to a different value from the stripe's, so that lifting tag 57 off a chip and encoding it onto a blank stripe produces a card that fails verification.

Note also 5F34, the PAN sequence number. When a bank reissues a card after loss without changing the PAN, this is the field that distinguishes the new plastic from the old. Reconciliation systems keying on PAN alone will merge two physically distinct cards, a bug that only surfaces during a dispute.

Everything above is data the chip will hand over. The keys it uses to generate cryptograms are not in that list, because they are not readable by any command. That is the point of the chip, and the subject of chapter 15.

The contactless antenna

The radio layer is ISO/IEC 14443, *Cards and security devices for personal identification — Contactless proximity objects*, in four parts: part 1 physical characteristics, part 2 radio frequency power and signal interface, part 3 initialisation and anticollision, part 4 transmission protocol. In 14443's vocabulary the card is a PICC (proximity integrated circuit card) and the reader is a PCD (proximity coupling device).

The physical parameters from ISO/IEC 14443-2 are:

Parameter	Value
Carrier frequency	13.56 MHz ± 7 kHz
Minimum operating field, Hmin	1.5 A/m rms
Maximum operating field, Hmax	7.5 A/m rms
Initial bit rate, both directions	fc/128, approximately 106 kbit/s
Subcarrier	fc/16, approximately 847 kHz

Two signalling types are defined and both are in use on payment cards. Type A uses 100% amplitude-shift keying with modified Miller coding from reader to card, and on-off keying of an 847 kHz subcarrier with Manchester coding from card to reader. Type B uses roughly 10% ASK from reader to card, with the modulation index required to lie between 8% and 14%, encoded NRZ-L, and binary phase-shift keying of the subcarrier from card to reader. A terminal must support both; a card implements one.

The 100% modulation of Type A is worth pausing on. Switching the field fully off is a blunt instrument, and during those gaps the card runs on whatever charge it has stored in its supply capacitor. Type B's shallower modulation never removes power. Both survive because both were deployed before either won.

The card side of this is a coil: several turns of fine enamelled copper wire ultrasonically embedded into an inlay layer of the card body, or an etched or printed conductive track on a substrate. ISO/IEC 14443-1 defines antenna size classes; a full-size payment card carries a Class 1 antenna, the largest, running close to the perimeter. The chip module connects to it either galvanically, by welding or soldering the wire ends to terminals on the underside of the module, or inductively, by putting a small coil on the module which couples to a matching loop in the card antenna. The inductive approach — Infineon's Coil on Module, and equivalents from other vendors — removes two solder joints from the most mechanically stressed part of the card.

On top of 14443 sits the EMV contactless specification set. It is published as *EMV Contactless Specifications for Payment Systems* in several books:

Book	Title
A	Architecture and General Requirements
B	Entry Point Specification
C-1 to C-8	Kernel Specifications
D	Contactless Communication Protocol Specification

Book D is the EMV profile of ISO/IEC 14443. Book B, Entry Point, is the logic that decides which kernel to invoke based on the application identifier the card presents. The kernels map to schemes as follows: Kernel 2 (Book C-2) for Mastercard AIDs, Kernel 3 for Visa, Kernel 4 for American Express, Kernel 5 for JCB, Kernel 6 for Discover, and Kernel 7 for UnionPay. Kernel 1 was removed from Book A as of version 2.7. Kernel 8, Book C-8, was published by EMVCo on 5 October 2022 as a scheme-neutral kernel intended to coexist with the legacy kernels and, over time, reduce the number terminals must carry.

That list is the honest answer to "why is contactless acceptance so complicated". A terminal accepting six brands runs six transaction state machines behind one tap.

Embossing

ISO/IEC 7811-1 governs embossing. It specifies three typefaces: ISO 1073-1 OCR-A in sizes I and IV, ISO 1073-2 OCR-B in sizes I and IV, and the type font Farrington 7B — the slightly squarish face you will recognise from every embossed card you have ever held.

Machine-readable embossed characters are 4.32 mm (0.170 in) high with a centre-to-centre spacing of 3.63 mm $\pm$ 0.15 mm (0.143 in $\pm$ 0.006 in). Visually readable characters, used for name and address lines, are set at 2.54 mm $\pm$ 0.15 mm (0.100 in $\pm$ 0.006 in) pitch.

Relief height — how far the character stands proud of the surface — is specified twice, once for unused cards and once for cards returned from the field, because embossing flattens with use:

Character type	Unused card	Returned card
Machine readable	0.40 mm to 0.48 mm	0.30 mm to 0.48 mm
Visually readable	0.36 mm to 0.46 mm	0.26 mm to 0.46 mm

The standard defines two embossing areas. Area 1 is the identification number line and accommodates a maximum of 19 character positions — the same 19 as the maximum PAN length, which is not a coincidence. Area 2 is the name and address area and accommodates four lines of 27 characters each.

Embossing exists because of the manual imprinter, and its machine-readability requirement exists because for a period the industry expected imprinted vouchers to be read back optically. Neither use case survives. Most cards issued today are flat: the PAN is applied by laser engraving, thermal transfer or indent printing, and a growing number carry no PAN on the front at all. Embossing on a modern card is a design choice about how the card feels in the hand, and usually a premium one.

The hologram

The optical variable device was introduced in the 1980s for one reason: a shopkeeper needed a counterfeit test that required no equipment, no phone call, and no training beyond "tilt it". It worked because holographic origination equipment was, at the time, capital-intensive and rare.

The schemes have been steadily reducing it. Visa's guidance is that hologram options were consolidated to a single design: vendors ceased production of the standard and mini dove designs on 31 December 2020, and from 1 January 2021 the updated Visa Product Brand Standards removed those options, leaving the silhouetted dove as the sole hologram. Cards carrying the Premium Visa Brand Mark do not require a dove hologram at all, because the brand mark itself carries the anti-counterfeiting security features.

The direction of travel is the same everywhere, and the reason is straightforward. A security feature that only works when a human inspects it loses its value in proportion to how many transactions involve a human inspecting anything. In a market where the overwhelming majority of face-to-face transactions are a tap and the rest are card-not-present, the hologram is defending a shrinking perimeter.

The signature panel

The panel is a printed receptive coating — a surface that ink will bind to, which the card's glossy overlay will not — usually laid over a tint or repeating pattern chosen so that an attempt to erase and re-sign leaves a visible mark.

Mastercard announced on 17 October 2017 that signatures would no longer be required at the point of sale in the United States and Canada from April 2018, and subsequently made the change global. From April 2019, Mastercard issuers globally were no longer required to include a signature panel on the back of Mastercard products. The other major schemes moved on comparable timelines.

The panel persists on many cards for two reasons that have nothing to do with payments: some jurisdictions and some non-payment uses of the card still expect a signature, and card designs are expensive to change so issuers batch such changes. Where the panel remains, it typically carries the printed card verification value — CVV2, CVC2 or CID depending on scheme — which chapter 13 distinguishes from the three other things called "the security code".

Tactile identifiers

ISO/IEC 7811-9 specifies a tactile identifier mark: a raised feature that lets a cardholder who cannot see the card determine what it is by touch.

The best-known implementation is Mastercard's Touch Card, announced on 25 October 2021. It places a notch on the short side of the card, with the shape encoding the product type: a squarish notch for credit, a rounded notch for debit, and a triangular notch for prepaid. The notch also tells the cardholder

which way round to insert the card, which matters when the chip contacts are on one face and one end only.

This is the one feature on the card that is getting more common rather than less, and the only one whose reader is a person rather than a machine.

Why all of this coexists

The honest answer is that a payment card is a network device with a five-year life, and the network it talks to has a thirty-year life. Every acceptance device in the world constrains what a card may stop carrying, and no issuer controls that estate.

The magnetic stripe is the clearest case, because for once there is a published timetable. Mastercard announced on 23 August 2021 that the stripe would no longer be required on new cards in selected regions including Europe from 2024; that the United States would follow from 2027; that from 2029 no new Mastercard cards would be issued with a magnetic stripe; and that by 2033 no Mastercard cards with stripes would remain in circulation. That is a twelve-year plan to remove a feature already technically obsolete in Europe.

Read that timetable and you can reconstruct the economics. Between 2021 and 2033 the cost of carrying the stripe — a few pence of tape and a persistent cloning risk — was judged lower than the cost of the transactions that would fail without it. The stripe is not there because anybody wants it. It is there because removing it costs more than keeping it, and the crossover point arrives at different dates in different regions. As of 2026 the early milestones are in force in Europe; the later ones remain a stated intention, and the industry has revised such timetables before.

The same logic, with smaller numbers, explains the embossing, the hologram and the signature panel. Each survives in the space between “no longer required” and “actively removed”, and that space is measured in years.

Manufacturing and personalisation in outline

Card production splits cleanly into two phases with different security regimes. Card manufacturing produces cards that are identical to one another. Personalisation makes one card belong to one cardholder. Everything expensive about card security governance lives at the boundary between them.

Manufacturing

Artwork is separated and printed onto polymer sheets, many cards to a sheet — typical layouts are 21-up, 24-up or 48-up depending on the plant. Offset lithography remains dominant in secure card printing, with screen printing for metallic and opaque effects and digital printing growing for short runs.

The printed sheets are collated into a stack: overlay, front core, inlay, back core, overlay, with the antenna in the inlay. The stack is tacked together, usually by ultrasonic or thermal welding at the edges, so it cannot shift. Lamination follows: the stack goes between polished steel plates in a press, under heat and pressure, then cools under pressure. The polished plates give the card its finish; the cooling stage under load is what stops it curling.

The laminated sheet is then die-punched into individual ID-1 cards, which is where the 0.08 mm burr limit and the 2.88 mm to 3.48 mm corner radius are either achieved or missed. The magnetic stripe is applied either as part of the overlay film or hot-stamped as a tape after lamination. The hologram is hot-stamped.

For a chip card, a stepped cavity is then milled into the card face: a wide, shallow step to seat the contact plate flush with the surface, and a narrower, deeper pocket beneath it for the die and its bond wires. The module is placed over a hot-melt adhesive film and pressed by a heated die. On a dual-interface card, the module's antenna terminals are joined to the embedded card antenna at this stage, or coupled inductively. Every card is then tested electrically, contact and contactless, and visually.

Personalisation

Personalisation has a logical half and a physical half.

The logical half begins with data preparation inside a hardware security module. The issuer holds an Issuer Master Key; for each card the HSM derives card-specific keys by diversifying that master key over the PAN and the PAN sequence number, so compromising one card yields no information about any other. The output is a personalisation data set: application data, derived keys, the offline PIN, and the private key material used for offline data authentication.

Writing that data into the chip happens over a secure channel. The commands and structures are defined by the EMV Card Personalisation Specification, the channel by GlobalPlatform's secure channel protocols — SCP02, triple-DES-based, deprecated but still widely deployed, and SCP03, AES-based. Where the personalisation bureau is not the key owner, secrets travel under separate transport keys: a data transport key, a PIN transport key and a key transport key. The bureau's HSM decrypts under the transport keys and re-encrypts under the card's session keys, so plaintext PINs and card keys never exist outside an HSM.

The physical half is embossing or laser engraving of the PAN, name and expiry; thermal transfer or indent printing of the CVV2; encoding of the magnetic stripe; and attachment of the card to its carrier for mailing. The PIN mailer, where one is produced, goes separately and is generated inside an HSM without any human seeing the value.

Who polices this

The applicable standard is the PCI Card Production and Provisioning Security Requirements, published by the PCI Security Standards Council in two parts, Physical and Logical. Version 3.0 was announced on 13 January 2022 and covers card manufacturing, magnetic-stripe encoding and embossing, personalisation, chip initialising and embedding, storage, shipping and mailing, and the over-the-air provisioning of card credentials to devices.

One point matters commercially and is frequently misunderstood: while the requirements are maintained by PCI SSC, compliance is managed directly by the payment brands. A vendor works to the PCI document but is assessed and approved through each scheme's own programme.

■ One card, summarised

Feature	Governing standard	What it holds	Status
Card body	ISO/IEC 7810 (ID-1)	Nothing; it is the substrate	Current
Embossing	ISO/IEC 7811-1	PAN, name, expiry, in relief	Obsolete, common
Magnetic stripe	ISO/IEC 7811-2, -6, -7, -8	Tracks 1 and 2 per ISO/IEC 7813	Obsolete, in use, scheduled for removal
Track 3	ISO/IEC 4909	Nothing, in practice	Dead

Feature	Governing standard	What it holds	Status
Contact chip	ISO/IEC 7816, EMV Books 1 to 4	Application data and keys	Current, primary
Contactless antenna	ISO/IEC 14443, EMV Books A to D	Same chip, radio interface	Current, primary
Hologram	Scheme brand standards	Visual authenticity	Being reduced
Signature panel	Scheme brand standards	Signature, printed CVV2	Optional since 2019
Tactile mark	ISO/IEC 7811-9	Product type by touch	Growing

Six ways of saying one thing, one of which does the work.

The next chapter takes the thing itself — the number, all nineteen possible digits of it — and shows what each part of it means, who assigned it, and how a terminal knows in a millisecond that you have mistyped it.

11.98 Common wrong ideas

Wrong: The chip and the stripe carry the same message in two formats. **Right:** They carry deliberately different values in one field, so chip data copied onto a blank stripe is declined — a scheme rule, not an accident of implementation.

Wrong: The chip is a container you read the account number out of. **Right:** It is a participant; what the terminal needs is computed freshly for that transaction using a key that never leaves the silicon, which is precisely what EMV was built to require.

Wrong: The stripe, the embossing, the hologram and the signature panel are equally vestigial. **Right:** Three of the four are, but the stripe is still the fallback when a chip fails to read and still the only thing some unattended machines handle, so removing it buys a measurable rate of declines.

Wrong: The card body is an inert rectangle cut to size. **Right:** It is a laminated polymer stack with tolerance bands, a specified bending deformation between 13 mm and 35 mm, and a limit of 0.08 mm on the burr the punching die may leave.

Wrong: ISO/IEC 7811-4 is the reference to quote for track geometry. **Right:** Parts 3, 4 and 5 were withdrawn and folded into Parts 1 and 2; the geometry is still correct but the citation is stale.

Wrong: LoCo at 300 Oe and HiCo at 2,750 or 4,000 Oe are figures from the standards. **Right:** ISO/IEC 7811-6 explicitly declines to specify coercivity and specifies the resulting signal amplitude instead; those numbers are trade convention.

Wrong: Expiry and service code are always four and three digits, so track 2 can be parsed at fixed offsets after the account number. **Right:** ISO/IEC 7813 permits a placeholder character in place of either, and the account number itself is variable up to nineteen digits.

Wrong: Tag `57` is the stripe's track 2, so the two can be compared as strings. **Right:** It is compressed numeric at two digits per byte, with the hexadecimal nibble `D` as separator and `F` padding, and its discretionary data carries a different verification value on purpose.

Wrong: One account number identifies one piece of plastic. **Right:** A reissue after loss can carry the same number, distinguished only by tag `5F34`, so reconciliation keyed on the number alone merges two physically distinct cards and the bug surfaces during a dispute.

Wrong: The gold plate has eight contacts because eight are needed. **Right:** Only five do anything under EMV; C4 and C8 are auxiliary and need not be physically present, and C6's programming voltage is a fossil that modern chips generate internally.

11.99 Chapter summary in 20 lines

1. A payment card carries one piece of information — which account to charge — in six different physical forms, three of which are obsolete and all six of which are still present.
2. It works like a note left in several formats at once, because the acceptance estate cannot be upgraded on your behalf before you next buy a sandwich.
3. The card body is ID-1 under ISO/IEC 7810, nominally 85.60 mm by 53.98 mm and 0.76 mm thick, specified as tolerance bands with corner radii and a burr limit.
4. It is not a slab but a laminated sandwich of overlay, front core, antenna inlay, back core and rear overlay, with a bending range specified so it neither cracks round the chip nor jams in a reader.
5. The magnetic stripe carries three parallel tracks recorded in self-clocking F2F encoding, so the reader recovers its clock from the data and does not care how fast you swipe.
6. Track 1 holds 79 alphanumeric characters, track 2 holds 40 numeric characters, and track 3 holds 107 and is, in payments, dead.
7. Those limits are what the physics allows with margin: 210 bits per inch across 2.80 usable inches at seven bits per character gives 84, against a 79-character maximum.
8. Per-character odd parity combined with per-column even parity in the check character means a single misread bit is always detected and can usually be located.
9. The standards specify signal amplitude rather than coercivity, so the familiar LoCo and HiCo figures are trade convention rather than published requirement.
10. ISO/IEC 7813 fixes what tracks 1 and 2 must carry, and the discretionary data field is where the PIN verification field and the stripe's own card verification value hide.
11. The three-digit service code tells the terminal what the issuer permits, and a leading 2 or 6 is the stripe telling a swipe terminal that there is a chip and it should be used.
12. Because the service code feeds the card verification value computation, a fraudster who changes it invalidates the value, and an issuer cannot change it without invalidating every value already in the field.
13. The contact chip is governed by ISO/IEC 7816, of which EMV is a profile rather than a superset, with eight contacts of which five do anything.
14. Chip data is a set of BER-TLV objects, and tag 57 reproduces track 2 in compressed numeric with a deliberately different verification value, so a chip-to-stripe copy fails verification.
15. The keys that generate cryptograms appear in no tag list, because no command will read them out, and that is the point of the chip.
16. The contactless layer is ISO/IEC 14443 at 13.56 MHz, with two signalling types that every terminal must support and each card implements one of.
17. Above it sit the EMV contactless books with a kernel per scheme, which is the honest answer to why contactless acceptance is complicated: six brands mean six state machines behind one tap.
18. Embossing, the hologram and the signature panel each survive in the space between "no longer required" and "actively removed", and that space is measured in years.
19. The tactile identifier mark is the one feature becoming more common rather than less, and the only one whose reader is a person rather than a machine.
20. The stripe is scheduled out over a twelve-year timetable to 2033 because the cost of carrying it was judged lower than the cost of the transactions that would fail without it, which is the economics of every layer on the card.

Chapter sources: ISO/IEC 7810:2019; ISO/IEC 7811 parts 1, 2, 4, 5, 6, 8, 9; ISO/IEC 7813:2006 and earlier editions; ISO/IEC 4909; ISO/IEC 7816-2 (1999 and 2007 editions); ISO/IEC 14443-2; ISO/IEC 10373-1; ISO/IEC 24789-1:2024; EMV Book 1, ICC to Terminal Interface Requirements; EMV Contactless Specifications for Payment Systems, Books A and D; EMVCo tag references and the October 2022 Kernel 8 announcement; PCI SSC Card Production and Provisioning Security Requirements v3.0; Mastercard and Visa published brand and product announcements.

2

CHAPTER

The Number

12.0 What this chapter gives you

1. You will be able to verify a card number by hand, on paper, in about ninety seconds, and to build the check digit rather than merely test it.
2. You will be able to say exactly what a commercial BIN lookup can and cannot honestly tell you.
3. You will be able to explain why hard-coding a six-digit issuer identifier is now a defect, and why six-digit and eight-digit identifiers coexist rather than one having replaced the other.
4. You will be able to say why the schemes route on a nine-digit or eleven-digit account range rather than on the issuer identifier, and what a six-digit classification gets structurally wrong.
5. You will be able to demonstrate the check digit's two known blind spots by hand, and derive why 0 and 9 is the only adjacent transposition it misses.
6. You will be able to explain why anchoring the doubling at the rightmost digit is the only safe implementation, and why anchoring from the left makes every American Express card appear invalid.
7. You will be able to account for the four representations of one expiry date across the printed card, the stripe, the chip and the authorisation message, and say which one carries a day.
8. You will be able to say why an expired card does not fail at the chip, and what the terminal verification results actually record.
9. You will be able to give the five separate forces that make card number length vary, without conflating them.
10. You will be able to store, mask and truncate an account number correctly, and explain why masking and truncation are different controls with different limits.

Take a card out of your wallet and look at the long number on the front of it. Sixteen digits, usually, in four groups of four. Most people assume it is a serial number — that somewhere there is a very large list, and your card is entry number four trillion and something on it.

It is not that. Almost none of those digits are about you. The number is an address. It is structured, it is hierarchical, and it is read from the left in exactly the way a postal address is read from the bottom up: which country, which town, which street, which house. The first digits say which institution in the world is answerable for this card. The middle digits say which account at that institution. And the final digit is not information at all — it is a checksum, a digit whose only job is to catch the mistakes made by human beings and card readers.

This matters more than it sounds. When a terminal in a petrol station in Carlisle reads a card issued by a bank in Manila, nothing in that terminal knows anything about Philippine banking. What it knows is how to look at the leading digits, find a matching row in a table it was given, and work out where to send the message. The number is the routing key for the entire global card system, and everything in

the next several chapters — the cryptogram, the authorisation message, the interchange fee — depends on the leading digits of the number being correct and being interpreted correctly.

This chapter takes the number apart. By the end of it you will be able to verify a card number by hand, on paper, in about ninety seconds, and you will know exactly what a commercial "BIN lookup" can and cannot honestly tell you.

The plain version

Think of a telephone number.

If someone reads you a number starting `+44 20`, you already know two things before you have heard the rest. `44` means the United Kingdom. `20` means London. You have not been told whose phone it is, or whether they are in, or whether they will answer. But you know where to send the call, and that is enough to get started.

A card number works the same way, with one extra trick at the end.

The first eight digits say **which bank or card issuer** this card belongs to. Not which country, exactly, and not which branch — which *institution*, out of the tens of thousands in the world that issue cards. Those eight digits are handed out by a registry, the way telephone dialling codes are handed out, so that no two issuers can ever claim the same ones.

The digits after that say **which account** at that institution. That part is the issuer's own business. It might be sequential. It might be random. Nobody outside the bank needs to know which, and in fact banks deliberately make it hard to guess.

And the very last digit is a **check digit**. It carries no information about you, your bank or your account. It exists solely so that if somebody reads the number out wrong, or types it wrong, or a card reader misreads a smudged digit, the mistake is caught immediately instead of sending a payment to a stranger.

Why a check digit is worth a whole digit

Imagine reading a sixteen-digit number aloud over a bad phone line. Two things go wrong constantly. Either one digit gets heard wrong — a five becomes a nine — or two neighbouring digits get swapped, because human beings swap things. `...4536...` becomes `...4356...` and nobody notices.

You could catch this by reading the number twice. That is slow and people cheat. So instead the last digit is made to *depend* on all the others, by a rule that both ends of the line agree on. If the sixteenth digit does not match what the first fifteen say it should be, something has gone wrong, and you know before you send any money anywhere.

Giving up one digit out of sixteen costs you a tenth of your available numbers. Catching essentially every single-digit typo and nearly every swap is worth vastly more than that.

Working it out by hand

The rule is called the Luhn formula, and it is genuinely simple enough to do at a kitchen table. Here it is on a real card number — a test number published by payment gateways for exactly this purpose, so it belongs to nobody:

4012 8888 8888 1881

Write it out with the positions numbered from the left.

Step one: **double every second digit, starting from the first one on the left.** (That works because this number has sixteen digits, an even count. Come back to that in a moment.)

Step two: **if a doubled digit comes out bigger than nine, subtract nine.** So an 8 doubles to 16, and 16 minus 9 is 7. A 6 doubles to 12, and 12 minus 9 is 3.

Step three: **add up everything.**

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Digit	4	0	1	2	8	8	8	8	8	8	8	8	1	8	8	1
Doubled?	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no
Becomes	8	0	2	2	7	8	7	8	7	8	7	8	2	8	7	1

Add the bottom row: 8 + 0 + 2 + 2 + 7 + 8 + 7 + 8 + 7 + 8 + 7 + 8 + 2 + 8 + 7 + 1.

Take it in stages so you can check yourself: the first four come to 12. The next four (7, 8, 7, 8) come to 30, running total 42. The next four (7, 8, 7, 8) come to 30 again, running total 72. The last four (2, 8, 7, 1) come to 18. **Total: 90.**

Ninety ends in a zero. That is the whole test. **If the total ends in zero, the number is well formed. If it does not, somebody made a mistake.**

Watch it catch a mistake

Now break the number deliberately. Swap the thirteenth and fourteenth digits, so `...1881` becomes `...8118`. That is exactly the sort of slip a tired person makes.

Position 13 was a 1, doubled to 2. It is now an 8, which doubles to 16, minus 9 is 7. That is five more than before. Position 14 was an 8, not doubled, contributing 8. It is now a 1, contributing 1. That is seven less than before. Net change: minus two. The new total is 88.

Eighty-eight does not end in zero. **Rejected**, in one pass, before any message goes anywhere.

Try a single-digit error instead. Change the fifth digit from 8 to 3. It is a doubling position, so where it used to contribute 7 (from 16 minus 9) it now contributes 6. Total drops to 89. **Rejected** again.

Building the check digit rather than testing it

The bank does the same arithmetic backwards. It decides the first fifteen digits — `401288888888188` — and then works out what the sixteenth must be.

Run the sum on those fifteen digits with a zero standing in for the check digit. Everything above stays the same except position 16, which is now 0 instead of 1, so the total is 89 instead of 90.

Whatever the check digit is, it sits in a non-doubling position, so it just gets added. You need a total ending in zero. 89 needs 1 more. **The check digit is 1** — which is precisely what is printed on the card.

The general rule: take the total ending in whatever it ends in, and the check digit is the number that brings it up to the next multiple of ten. If your running total ended in 4, the check digit is 6. If it ended in 0 already, the check digit is 0.

Two warnings you can act on tonight

First, you can verify your own card this way in about a minute and a half, on paper. Do it on paper. Do not type your card number into a website that offers to "check if your card number is valid", because such a site has just been given your card number and you have learned nothing you could not have worked out with a pencil.

Second — and this is the part people get wrong — **passing this test does not mean the card exists.** It means the number is spelled correctly. A number can pass the Luhn test perfectly and correspond to no account anywhere on earth. It is a spell-checker, not a lock.

The date

The other number on the front is the expiry date, printed as month and year: `09/28` means September 2028.

Cards expire for boring reasons rather than sinister ones. The plastic wears, the chip contacts corrode, the magnetic stripe demagnetises in a pocket next to a phone, and the bank wants a periodic excuse to check that you still live where it thinks you live. A typical card runs three to five years.

The single fact people get wrong: a card marked `09/28` is good **through the whole of September 2028**, right up to the last second of the 30th. It stops working on 1 October. "Valid thru" means exactly that.

Why some cards are shorter

If you have an American Express card in the drawer, it has fifteen digits, not sixteen, grouped 4-6-5 rather than 4-4-4-4. Diners Club cards have historically had fourteen. Some debit cards abroad have nineteen.

The reason is history rather than logic. The card schemes invented their own numbering before there was an international standard, the standard was written afterwards to accommodate what already existed, and it set a ceiling — no more than nineteen digits — rather than a fixed size. Sixteen won because it was what the two biggest schemes chose, and because acceptance equipment built in the 1980s got used to it.

The arithmetic still works on all of them. But note the trap: because the doubling is anchored at the check digit on the *right*, a fifteen-digit number doubles a different set of positions counted from the left than a sixteen-digit number does. On a fifteen-digit Amex number you double the second, fourth, sixth digit and so on from the left. Get that backwards and every American Express card in the world will appear invalid to your software — which is, reliably, the first bug in every hand-written card validator ever shipped.

Where the plain version stops being true

The telephone-dialling-code picture is the right shape and it is wrong in four specific places, each of which costs money when a working system relies on it.

The issuer prefix is not six digits, and hard-coding six is now a defect

For roughly forty years the leading **six** digits were the issuer's identifier, and an entire generation of payment software, database schemas, fraud rules and merchant reports was written on that assumption. The relevant international standard changed in January 2017. The issuer identifier is now **eight** digits, and both Visa and Mastercard made support for eight-digit issuer identifiers mandatory for their participants from their April 2022 releases.

The plain version above already used the correct figure, which is why it is worth stating the correction loudly: any system built before about 2019 almost certainly assumes six, and a great deal of tooling still in production does. Six-digit and eight-digit identifiers now coexist — the old ones were not withdrawn, they were subdivided — so code that assumes either number in isolation is wrong.

Even eight digits is not how the network actually routes

Here is the correction that surprises practitioners. The issuer identifier tells you the institution. It does not tell you the **product**, and the product is what determines the interchange fee, the debit-or-credit distinction, whether the card is commercial, and which processing rules apply.

For that, the schemes route on something finer than the issuer identifier: an **account range**. Visa's is nine digits; Mastercard's is eleven. A single issuer identifier is deliberately carved into multiple account ranges so that one bank can put its consumer debit, premium credit and corporate cards on the same identifier with different behaviour.

Visa has told its acquirers explicitly, in its own operations bulletins, to process on the complete nine-digit account range record rather than the first six digits. A merchant system that classifies a card by its leading six digits is not merely out of date; it can and does classify cards into the wrong fee category.

Luhn is arithmetic, not security, and it never was

The check digit's rule was patented by an IBM engineer in the 1950s, has been published for over sixty years, and is in the public domain. It is in every standard, every textbook and roughly ten thousand GitHub repositories. It is designed to catch accidents, and it does that superbly. It is not designed to resist an adversary, and against one it is worthless — generating millions of Luhn-valid card numbers is a first-week programming exercise, and "card testing" attacks do exactly that, then hunt for the live ones by firing small authorisations at merchants.

There is a second, subtler failure that follows from the arithmetic itself. Luhn catches every single-digit error and every adjacent transposition **except one pair**, and it misses certain twin-digit substitutions. You can prove both by hand.

Take the four-digit number `1099`. Doubling the first and third: 1 becomes 2, 0 stays 0, 9 becomes 18 which reduces to 9, 9 stays 9. Total $2 + 0 + 9 + 9 = 20$. Valid. Now swap the middle two digits to get `1909`. Doubling the first and third: 1 becomes 2, 9 stays 9, 0 becomes 0, 9 stays 9. Total $2 + 9 + 0 + 9 = 20$. **Also valid.** Swapping an adjacent `0` and `9` is invisible to Luhn, always, everywhere, on every card number in the world.

The same trick works on twins. `1222` gives $2 + 2 + 4 + 2 = 10$, valid. Replace the middle `22` with `55` to get `1552`, and you get $2 + 5 + 1 + 2 = 10$, also valid. The pairs that collide this way are 22 against 55, 33 against 66, and 44 against 77.

None of this makes Luhn a bad check. It makes it a *known* check with *known* blind spots, which is a different thing from a guarantee.

The number does not identify a card, and may not identify a bank account

Three separate collapses of the plain version here.

A single card number can belong to several pieces of plastic. Replacement cards, and additional cardholders on the same account, may carry the same number distinguished only by a separate sequence number that is not printed on the card at all and lives only in the chip.

Worse for anyone building a lookup: the number in front of you may be a **network token** rather than a card number. When a card is added to a phone wallet or stored by a large merchant, the scheme can

issue a substitute number, of the same length, that passes the Luhn test and looks in every respect like a card number, but resolves to the real one only inside the scheme's own vault. This is the subject of a later chapter. Its consequence here is immediate: some of the numbers flowing through your systems are not the numbers printed on any card, and a lookup on their leading digits does not describe the issuer's card product.

And finally, the number does not identify a *bank account*. A card is an access mechanism attached to an account. The mapping is the issuer's private business, it is not one-to-one in either direction, and nothing in the number exposes it.

The technical version

ISO/IEC 7812-1: what the standard actually specifies

The governing document is **ISO/IEC 7812-1, "Identification cards — Identification of issuers — Part 1: Numbering system"**. The current text is the fifth edition, published January 2017, confirmed on systematic review in 2022 and still current as of writing in 2026. Its companion, **ISO/IEC 7812-2**, covers application and registration procedures rather than format.

The standard defines the **Primary Account Number** (**PAN**) as three concatenated components:

Component	Length	Assigned by
Issuer Identification Number (IIN)	8 digits	Registration Authority
Individual account identifier	1 to 10 digits	The issuer
Check digit	1 digit	Computed

The maximum total PAN length is **19 digits**, unchanged by the 2017 revision. The fifth edition's substantive change was to fix the IIN at eight digits rather than six. The mechanics of that change matter: existing six-digit IINs were not invalidated, they were **converted into a block of one hundred eight-digit IINs**. The six-digit IIN `412345` becomes the eight-digit block `41234500` through `41234599`, all still belonging to the same institution. This is why six-digit and eight-digit identifiers coexist rather than one having replaced the other.

Two further consequences are stated in the standard and are frequently missed. Issuers holding eight-digit IINs are required to issue a **minimum PAN length of ten digits** — you cannot have an eight-digit issuer identifier, a check digit and no account identifier at all. And with an eight-digit IIN and a nineteen-digit ceiling, the individual account identifier can be at most ten digits.

The Registration Authority for ISO/IEC 7812 is the **American Bankers Association**; administration passed to CUSIP Global Services acting on the ABA's behalf, reported as of February 2024.

The Major Industry Identifier

The first digit of the IIN is the **Major Industry Identifier** (**MII**), a classification inherited from the earliest days of the standard.

MII	Category
0	ISO/TC 68 and other industry assignments
1	Airlines
2	Airlines and other industry assignments

MII	Category
3	Travel and entertainment
4	Banking and finance
5	Banking and finance
6	Merchandising and banking/finance
7	Petroleum
8	Healthcare, telecommunications and other industry assignments
9	National assignment

Treat this table as archaeology, not as operational data. The MII is a historical classification with almost no present-day predictive value, and the clearest proof is Mastercard's own expansion: when Mastercard ran short of numbers under its traditional 51–55 prefixes it was allocated the range **222100–272099**, announced in November 2014 with participant systems required to be ready by October 2016. Every one of those cards is a mainstream consumer payment card whose MII digit is 2, a value the table above labels "airlines". Any code branching on the first digit to guess what kind of card it is holding is guessing.

The one MII with a live structural rule is 9, reserved for assignment by national standards bodies, where the convention is that the three digits following the 9 are the ISO 3166 numeric country code and the remainder of the number is defined nationally. In practice the national card schemes largely did not take up this space, choosing ranges allocated to them under the banking MIIs instead.

BIN, IIN and account range

The industry says **BIN**, for Bank Identification Number. The standard says **IIN**. For payment cards they denote the same leading digits; BIN is the older, commercial term and it has stuck so firmly that the schemes themselves use it in their bulletins. Where the two words genuinely diverge is scope: IIN is a general card-numbering concept covering non-payment cards too, whereas BIN in scheme usage carries specific commercial and licensing meaning.

The migration, scheme by scheme, as documented in the schemes' own material:

Visa. Effective with the **April 2022 VisaNet Business Enhancements release**, Visa began assigning eight-digit issuer BINs and required all clients to process using the eight-digit BIN structure. Visa will not assign new six-digit issuing BINs after that release, but both six-digit and eight-digit BINs exist in the estate. Notably, **acquirer numerics remain six digits** and were renamed "Acquirer Identifiers" — the expansion applied to the issuing side only. Visa also confirmed that PANs and tokens **remain sixteen digits**, with V PAY in Europe called out as an exception using nineteen-digit PANs in some implementations.

Mastercard. Adoption of the eight-digit standard was announced in 2017, with **April 2022** as the mandatory date for acquirers and service providers to support eight-digit BINs and **eleven-digit account ranges**.

American Express. Fifteen digits, prefixes 34 and 37. Amex is a three-party scheme, and its numbering, allocation and product identification are internal to Amex in a way that four-party scheme numbering is not.

The account-range arithmetic is worth doing explicitly, because it explains the schemes' issuance policies. Take a sixteen-digit PAN with an eight-digit BIN. Subtract the BIN and the check digit and you have **seven digits of individual account identifier: ten million accounts per BIN.** Visa's account range

is nine digits — the BIN plus one more — so a single eight-digit BIN divides into at most **ten** account ranges, which is exactly what Visa's own documentation states. Mastercard's eleven-digit range is the BIN plus three, giving up to a thousand ranges of ten thousand accounts each.

This is also why Visa's BIN utilisation policy, effective from its October 2017 release, instructs issuers not to randomise card issuance across an entire BIN but to **randomise starting at the tenth digit** — the tenth digit is the first digit that is not fixed by the nine-digit account range. The same bulletin instructs acquirers and processors to process transactions "based on complete nine-digit account range records in ARDEF tables rather than relying solely on the first six digits". ARDEF is Visa's **Account Range Definition** file, and it is a restricted product distributed to Visa clients through their account managers, not a public dataset.

Scheme prefixes and lengths

The following table is the conventional industry summary. Treat it as indicative. **The authoritative source for any given range is the scheme's own account range file**, not a published table, because ranges are added, moved and retired continuously.

Scheme	Leading digits	PAN length	Luhn
Visa	4	13, 16, 19	Yes
Mastercard	51–55	16	Yes
Mastercard 2-series	222100–272099	16	Yes
American Express	34, 37	15	Yes
Discover	6011, 644–649, 65	16–19	Yes
Diners Club International	30, 36, 38, 39	14–19	Yes
JCB	3528–3589	16–19	Yes
UnionPay	62	16–19	Yes
Maestro	5018, 5020, 5038, 5893, 6304, 6759, 6761–6763	12–19	Yes
Mir	2200–2204	16–19	Yes
RuPay	60, 65, 81, 82, 508	16	Yes

Two dated observations. Maestro is the widest-ranging length in the table and also the one being withdrawn: Mastercard began phasing Maestro out across Europe from **1 July 2023**, with issuers required to replace expired or lost Maestro cards with a different product. And note the overlap: 65 appears against Discover, RuPay and other schemes, which is a reminder that a two-digit prefix identifies nothing reliably.

The check digit, formally

The check digit is computed by the **Luhn formula**, described in US patent **2,950,048, "Computer for verifying numbers"**, filed by Hans Peter Luhn on 6 January 1954, granted 23 August 1960 and assigned to IBM. The patent has long since expired and the algorithm is specified within ISO/IEC 7812-1 for PAN check-digit computation.

The exact procedure, stated so that an implementation can be written from it without ambiguity:

1. Starting from the **rightmost** digit of the PAN, which is the check digit, and moving left, double the value of every second digit — that is, digits in positions 2, 4, 6 and so on counted from the right.
2. If a doubled value exceeds 9, subtract 9 from it. Equivalently, sum its two decimal digits; the results are identical for all inputs 0 to 9.

3. Sum all resulting values together with the undoubled digits.
4. The PAN is well formed if and only if the sum is congruent to 0 modulo 10.

To **generate** a check digit for a payload of n-1 digits, append a placeholder `0`, run steps 1 to 3, and set the check digit to `(10 - (sum mod 10)) mod 10`. The outer `mod 10` is not decorative: when the running sum already ends in zero the check digit is `0`, and an implementation that computes `10 - 0 = 10` and stores the low digit by accident will pass its own tests and fail in production on roughly one PAN in ten.

Anchoring at the right-hand end is the specification's wording and it is the only safe way to implement it, because it makes the algorithm indifferent to PAN length. Implementations that anchor from the left must branch on odd versus even length, and this is the single most common defect in hand-rolled validators. Worked on a fifteen-digit American Express test number, **3782 8224 6310 005**, the doubling positions counted from the left are 2, 4, 6, 8, 10, 12 and 14:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Digit	3	7	8	2	8	2	2	4	6	3	1	0	0	0	5
Becomes	3	5	8	4	8	4	2	8	6	6	1	0	0	0	5

The sum is 60, congruent to 0 modulo 10, and the number is well formed.

The blind spots demonstrated in the previous section are derivable rather than anecdotal. Define $f(d)$ as the doubled-and-reduced value of a digit minus the digit itself. Then $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 3$, $f(4) = 4$, $f(5) = -4$, $f(6) = -3$, $f(7) = -2$, $f(8) = -1$, $f(9) = 0$. An adjacent transposition changes the checksum by $f(a) - f(b)$, which is zero only when $f(a) = f(b)$, and the only distinct pair for which that holds is **0 and 9**. Every other adjacent transposition is detected.

For twin substitutions, define $g(d)$ as the doubled-and-reduced value plus the digit. Modulo 10 the values are 0, 3, 6, 9, 2, 6, 9, 2, 5, 8 for digits 0 to 9. The collisions are $g(2) = g(5)$, $g(3) = g(6)$ and $g(4) = g(7)$, giving the undetected substitutions **22<->55, 33<->66 and 44<->77**.

Finally, a scoping note. Within ISO/IEC 7812 the check digit is mandatory and a PAN that fails Luhn is malformed. Numbers that merely look like PANs but are issued outside ISO/IEC 7812 — some private-label and closed-loop store cards, for instance — are under no obligation to use it, and refusing them on Luhn grounds is a category error.

Expiry: four representations of one date

The expiry date exists in at least four forms in a live transaction, and they do not all have the same precision.

Printed on the card. Month and year, `MM/YY`, meaning valid to and including the final day of that month.

Magnetic stripe, ISO/IEC 7813:2006. On **track 2**, the fields in order are: start sentinel, PAN of up to 19 digits, field separator, expiration date as **four digits `YYMM`** (or the separator again if absent), a three-digit service code, discretionary data, end sentinel and a longitudinal redundancy check character, with a maximum record length of **40 numeric characters**. On **track 1**, Structure B, the record is up to **79 alphanumeric characters** with the expiry likewise four digits.

Note the capacity arithmetic that follows, because it is the constraint people reach for when explaining PAN lengths. Forty characters on track 2 less the start sentinel, end sentinel and LRC leaves 37 char-

acters of data — which is exactly why ISO 8583 defines its track 2 field as a maximum of 37. Subtract the field separator, four digits of expiry and three digits of service code and you have **29 characters to share between the PAN and the discretionary data**. A nineteen-digit PAN therefore leaves ten digits of discretionary data, which is tight but workable given that the discretionary data conventionally carries a PIN verification value and a card verification value; a sixteen-digit PAN leaves thirteen.

There is a live inconsistency here worth naming rather than papering over. **ISO/IEC 7813:2006 still describes the PAN as comprising a six-digit IIN and a variable individual account number of up to twelve digits**, because it predates the 2017 revision of ISO/IEC 7812-1. Two current ISO standards describe the same field differently. Implementers should follow ISO/IEC 7812-1:2017 for the IIN length and treat the 7813 text as historic in that one respect.

In the chip, EMV. The relevant data elements from the EMV Integrated Circuit Card Specifications, Book 3, Annex A:

Tag	Name	Source	Format	Template	Length
5A	Application Primary Account Number (PAN)	ICC	cn	70 or 77	var. up to 10 bytes
5F34	Application PAN Sequence Number	ICC	n 2	70 or 77	1 byte
5F24	Application Expiration Date	ICC	n 6 YYMMDD	70 or 77	3 bytes
5F25	Application Effective Date	ICC	n 6 YYMMDD	70 or 77	3 bytes
57	Track 2 Equivalent Data	ICC	b	70 or 77	var. up to 19 bytes

Three things to read off that table. First, tag `5A` is compressed numeric: two digits packed per byte, so ten bytes carry up to twenty digit positions of which at most nineteen are PAN, padded on the right with hexadecimal `'F'`. Second, tag `5F24` carries **six digits, `YYMMDD`** — the chip holds a day, where the printed card holds only a month. What the issuer puts in the day field is a personalisation decision; a terminal performs a date comparison and must not assume a particular value. Third, tag `57`, Track 2 Equivalent Data, reproduces the track 2 structure in binary with hexadecimal `'D'` as the field separator in place of the = character and `'F'` padding — which is why the chip and the stripe can produce byte-different but semantically identical data, and why comparing them naively as strings fails.

Tag `5F34`, the PAN Sequence Number, is the field that resolves the "one number, several cards" problem raised earlier: it identifies and differentiates cards carrying the same PAN.

In the authorisation message, ISO 8583. The relevant data elements:

DE	Name	Format
2	Primary account number	n ..19
14	Date, expiration	n 4, YYMM
35	Track 2 data	z ..37
45	Track 1 data	an ..76

DE2's `n ..19` is a variable-length numeric field of up to nineteen digits, carried with a two-digit length prefix. DE14 is fixed at four digits, `YYMM` — the message format drops the day that the chip carried.

What the terminal does with the date

Expiry checking happens in EMV **processing restrictions**, where the terminal compares the Application Expiration Date, tag `5F24`, against the Transaction Date, tag `9A`. The outcome is recorded in the **Terminal Verification Results**, tag `95`, a five-byte field. In byte 2:

Bit	Meaning
b8	Card and terminal have different application versions
b7	**Expired application**
b6	**Application not yet effective**
b5	Requested service not allowed for card product
b4	New card
b3–b1	Reserved for future use

Setting a TVR bit is not the same as declining. The TVR records observations; the decision of whether an observation causes a decline, an online authorisation request or nothing at all is made later, by comparing the TVR against the terminal action codes and issuer action codes. That comparison is the subject of a later chapter. The point to carry forward is that an expired card does not fail at the chip — it fails, if it fails, because somebody's action codes said it should.

Why PAN length varies, definitively

Five separate forces, and it is worth separating them because they are usually conflated.

The standard sets a ceiling and, since 2017, a floor. Nineteen digits maximum; ten digits minimum for issuers holding eight-digit IINs. Everything between is permitted.

History precedes the standard. American Express and Diners Club were issuing numbered cards before ISO/IEC 7812 consolidated the practice. The standard was written to accommodate the installed base, not to replace it, which is why fifteen and fourteen digit numbers remain conformant.

The magnetic stripe imposed a practical squeeze but never forced sixteen. As shown above, track 2 leaves 29 characters for PAN and discretionary data combined. Long PANs are possible and merely leave less room for the values that the stripe also had to carry.

Acceptance infrastructure calcified around sixteen. Merchant databases, printed forms, screen layouts and card-embossing conventions all standardised on four groups of four. This inertia is real enough that Visa, announcing the eight-digit BIN expansion, went out of its way to reassure the market that PANs and tokens **remain sixteen digits** and that card embossing alignment was unaffected. The nineteen-digit capability exists; it is used sparingly, and V PAY in Europe is the exception Visa itself names.

BIN capacity is the modern pressure. The whole reason for the 2017 expansion is that six-digit IINs ran out. Moving to eight digits multiplied the available issuer identifiers by a hundred, at the cost of two digits of account space per PAN — from nine digits of account identifier down to seven on a sixteen-digit PAN. That is the trade the industry made, and it is why the schemes now police BIN utilisation and instruct issuers to fill existing ranges before requesting new ones.

What a BIN lookup can and cannot tell you

A BIN lookup is a table join. Its answer is only ever as good as the table, and the tables differ enormously in provenance.

The authoritative sources are the schemes' own range files, distributed to their licensed participants: Visa's Account Range Definition (ARDEF) file, restricted to Visa clients, and Mastercard's equivalent account range data. Separately, in EMV 3-D Secure, the **Directory Server** supplies card range data to a 3DS Server in the **PRes** message, the response to a PReq, containing `startRange`, `endRange`, `acsStartProtocolVersion`, `acsEndProtocolVersion`, an optional `threeDSMethodURL` and an `actionInd` for each range. That is an authoritative, scheme-supplied range list, and it exists for one narrow purpose: to let the 3DS Server work out which protocol versions the issuer's Access Control Server supports before it attempts an authentication.

Everything else on the market — the commercial BIN databases sold by the gigabyte — is derived, aggregated and of variable freshness.

What a good lookup can tell you, subject to the table being current: the scheme, the issuing institution, the issuer's country, the product identifier and hence the broad product type (credit, debit, prepaid), and whether the card is a consumer or commercial product. Those attributes are genuinely useful, and they are used for real decisions: routing, fee categorisation, eligibility for dynamic currency conversion, fraud rules keyed on issuer country, and 3DS protocol version discovery.

What it cannot tell you, however good the table:

Whether the card exists. A range covers ten million potential accounts and a lookup does not know which are issued.

Whether the card is active, blocked, expired, or has any funds behind it. Those questions have exactly one answer, and it comes from an authorisation.

Anything about the cardholder. Not name, not address, not age, not creditworthiness.

Anything reliable at a granularity coarser than the account range. This is the correction that costs money: a six-digit lookup against a scheme that routes on nine or eleven digits can straddle several products with different funding sources and different interchange. If your table is keyed on six digits and the scheme's is keyed on nine, your answers are not merely stale, they are structurally incapable of being right.

Whether the number is a card number at all. EMVCo's Payment Tokenisation Specification – Technical Framework, which EMVCo publishes for public access, defines a payment token as a variable-length, ISO/IEC 7812-compliant numeric issued from a designated token BIN or token BIN range and flagged as such in the appropriate BIN tables; it passes the same basic validation rules as an account number, including the Luhn check digit, and it never takes the same value as a real PAN. A network token therefore has the length and shape of a PAN and satisfies Luhn, but its leading digits belong to a range set aside for tokens rather than to the issuer's card product, so a conventional lookup describes the token range and not the card.

Which scheme a transaction will actually route on. Co-badged cards, particularly domestic debit schemes in continental Europe, carry more than one acceptance brand on a single PAN, and the routing choice is made by rules outside the number.

Handling the number in a system you are responsible for

Four rules, each with a citable basis.

Store it as a string, never as an integer. IINs under MII `0` may begin with a zero, and integer storage silently destroys leading zeros. Separately, the largest signed 64-bit integer is 9,223,372,036,854,775,807 — itself nineteen digits — so the upper part of the nineteen-digit PAN space does not fit. And no arithmetic operation on a PAN is meaningful.

Accept the full permitted range of lengths. Twelve to nineteen digits will cover the schemes in circulation; ISO/IEC 7812-1 permits up to nineteen. Do not validate against sixteen.

Mask on display, truncate in storage, and know that these are different controls. PCI DSS separates them: **Requirement 3.4.1** governs masking when the PAN is displayed or printed, and **Requirement 3.5.1** governs rendering the PAN unreadable wherever it is stored. The PCI SSC's own guidance puts the distinction plainly: masked data can be unmasked, but there is no un-truncation without recovering the PAN from another source.

Know the truncation limits. PCI SSC FAQ 1091, as updated in June 2022, gives the maximum digits that may be retained:

PAN length	Brands	Minimum removed	Maximum retained
16 digits (6- or 8-digit BIN)	Discover, JCB, Mastercard, UnionPay, Visa	At least 4 digits	First 8, any other 4
15 digits	American Express	At least 5 digits	First 6, last 4
Fewer than 15 digits	Discover	—	First 6, any other 4

Three cautions on that table. It states **maxima permitted where there is a documented business need**, not defaults; first six and last four remains the format accepted by every brand and remains the sensible starting point. Mastercard's own position paper states that its allowable truncation format is not mandatory and that entities are not required to change their existing truncation as a result of the eight-digit BIN migration, and recommends retaining the fewest digits possible. And the PCI SSC warns specifically that holding the **same** PAN truncated **different ways** in different data stores can expose more digits in aggregate than either format permits alone — a failure mode that only appears when two systems are correlated, which is to say during a breach.

The number in one paragraph

A PAN is an eight-digit issuer identification number allocated by a registry, followed by up to ten digits of issuer-assigned account identifier, followed by one Luhn check digit, to a maximum of nineteen digits in total. The schemes route on finer account ranges than the issuer identifier — nine digits at Visa, eleven at Mastercard — because that is where product, funding source and fee category are determined. The check digit catches accidents and nothing else. The expiry date exists as four digits on the stripe and in ISO 8583, six on the chip, and means valid through the final day of the stated month. And a lookup on the leading digits tells you about a *range*, never about a card, and never about the person holding it.

12.98 Common wrong ideas

Wrong: A card number is a serial number, and somewhere there is a very large list. **Right:** It is a structured hierarchical address: which institution is answerable, which account at that institution, and one final digit that is not information at all.

Wrong: The issuer identifier is six digits. **Right:** It has been eight since January 2017, Visa and Mastercard made support mandatory from their April 2022 releases, and the old six-digit identifiers were subdivided into blocks of a hundred rather than withdrawn.

Wrong: The issuer identifier is what the network routes and prices on. **Right:** The schemes route on a finer account range — nine digits at Visa, eleven at Mastercard — because that is where product, funding source and fee category are determined.

Wrong: Passing the check-digit test means the card exists. **Right:** It means the number is spelled correctly; it is a spell-checker rather than a lock, and a well-formed number may correspond to no account anywhere on earth.

Wrong: The check digit gives some protection against an attacker. **Right:** The rule has been public for over sixty years, generating millions of valid numbers is a first-week programming exercise, and card-testing attacks do exactly that before hunting for the live ones.

Wrong: The check digit catches every adjacent transposition. **Right:** Swapping an adjacent 0 and 9 is invisible to it always and everywhere, and the twin substitutions 22 against 55, 33 against 66 and 44 against 77 also pass.

Wrong: The doubling starts at the first digit on the left. **Right:** The specification anchors at the rightmost digit, which makes it indifferent to length; anchoring from the left requires branching on odd against even and is the single most common defect in hand-rolled validators.

Wrong: The first digit tells you what kind of card you are holding. **Right:** The Major Industry Identifier is archaeology, and Mastercard's 222100–272099 range puts mainstream consumer cards under a digit the table labels "airlines".

Wrong: Every number flowing through your systems is a card number. **Right:** Some are network tokens issued from token ranges, of the same length and satisfying the same check, and a lookup on their leading digits describes the token range rather than any issuer's card product.

Wrong: A card number identifies one card and one bank account. **Right:** Replacements and additional cardholders can share a number distinguished only by the chip's sequence number, and the mapping to accounts is the issuer's private business and is not one-to-one in either direction.

12.99 Chapter summary in 20 lines

1. A card number is not a serial number but an address, read from the left in the way a postal address narrows from country to house.
2. ISO/IEC 7812-1 defines the primary account number as an issuer identification number, an individual account identifier and one check digit, to a maximum of nineteen digits.
3. Since the fifth edition of January 2017 that issuer identifier is eight digits rather than six, and each old six-digit identifier became a block of one hundred eight-digit ones.
4. That is why six-digit and eight-digit identifiers coexist, and why any system built before about 2019 almost certainly assumes the wrong figure.

5. Even eight digits is not how the network routes: Visa uses a nine-digit account range and Mastercard an eleven-digit one, because that is where product and fee category live.
6. Visa has told its acquirers explicitly to process on the complete nine-digit account range record rather than on the first six digits.
7. The first digit is the Major Industry Identifier, and it is archaeology, as Mastercard's 2-series consumer cards under an "airlines" digit demonstrate.
8. The last digit is a Luhn check digit: double every second digit from the right, subtract nine where a doubled value exceeds nine, and require the total to be congruent to zero modulo ten.
9. Worked on 4012 8888 8888 1881 the total is 90, and both a transposition of the last two digits and a single-digit change push it off a multiple of ten.
10. To generate rather than test, append a placeholder zero and set the check digit to ten minus the sum modulo ten, all taken modulo ten again, because the outer step is not decorative.
11. Anchoring at the rightmost digit makes the algorithm indifferent to length; anchoring from the left forces a branch on parity and makes every fifteen-digit American Express number appear invalid.
12. The check digit catches accidents superbly and adversaries not at all, and it has known blind spots rather than being a guarantee.
13. Those blind spots are derivable rather than anecdotal, from the doubled-and-reduced value of each digit taken against the digit itself.
14. The expiry date exists in four forms: `MM/YY` printed, `YYMM` on the stripe and in the authorisation message, and `YYMMDD` in the chip, which carries a day the other three drop.
15. A card marked 09/28 is good through the last second of 30 September 2028 and stops working on 1 October.
16. An expired card does not fail at the chip: the terminal records the observation in the terminal verification results, and whether that becomes a decline is decided later by the action codes.
17. Card number length varies for five separate reasons: the standard's ceiling and floor, history that predates the standard, the stripe's capacity squeeze, acceptance infrastructure that calcified around sixteen, and issuer identifier capacity pressure.
18. Track 2's forty characters leave twenty-nine to share between the number and the discretionary data, which is a squeeze rather than a rule forcing sixteen digits.
19. A BIN lookup is a table join whose authoritative sources are the schemes' own range files, and it can give scheme, issuer, country and product type but never whether a card exists, whether it has funds, or anything about the person holding it.
20. Store the number as a string, accept the full permitted range of lengths, mask on display and truncate in storage knowing these are different controls, and remember that the same number truncated two ways in two stores can expose more digits in aggregate than either format permits alone.

Chapter sources: ISO/IEC 7812-1:2017 (5th edition, confirmed 2022) and ISO/IEC 7812-2:2017, with the ISO announcement of the eight-digit IIN change and the ISO/IEC 7812 Registration Authority information sheet published via ANSI; ISO/IEC 7813:2006 (track 1 and track 2 structure); EMV Integrated Circuit Card Specifications for Payment Systems, Book 3, Annex A (tags 5A, 57, 5F24, 5F25, 5F34, 95, 9A); EMV 3-D Secure Protocol and Core Functions Specification (PReq/PRes card range data); EMV Payment Tokenisation Specification – Technical Framework (definitions of Payment Token, Token BIN and Token BIN Range); Visa bulletin on the implementation of eight-digit BINs, Visa article AI09153 on BIN management policies, Visa's eight-digit BIN numerics FAQ and Visa's eight-digit BIN PCI position paper; Mastercard's 2-Series BIN issuer impact checklist and its "8-Digit BIN Expansion and PCI Standards" paper; PCI DSS v4.0.1 requirements 3.4.1 and 3.5.1, PCI SSC FAQ 1091 (updated June 2022) and the PCI SSC masking-versus-truncation FAQ; US patent 2,950,048, Hans P. Luhn, IBM, filed 6 January 1954, granted 23 August 1960. Check-digit arithmetic, track capacity arithmetic and account-range capacity arithmetic in this chapter were computed and verified by

hand rather than quoted.

3

CHAPTER

The Four Things Called the Security Code

13.0 What this chapter gives you

1. You will be able to name the four values that a card carries under the heading "security code" — CVV1, CVV2, iCVV and dCVV — and say which of the four reading interfaces each one belongs to.
2. You will be able to explain why a criminal who copies every byte out of a chip onto a blank magnetic stripe gets a card that declines, and name the single issuer check that makes it decline.
3. You will be able to say why that same attack nevertheless succeeded against a supermarket group and a Georgia liquor retailer in 2020, and what the issuers concerned had failed to do.
4. You will be able to derive all three static values from the same algorithm, and explain why feeding in a service code of `000` rather than `999` changes every digit of the answer with no pattern connecting them.
5. You will be able to tell a caller that no bank anywhere can look up the printed code and read it back, because nothing is stored and every value is recomputed inside a hardware security module at authorisation time.
6. You will be able to correct the folklore that iCVV is what makes a chip transaction impossible to counterfeit, and name the Application Cryptogram as the actual defence.
7. You will be able to read a POS entry mode field and predict which value the issuer will compute, and so diagnose the decline caused by a terminal that reads a chip and reports a swipe.
8. You will be able to trace the printed code through the authorisation message on both Visa and Mastercard, and interpret the result codes `M`, `N`, `P`, `S` and `U` without treating them as the authorisation decision.
9. You will be able to list the places where PCI DSS requirement 3.3.1.2 is most often broken — debug logs, crash dumps, access logs, queues, call recordings, paper and backups — and explain why encrypting a retained code is no defence at all.
10. You will be able to explain why a monthly subscription can never re-present the printed code, and why the merchant must retain the `M` result instead of the value.

Ask a room of payments people what "the CVV" is and you will get one answer: the three digits on the back of the card. Ask the same room what the issuer's authorisation host actually checks, and the answers begin to diverge, because there is not one such value on a modern card. There are at least four. They are computed by the same family of algorithm, by the same machine, from mostly the same inputs, and they come out looking identical — three unremarkable decimal digits. They are not interchangeable. Presenting the wrong one at the wrong interface is precisely the event that a large part of card security is designed to detect.

The four are conventionally named CVV1, CVV2, iCVV and dCVV. Three of them are burned into the card once, at personalisation, and never change. One of them is manufactured fresh for every tap. One of them is printed in ink where a human can read it, and is the only one of the four that a merchant is forbidden to keep. The industry uses different names for all of them depending on which scheme's rulebook is open on the desk, which is why the same value can appear in three different documents as CVV, CVC and CID without anybody being wrong.

This chapter separates them. It explains what each one is a defence against, why removing any one of them would open a specific and historically exploited hole, and — the question that this chapter exists to answer properly — why copying the data off a chip and writing it onto a blank magnetic stripe does not give you a working card, except when an issuer has failed to do the one check that makes it not work.

The plain version

Imagine a building with four different doors: a front door, a side door, a delivery entrance and a gate at the back that opens onto the car park. All four doors belong to the same building. The same security office issues the passes for all four. But each door has its own pass, and the passes are deliberately different from one another, so that a pass for the delivery entrance is refused at the front door even though it was issued by the same office to the same person on the same morning.

Your bank card is that building. The four doors are the four ways a shop can read it.

The **first door is the brown stripe** on the back. A machine drags the card through a slot, reads the magnetism, and gets a line of digits.

The **second door is the printed code**, the three digits on the signature panel. Nobody's machine reads that. A human reads it, off the card in their hand, and types it into a website or reads it down a telephone.

The **third door is the chip**, the little gold rectangle. A terminal pushes the card into a slot, the chip wakes up, and the terminal asks it questions.

The **fourth door is the tap**. The card has an aerial hidden inside the plastic, and holding it near a reader lets the two of them talk by radio for about a fifth of a second.

Four doors, four passes. That is all "the four things called the security code" means.

The pass machine

The bank has a machine — really a locked box in a data centre, but think of it as a machine — that makes these passes. You feed it three things: your card number, your expiry date, and a three-digit **door number**. It thinks for a moment and prints out three digits.

The clever part is that the door number goes *into* the calculation. Change the door number, and every digit of the answer changes completely. There is no pattern connecting them. Knowing the answer for one door tells you nothing at all about the answer for another.

So the bank runs the machine three times for each card it makes. The numbers below are invented for the example — only the bank's machine knows the real ones — but the shape of what happens is exactly right.

What the bank feeds in	Door number	Pass that comes out	Where it is put
Card number and expiry	101 (the stripe's door)	561	Written into the brown stripe
Card number and expiry	000 (the printed door)	348	Printed in ink on the back
Card number and expiry	999 (the chip's door)	907	Stored inside the chip

Three passes. One card. Nothing about 561 helps you guess 348 or 907.

Why this stops a thief

Here is the attack the third row exists to defeat, and it is worth walking through slowly because it is the point of the whole chapter.

A criminal gets hold of a shop's card terminal, or infects the shop's till computer with software that watches everything the terminal sends. A customer inserts a chip card. The criminal's software captures everything the chip said: the card number, the expiry date, and the chip's pass — **907**.

Now the criminal buys a blank plastic card and a stripe-writing machine, both of which are cheap and legal, and writes the stolen data onto the stripe. The card number is right. The expiry is right. The pass written into the stripe is 907, because 907 is what was stolen.

The criminal walks into a shop and swipes it.

The shop's till sends the card number, the expiry date and the pass it just read — 907 — to the bank. The bank looks at the message and sees that this came in through the stripe door. So it runs its machine with the stripe's door number, 101, and gets **561**.

907 is not 561. **Declined.**

The criminal has a perfect copy of the card number and it is worthless at the stripe door, because the pass he stole was cut for a different door. That is the entire idea. It is called iCVV — the "i" is for the chip inside — and it is the reason the industry did not simply copy the stripe's pass into the chip when chips were introduced. Had it done so, every chip terminal in the world would have been a machine for mass-producing stripe clones.

The fourth pass is different

The tap door does not use a stored pass at all. It makes one up each time.

Inside the chip there is a counter. It starts at 1 when the card is made and goes up by one every single time the card is used. Nobody can wind it back. When you tap, the chip takes the current counter value, mixes it with a random number the reader has just invented, does its sum, and produces a fresh three-digit code that has never existed before and will never be valid again.

Somebody actually recorded this happening. A researcher held the same contactless card against a reader four times in a row and wrote down what came out each time. The card number was identical every time. The expiry date was identical every time. The last stretch of digits was different every time:

```
...0000072029191
...0000018129281
...0000054629311
...0000099829771
```

Same card, four seconds apart, four different endings. Copy one of those and try to use it again and the bank sees a counter value it has already seen, and refuses. This is called **dCVV**, for dynamic — a pass that is only good for one entry.

So why four?

Because each door is attacked differently.

The stripe is trivially copyable, so its pass exists only to prove that the copier saw a real stripe rather than merely learning a card number from a receipt.

The printed code is not in the card's data at all, anywhere, in any machine-readable form. That is its whole purpose. If a thief steals a database of card numbers from a shop's computer, or reads them off a stripe with a skimmer glued inside a cash machine, he does not get the printed code, because the printed code was never in the data. He would have had to physically hold the card and turn it over.

The chip's pass exists to keep the chip from becoming a stripe-cloning service, as above.

And the tap's pass exists because radio is broadcast: anyone standing nearby with the right aerial might overhear, so nothing overheard may be worth replaying.

One thing, many names

Visa calls it CVV, for Card Verification Value. Mastercard calls it CVC, for Card Validation Code. American Express calls it CID, and prints four digits on the front of the card rather than three on the back. Discover also says CID. JCB says CAV. China UnionPay says CVN. They are the same idea. When a specification says "CVC 2" and another says "CVV2", they mean the printed code, and a system that handles both must simply know that it is being told the same thing twice in two dialects.

The rule to remember

Of the four, exactly one must never be written down by anybody except the bank: the printed one. A shop is allowed to keep your card number for refunds. A shop is never allowed to keep the printed code — not in a database, not in a spreadsheet, not in a note on an order, not scribbled on a pad by the till. The moment a shop's payment has gone through, that code must be gone. Its only value is that it is scarce, and a code that thousands of shops are storing is not scarce.

Where the plain version stops being true

The building-with-four-doors picture gets you to the right conclusion, and it is wrong in four ways that matter to anyone implementing this.

There is no stored pass anywhere, and no lookup

The plain version implies the bank keeps a list: card 4123..., stripe pass 561, printed pass 348. It does not. Nothing is stored. Each value is *recomputed from scratch* at authorisation time, inside a hardware security module, from the card number, the expiry date, the service code and a pair of secret keys, and the freshly computed answer is compared with whatever arrived in the message. The keys are the only thing kept.

This has a consequence that surprises people, including bank staff: **an issuer cannot look up your CVV2 and tell you what it is.** It can only regenerate it. Some issuers' card-management systems expose that as a function, most deliberately do not, and none of them are reading it out of a column in a table. Similarly, when a call centre says "we can see the last four digits", they mean the card number; there is no screen anywhere with your printed code on it.

Four is a simplification in one direction and an undercount in the other

"Four" is a teaching number. In practice the same personalisation run may produce a CVV1 for the stripe, a distinct value for track 1 and track 2 discretionary data if the issuer chooses to differ them, an iCVV for the contact chip, a separate contactless value, a CVV2 for print, and — for issuers who offer it — a rotating code shown in the banking app that changes every few minutes. Card-not-present verification values used inside 3-D Secure are a separate family again, and are not card verification values at all.

Pulling the other way, the four are not equals. CVV1, CVV2 and iCVV are static three-digit numbers. dCVV and its Mastercard equivalent are dynamic. And none of them is what actually secures a chip transaction.

The security code is not why chip transactions cannot be cloned

This is the correction most worth internalising, because the popular explanation is backwards.

A chip transaction is not made hard to counterfeit by iCVV. It is made hard to counterfeit by the **application cryptogram** — a message authentication code the chip computes over the transaction amount, currency, date, terminal country, a terminal-supplied random number and its own transaction counter, using a key derived from a master key that the chip will not disclose and cannot be read out of it. Chapter 5 of this volume is entirely about that cryptogram. A criminal holding every byte of static data from a chip still cannot produce a valid cryptogram for tomorrow's £47.20 purchase in Leeds, because he does not have the key.

iCVV is not the lock on the chip door. iCVV is the thing that stops data taken from the chip door being reused at the stripe door. It is a *cross-channel* defence, not a chip defence, and it is doing a much narrower job than the folklore suggests.

The codes are only worth anything if somebody checks them, and sometimes nobody does

All of this assumes the issuer validates. In 2020 researchers at Cyber R&D Labs tested chip card implementations from ten banks across Europe and the United States, harvested data from four of them and produced cloned magnetic stripe cards that were used to place real transactions. In the same period, Gemini Advisory documented what it called EMV-Bypass Cloning in two breaches — a supermarket group where more than 720,000 cards were compromised from 16 January 2020, and a Georgia liquor retailer on 29 June 2020 — in which iCVV harvested from chip transactions was written onto counterfeit stripes and worked, because the issuers concerned were not checking that the value arriving on a stripe transaction was the stripe's value.

Visa's own commentary on those events makes the conditionality explicit: the data available from a chip transaction is the account number, iCVV and expiry date, and "provided iCVV is validated properly, the risk of counterfeit fraud was minimal". Provided. The mechanism is sound. Its deployment is uneven, and it is smaller issuers and processors, not the large ones, where the check has been found missing.

There is a fifth, subtler failure in the same family. The service code is an input to the calculation, but it is also a field that travels in the message. Visa has warned acquirers and issuers that service codes of 000 or 999 should never be encoded on a magnetic stripe, and that it "is aware of scenarios in which either 000 or 999 has been encoded on the magnetic stripe of counterfeit cards, resulting in issuer fraud losses". The attack is exactly what it sounds like: if an issuer's validation takes the service code from the incoming track data rather than from its own record of the card, a fraudster can encode 999 on the

stripe alongside a stolen iCVV, and the issuer will helpfully compute the chip's value and find that it matches.

The technical version

Where each value physically lives

The magnetic stripe layout is defined by ISO/IEC 7813, which specifies the data structure and content of tracks 1 and 2. Track 1, Structure B, carries up to 79 alphanumeric characters; track 2 carries up to 40 numeric characters at a lower recording density but a more compact character encoding. Track 3, defined separately in ISO/IEC 4909, holds 107 numeric characters and is unused by the card schemes in practice.

The field order on track 2 is: start sentinel `;`, primary account number of up to 19 digits, field separator `=`, expiry date as YYMM, three-digit service code, discretionary data, end sentinel `?`, and a longitudinal redundancy check character. Track 1 is the same information plus a format code `B`, the cardholder name between `^` separators, and its own discretionary data field.

Discretionary data is where CVV1 lives. ISO/IEC 7813 does not say what goes in it; it is left to the issuer. What the schemes put there is the three-digit card verification value, usually followed by a PIN verification key indicator and PIN verification value, and padding.

For the chip, EMV Book 3 defines tag `57`, Track 2 Equivalent Data, described as containing "the data elements of track 2 according to ISO/IEC 7813, excluding start sentinel, end sentinel, and Longitudinal Redundancy Check", packed two digits to a byte with hexadecimal `D` standing in for the `=` separator and a trailing `F` nibble as padding where the length is odd. Tag `9F1F` is Track 1 Discretionary Data. Tag `5F30` is the service code, formatted `n 3` in two bytes.

The composite picture:

Value	Lives in	Read by	Static or dynamic
CVV1 / CVC1	Discretionary data of the encoded magnetic stripe tracks	A stripe reader	Static
CVV2 / CVC 2 / CID / CAV2 / CVN2	Ink on the card body only	A human being	Static
iCVV / Chip CVC	Discretionary data inside EMV tag `57` and, where present, `9F1F`	A chip terminal, contact or contactless in EMV mode	Static
dCVV / CVC3	Substituted into discretionary data at read time	A contactless reader in magnetic-stripe mode	Dynamic, per tap

The generation algorithm

The card verification value algorithm originates with Visa and is implemented identically in every payment HSM. It is not published by the schemes, but it is documented in the command sets of the HSM vendors and in IBM's cryptographic coprocessor documentation, which describes the Visa CVV as calculated by "the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys". Those two keys are the **card verification key pair**, conventionally CVK A and CVK B.

The steps are:

1. Concatenate the primary account number, the expiry date and the three-digit service code as decimal digits, left-justify the result in a 128-bit field and pad to the right with binary zeroes.
2. Split the 128-bit field into two 64-bit blocks, Block 1 and Block 2.
3. DES-encrypt Block 1 under CVK A.
4. Exclusive-OR the result with Block 2.
5. Apply the EDE sequence to that: encrypt under CVK A, decrypt under CVK B, encrypt under CVK A. This is Triple DES with a two-key bundle.
6. Decimalise the 16 hexadecimal digits of the result in two passes. The first pass walks left to right and extracts every digit in the range 0 to 9. The second pass walks left to right again and extracts every digit in the range A to F, subtracting 10 from each so that A becomes 0 and F becomes 5. Concatenate the two extracted strings.
7. Take the leftmost three digits. For American Express, which prints four, take four.

Everything about which value you get is decided by step 1. The keys are the same, the arithmetic is the same, the only thing that varies is what you feed in:

Value	PAN	Expiry	Service code input
CVV1	Yes	Yes	The card's actual service code, for example `101` or `201`
CVV2	Yes	Yes	`000`
iCVV	Yes	Yes	`999`

Note carefully what this does and does not say. The `999` used to compute iCVV is a value *fed into the calculation*. It is not a claim about what is encoded in the service code field of tag `57`. Whether an issuer also encodes a distinguishing service code in the chip's Track 2 Equivalent Data is an issuer decision and implementations differ; Visa's warning about `000` and `999` on counterfeit stripes only makes sense in a world where the encoded field and the calculation input are separable things. Nothing in the algorithm requires the same key pair to be used for all three values either, and an issuer is free to provision a separate card verification key for iCVV; Amazon's payment cryptography documentation, for one, creates the iCVV key as a card verification key in its own right rather than reusing an existing one.

The output space is three decimal digits, so a thousand possibilities. That is deliberately weak against brute force and deliberately strong against the thing it is for. Guessing a CVV2 for a known card number takes on average 500 attempts, which is why the defence against CVV2 guessing is not the code's length but the issuer's velocity controls and the scheme's rules on repeated declines.

CVV1 and the magnetic stripe path

When a card is swiped, the terminal sends what it read. In ISO 8583 that is data element 35 for track 2 and data element 45 for track 1, and the schemes overwhelmingly prefer track 2: Visa's guidance to acquirers is blunt that "Track 1 should not be supported in the authorization message."

The issuer's decision about which value to compute is driven by data element 22, POS entry mode. The values that matter here, in Visa's numbering, are `01` for manual key entry, `02` or `90` for a magnetic stripe read, `05` or `95` for a contact chip read, `07` for contactless using chip data rules and `91` for contactless using magnetic-stripe data rules. A fallback swipe — a chip card at a chip-capable terminal where the chip could not be read — carries its own distinct entry mode so that the issuer can see it happened, which is the basis of every fallback-monitoring rule the schemes operate.

Entry mode is therefore load-bearing in a way that is easy to underestimate. Visa's own instruction to devices is that "if a transaction is processed as magnetic stripe, the track data used in the transaction is read from the magnetic stripe and correspondingly, if a transaction is processed as chip, the track data used should be read from the chip", and that the entry mode field "must be accurate to avoid unnecessary declines". A terminal that reads the chip and reports entry mode 90 will hand the issuer chip-derived track data labelled as a swipe, the issuer will compute CVV1, the iCVV will not match, and the transaction will decline for reasons that will take the merchant's acquirer several days to work out.

The corresponding rule for chip data is that "the terminal must always transmit the full, unmodified contents of the Track 2 Equivalent Data in the chip to the acquirer including the Issuer Discretionary data". Terminals that trim, re-pad or normalise the discretionary data destroy the iCVV in the process. This is a real and recurring integration defect, particularly in software stacks that parse tag 57 into fields and then reassemble it.

CVV2 and the card-not-present path

CVV2 is printed, never encoded. On Visa, Mastercard and Discover it is three digits in or beside the signature panel on the reverse. On American Express it is four digits printed on the front, above and to the right of the embossed account number, and Amex calls it the Card Identification Number.

Its carriage in the authorisation message is scheme-specific and this is one of the places where a generic integration goes wrong. In VisaNet the value travels in Field 126, usage 10, CVV2 Authorization Request Data, and the result comes back in Field 44, usage 10. In Mastercard's Authorization Request/0100 the value goes in data element 48, Additional Data — Private Use, subelement 92, and the result returns in DE 48 subelement 87, Card Validation Code Result.

The result codes are common across both:

Code	Meaning
`M`	CVV2/CVC 2 matched
`N`	Did not match
`P`	Not processed
`S`	The merchant indicated the code was not on the card, but the issuer says it should be
`U`	Issuer is not certified, or has not supplied keys to the scheme
blank	Unknown or not applicable

Two operational points follow. First, the CVV2 result is *not* the authorisation decision. An issuer may return an approval with a result of `N`. Whether the merchant proceeds is the merchant's risk decision, and a great many merchants approve transactions their own issuer has told them did not match. Second, the result is durable evidence. Mastercard's chargeback rules for recurring transactions accept, as part of the merchant's defence, that the CVC 2 was provided in the initial Authorization Request/0100 and that DE 48 subelement 87 came back with a value of `M`. The result code may be retained. The code itself may not.

iCVV and the chip

At personalisation the issuer computes the value with service code input `999` and writes it into the discretionary data portion of tag `57`, and into `9F1F` where track 1 equivalent data is personalised. From that moment the chip's static track data and the stripe's track data carry the same PAN, the same expiry and different verification values.

At authorisation the issuer sees entry mode `05`, `95` or `07`, takes the discretionary data out of the chip track data in DE 35 or from the ICC data in DE 55, recomputes with `999`, and compares.

What this defeats, precisely: the conversion of chip-sourced static data into a working magnetic stripe. Gemini Advisory's name for the attack — EMV-Bypass Cloning — captures it exactly, since it "leverages information from one technology (EMV chips) and converts it into another less-secure technology (magstripe)". The countermeasure is a single comparison, and its failure mode is silent.

Why cloning a stripe does not give you a working chip transaction

The reverse direction is where the asymmetry is total, and it is worth being exact about why, because it is the reason the entire EMV migration was worth its cost.

A chip transaction requires the card to produce an **Application Cryptogram**, EMV tag `9F26`, in response to a GENERATE AC command. The cryptogram is a MAC computed over data the terminal supplies — amount, other amount, terminal country code, terminal verification results, transaction currency, transaction date, transaction type and the Unpredictable Number in tag `9F37` — together with data the card supplies, principally the Application Interchange Profile and the Application Transaction Counter in tag `9F36`. The key is an application cryptogram session key derived from a card-unique key, which is itself derived from an issuer master key using the PAN and the PAN Sequence Number in tag `5F34`. Tag `9F10`, Issuer Application Data, carries the issuer's own proprietary block, including in most schemes the counters and derivation information the issuer needs to reproduce the calculation.

None of that is present in stripe data. The stripe carries a PAN, an expiry, a service code and three static digits. It carries no key, no counter and no ability to compute anything. A terminal presented with a card that has no chip either declines, or performs a fallback swipe that the issuer can see and refuse.

And a criminal in possession of every static byte from a chip is no better off, because he does not have the card's key. He can replay a captured cryptogram, and the Application Transaction Counter will be one the issuer has already seen for that card. He can attempt to build a card that generates cryptograms, and he cannot, because the derivation key never leaves the issuer's HSM or the chip. For cards supporting dynamic data authentication the position is stronger still: the chip holds an RSA private key it will sign challenges with and will not disclose, so even an offline terminal with no network connection can distinguish a genuine card from a copy.

The security codes are three digits of static data. The cryptogram is the actual defence. iCVV exists to make sure that the three digits of static data cannot be smuggled sideways into a channel that has no cryptogram to check.

dCVV, CVC3 and contactless magnetic-stripe mode

For a period, contactless cards in some markets were deployed in a mode designed to reuse existing magnetic stripe message formats end to end, so that acquirers and issuers needed no message changes at all. This is MSD, magnetic-stripe data mode, and it is where the dynamic values live.

The trick is that the card hands the reader a track image with holes in it, plus instructions for filling the holes. In EMVCo's Kernel 2 specification — the Mastercard kernel, Book C-2 of the EMV Contactless Specifications for Payment Systems — the relevant data objects are:

Tag	Name	Source	Length
56	Track 1 Data	Card	variable
9F6B	Track 2 Data	Card	variable
9F60	CVC3 (Track1)	Card	2 bytes
9F61	CVC3 (Track2)	Card	2 bytes
9F62	PCVC3 (Track1)	Card	6 bytes
9F63	PUNATC (Track1)	Card	6 bytes
9F64	NATC (Track1)	Card	1 byte
9F65	PCVC3 (Track2)	Card	2 bytes
9F66	PUNATC (Track2)	Card	2 bytes
9F67	NATC (Track2)	Card	1 byte
9F6A	Unpredictable Number (Numeric)	Terminal	variable
9F6C	Mag-stripe Application Version Number (Card)	Card	2 bytes

The names decode as follows. **PCVC3** is a bit map that "indicates to the Kernel the positions in the discretionary data field of the Track 2 Data where the CVC3 (Track2) digits must be copied". **PUNATC** is a bit map that "indicates to the Kernel the positions in the discretionary data field of Track 2 Data where the Unpredictable Number (Numeric) digits and Application Transaction Counter digits have to be copied". **NATC** "represents the number of digits of the Application Transaction Counter to be included in the discretionary data field of Track 2 Data".

The sequence at the reader is: select the application, read the static Track 1 and Track 2 Data and the position bit maps, generate a numeric unpredictable number, issue a COMPUTE CRYPTOGRAPHIC CHECKSUM command carrying it, and receive back a two-byte CVC3 for each track. The reader then converts the CVC3 to decimal, overwrites the positions the bit maps identified with the CVC3 digits, the unpredictable number digits and the low-order ATC digits, and sends the resulting track image onwards in an ordinary magnetic-stripe-format authorisation. To every system between the terminal and the issuer, it looks like a swipe. Only the issuer, which knows the card's CVC3 key and the expected ATC, can tell that it was not.

A published capture of a Mastercard PayPass card makes this concrete. The card returned application identifier `A0 00 00 00 04 10 10` and, in its magnetic-stripe record, these values:

```
9F6C  02  00 01
9F62  06  00 00 00 00 01 C0
9F63  06  00 00 00 00 00 3C
9F64  01  03
9F65  02  00 E0
9F66  02  00 1E
9F67  01  03
```

`9F64` and `9F67` are both `03`: three ATC digits go into each track. `9F65` is `00E0` and `9F66` is `001E`, two bit maps marking disjoint runs of digit positions in the track 2 discretionary data — three positions for the CVC3, four for the unpredictable number and counter. The static Track 2 Data in `9F6B` began `...D 1103 502 0000000...`, showing the hexadecimal `D` separator, an expiry of March 2011, a service code of `502`, and a discretionary data field of zeroes waiting to be overwritten.

Read four times in quick succession, the same card produced:

```
;5XXXXXXXXXXXXXX2=11035020000072029191?
;5XXXXXXXXXXXXXX2=11035020000018129281?
;5XXXXXXXXXXXXXX2=11035020000054629311?
;5XXXXXXXXXXXXXX2=11035020000099829771?
```

Identical PAN, identical expiry, identical service code, four different discretionary data fields — the CVC3 digits and the counter digits, changing on every read exactly as the bit maps promise. The same capture also shows the contactless interface reporting service code `502` where the physical stripe on the same card carried `101`, and substituting `SUPPLIED/NOT` for the cardholder name, both of which are ordinary personalisation practice and neither of which is a security control.

Visa's equivalent for the MSD path is dCVV, a three-digit dynamic value derived using the ATC and carried in the track 2 discretionary data. Issuer guidance circulated in the United States is explicit about the obligation: "always validate for Dynamic Card Verification Value (dCVV) on POS entry mode 91 transactions and decline if dCVV fails", and where the issuer relies on a scheme service to do the validation, "check the authentication result field and decline if authentication fails."

MSD is a legacy path and is being removed. In the United States and Canada, new point-of-sale terminals have been required since 18 October 2019 to support only EMV mode for contactless, excluding magnetic-stripe mode; Visa set an April 2019 deadline for terminals to retire MSD in favour of qVSDC. The stripe itself is following: Mastercard has said that newly issued credit and debit cards will not be required to carry a magnetic stripe from 2024 in most markets and from 2027 in the United States, that no new Mastercard credit or debit cards will be issued with a stripe from 2029, and that no such cards will have stripes at all from 2033. All of these dates are as of writing and are the kind of commitment that moves.

The naming table

Scheme	Stripe value	Printed value	Chip value	Contactless dynamic value
Visa	CVV	CVV2 (3 digits, reverse)	iCVV	dCVV
Mastercard	CVC 1	CVC 2 (3 digits, reverse)	Chip CVC	CVC3
American Express	CSC	CID (4 digits, front)	—	—
Discover	—	CID (3 digits, reverse)	—	—
JCB	CAV	CAV2 (3 digits, reverse)	—	—
China UnionPay	CVN	CVN2 (3 digits, reverse)	—	—

The dashes are not assertions that the scheme lacks the mechanism; they mark places where the scheme does not publish a distinct public name for it. Generic industry usage covers the printed value with **CSC**, card security code, and PCI documentation uses "card verification code or value" precisely to avoid choosing a dialect.

Why CVV2 must never be stored

PCI DSS divides account data into cardholder data and **sensitive authentication data**. The PCI Security Standards Council defines SAD as "security-related information used to authenticate cardholders and/or authorize payment card transactions", and states that this "includes, but is not limited to, card verification codes, full track data (from magnetic stripe or equivalent on a chip), PINs, and PIN blocks."

The operative requirement in PCI DSS v4.0.1 is 3.3.1: "SAD is not retained after authorization, even if encrypted. All SAD is rendered unrecoverable upon completion of the authorization process." It is broken out into three sub-requirements covering the full contents of any track (3.3.1.1), the card verification code (3.3.1.2 — "The card verification code is not stored upon completion of the authorization process"), and the PIN and PIN block (3.3.1.3). Requirement 3.3.2 permits SAD to exist before authorisation completes only if "stored electronically prior to completion of authorization is encrypted using strong cryptography", and 3.3.3 carves out the only exception, for issuers and issuer processors, limiting any stored SAD "to that which is needed for a legitimate issuing business need" and requiring it to be encrypted with strong cryptography.

Read that carefully. Encryption is not a defence for retained CVV2. Tokenisation of the PAN is not a defence. There is no retention period, no "we keep it for seven days for chargebacks", no exception for a merchant with a good reason. There is no good reason, because the value's only function is to be presented once at the moment of authorisation and its only property of interest is that it is not held anywhere in bulk.

The requirement bites in places that surprise engineers who have only ever thought about the primary database:

- **Application logs.** A payment request logged at debug level with its full body is a PCI DSS 3.3.1.2 violation, and it is by an enormous margin the most common one.
- **Crash dumps and exception reports.** A framework that serialises the request object into an error tracker has just posted the code to a third party.
- **HTTP access logs and proxies.** A card form submitted by GET, or a badly built API that accepts the code as a query parameter, writes it into every log on the path.
- **Message queues, replay buffers and event streams.** Anything with a retention setting is storage.
- **Call recordings.** Telephone orders where the cardholder reads the code aloud require pause-and-resume recording or DTMF suppression; the recording is storage.
- **Paper.** PCI's own guidance addresses this directly, noting that where "card verification codes are stored on paper media prior to completion of authorization, a method of erasing or covering the codes should prevent them from being read after authorization is complete" — literally, cutting the code out with scissors or blacking it out with an opaque permanent marker.
- **Backups and analytics warehouses.** A nightly extract that includes the order payload propagates the violation into systems nobody has scoped.

There is one more consequence that catches product teams. Because the code cannot be retained, **recurring and merchant-initiated payments cannot resupply it.** A subscription billed monthly presents the code once, on the first transaction, and never again. This is why Mastercard's chargeback rules for recurring transactions let the merchant point back at the original authorisation and the `M` result stored against it, and why any product design that assumes "we will just re-verify the CVV each month" is not implementable. Store the result. Never the value.

Failure modes, and what each one is telling you

When these values are wrong in production, the symptom is usually a decline with an unhelpful response code, and the diagnosis lies in matching the interface to the value. This table is the one worth keeping.

Symptom	Likely cause
Chip transactions decline; swipes of the same card work	Terminal or gateway is reporting a chip read with a magnetic-stripe entry mode, so the issuer computes CVV1 against an iCVV
Contactless declines while contact chip works on the same card	Reader is in the wrong operating mode, or the track image was reassembled after the CVC3 digits were substituted
All chip transactions from one terminal model fail at one issuer	The terminal is normalising or re-padding tag 57, destroying the discretionary data that carries the iCVV
CVV2 result U on every transaction to a particular issuer	That issuer has not supplied verification keys to the scheme; the check is not being performed at all, by anyone
CVV2 result S	The merchant's message said the code was absent or illegible; the issuer's record says the card has one
A cloned stripe works when it should not	The issuer is not validating that a stripe-entry transaction carries the stripe's value, or is taking the service code from the incoming track data rather than its own record

Every one of those is a mismatch between an interface and a value that belongs to a different interface. That is the whole subject. Four values, four doors, and a system whose security depends on nobody confusing them — including, and especially, the people building it.

13.98 Common wrong ideas

Wrong: There is one security code on a card, the three digits on the back. **Right:** There are at least four values from the same family of algorithm — CVV1 in the stripe, CVV2 in ink, iCVV in the chip, dCVV or CVC3 manufactured per tap — and presenting one at another's interface is precisely the event the system is built to detect.

Wrong: The bank keeps your CVV2 in a column somewhere and can look it up. **Right:** Nothing is stored but the keys; each value is recomputed from scratch inside a hardware security module at authorisation time, which is why no issuer can read the printed code back to you and no screen anywhere displays it.

Wrong: iCVV is what stops a chip card being cloned. **Right:** iCVV stops chip-sourced static data being reused at the stripe door and nothing more; what makes a chip transaction impossible to counterfeit is the Application Cryptogram, computed with a key the chip will not disclose.

Wrong: CVV, CVC and CID are different mechanisms. **Right:** They are the same idea in different schemes' dialects — Visa says CVV, Mastercard says CVC, American Express and Discover say CID, JCB says CAV, China UnionPay says CVN — and a system handling several is being told the same thing twice in two languages.

Wrong: A CVV2 result of `N` means the transaction was declined. **Right:** The result is not the authorisation decision; an issuer may return an approval with a result of `N`, and whether to proceed is the merchant's own risk decision.

Wrong: The printed code may be kept encrypted for a few days to handle chargebacks. **Right:** PCI DSS 3.3.1 forbids retention after authorisation even if encrypted, with no retention period and no merchant exception; the defence in a recurring-transaction chargeback is the stored `M` result, never the value.

Wrong: Three digits is far too short to be worth anything. **Right:** The shortness is deliberate — guessing a code for a known card number takes about five hundred attempts on average — and the defence is the issuer's velocity controls and the scheme's rules on repeated declines, not the length.

Wrong: Because iCVV is computed with a service code input of `999`, the chip's track data must carry `999` in its service code field. **Right:** The `999` is fed into the calculation and is not a claim about what is encoded; Visa's warning about `000` and `999` appearing on counterfeit stripes only makes sense because the encoded field and the calculation input are separable things.

Wrong: A terminal may safely normalise or re-pad tag `57` before forwarding it. **Right:** The full unmodified Track 2 Equivalent Data including the issuer discretionary data must be transmitted, and stacks that parse tag `57` into fields and reassemble it destroy the iCVV, which is why every chip transaction from one terminal model can fail at one issuer.

Wrong: A stolen database of card numbers gives a thief everything needed for a card-not-present purchase. **Right:** The printed code is not in the card's machine-readable data anywhere, so it is not in the database and not on a skimmed stripe; the thief would have had to hold the card and turn it over.

13.99 Chapter summary in 20 lines

1. A modern payment card carries at least four values that all look like three unremarkable decimal digits, and they are not interchangeable.
2. They are conventionally CVV1 encoded in the magnetic stripe, CVV2 printed in ink, iCVV stored inside the chip, and dCVV or CVC3 manufactured fresh for every contactless tap.
3. All of them come out of the same algorithm in the same hardware security module, from the primary account number, the expiry date and a three-digit service code.
4. The service code is the only input that varies — the card's own code for CVV1, `000` for CVV2, `999` for iCVV — and changing it changes every digit of the answer with no pattern connecting the results.
5. Nothing is stored except the key pair, so each value is recomputed and compared at authorisation time, which is why an issuer can regenerate your printed code but cannot look it up.
6. CVV1 lives in the discretionary data of the encoded tracks, and exists to prove that whoever presented the card had seen a real stripe rather than merely learned a number from a receipt.
7. CVV2 is printed and never encoded anywhere in machine-readable form, which is its entire purpose: a stolen database or a skimmed stripe does not contain it.
8. iCVV lives in the discretionary data inside the chip's Track 2 Equivalent Data, tag `57`, and differs from the stripe's value so that data lifted from a chip fails when written onto a blank stripe.
9. That makes iCVV a cross-channel defence rather than a chip defence, and the popular explanation that it is what secures chip transactions is backwards.
10. What actually secures a chip transaction is the Application Cryptogram, a message authentication code over the amount, currency, date, terminal country, an unpredictable number and the card's own transaction counter, computed with a key that never leaves the silicon.

11. dCVV and Mastercard's CVC3 are dynamic: the chip mixes its ever-increasing counter with a number the reader has just invented and produces digits that have never existed before and will never be valid again.
12. In magnetic-stripe data mode the card hands the reader a track image with holes in it plus bit maps saying where to write the fresh digits, so everything downstream of the terminal sees an ordinary swipe.
13. A published capture of one contactless card read four times in a row shows an identical account number, expiry and service code with four different discretionary data fields, which is that mechanism made visible.
14. POS entry mode is load-bearing, because it tells the issuer which value to compute, and a terminal that reads the chip but reports a swipe will have chip data checked against the stripe's value and decline.
15. Terminals must transmit tag `57` unmodified, and software that trims, re-pads or normalises the discretionary data destroys the iCVV in the process — a real and recurring integration defect.
16. The mechanism is sound but its deployment is uneven, and in 2020 both Cyber R&D Labs and Gemini Advisory documented issuers who were not validating the stripe's value on stripe transactions, so chip-harvested iCVV worked on counterfeit stripes.
17. A subtler failure in the same family is taking the service code from the incoming track data rather than the issuer's own record, which lets a fraudster encode `999` on a counterfeit stripe and have the issuer helpfully compute the chip's value.
18. The printed code travels in scheme-specific fields — Visa Field 126 usage 10, Mastercard DE 48 subelement 92 — and returns a result of `M`, `N`, `P`, `S` or `U`, which is durable evidence and not an authorisation decision.
19. PCI DSS classes the printed code as sensitive authentication data and forbids retaining it after authorisation even encrypted, which reaches debug logs, crash dumps, access logs, queues, call recordings, paper and backups, and means a recurring payment can never resupply it.
20. Every production failure in this area has the same shape — an interface presented with a value belonging to a different interface — which is the whole subject of the chapter.

Sources: ISO/IEC 7813:2006 track structures; EMV 4.x Book 3 tag definitions (`57`, `5F30`, `5F34`, `9F1F`, `9F10`, `9F26`, `9F36`, `9F37`); EMVCo EMV Contactless Specifications for Payment Systems, Book C-2 (Kernel 2) data objects `56`, `9F60`–`9F67`, `9F6A`–`9F6C`; PCI SSC glossary and PCI DSS v4.0.1 requirements 3.3.1–3.3.3; Mastercard Switch Rules and Security Rules and Procedures (DE 48 subelements 87 and 92); Visa VisaNet authorisation technical specifications (Fields 44.10 and 126.10) and Visa EMV Chip News, October 2019; IBM cryptographic services documentation on the Visa CVV algorithm; Visa and ICBA issuer guidance on POS entry modes and dCVV validation; Gemini Advisory / Recorded Future, "Cybercriminals Deploy EMV-Bypass Cloning" (2020); Cyber R&D Labs card cloning research as reported by Krebs on Security, July 2020; Mastercard magnetic stripe retirement announcement, 2021; Secure Technology Alliance, "Card Payments Roadmap in the United States"; published contactless card capture data (Ambimat Electronics service code analysis).

4

CHAPTER

The Chip Is a Computer

14.0 What this chapter gives you

1. You will be able to state precisely what separates a chip from a magnetic stripe: a stripe is a recording and a chip is a correspondent, so a stripe reader takes and a chip reader asks.
2. You will be able to read a terminal log and name each exchange in it, from application selection through GET PROCESSING OPTIONS and READ RECORD to GENERATE APPLICATION CRYPTOGRAM.
3. You will be able to decode a command by hand from its class, instruction and parameter bytes, and explain why the trailing `00` on almost every EMV command means "send me everything you have" rather than padding or a null argument.
4. You will be able to decode an Application File Locator four bytes at a time, recover the short file identifier from the top five bits, and build the matching READ RECORD parameter byte from it.
5. You will be able to explain why records with short file identifiers 1 to 10 have their template tag and length stripped before they feed the authentication hash while records from 11 to 30 go in whole, and why that defect reaches production so often.
6. You will be able to say what static, dynamic and combined data authentication each prove and each fail to prove — that the data was signed, that the card is present, and that the card, the decision and the transaction are one indivisible statement.
7. You will be able to fill in a Processing Options Data Object List correctly, applying padding and truncation rules that run one way for numeric formats and the other way for everything else.
8. You will be able to look at a GENERATE AC exchange in which the terminal asked for an offline approval and the card returned an online request, and explain why the card may always answer weaker than it was asked but never stronger.
9. You will be able to explain why reading a chip is not an attack, and why any reader obtaining the account number, expiry, name and usage rules from any card is working exactly as designed.
10. You will be able to say why "the chip killed cloning" is true of one attack and one only, and where the fraud went instead.

Hold a card with the contact plate facing you and you are looking at eight gold pads, five of which do anything. Behind them, embedded in a cavity milled into the plastic, is a die perhaps a couple of millimetres square containing a processor, some read-only memory holding a program, some non-volatile memory holding data and keys, and — on any card issued this century — hardware dedicated to arithmetic on numbers hundreds of digits long.

This is the fact most explanations of card payments skip. The magnetic stripe is a *recording*. The chip is a *correspondent*. A stripe reader takes; a chip reader asks. Everything that follows in this volume — the cryptogram, the decision to go online, the cardholder verification list, the contactless tap — follows

from that one structural difference. It is also why counterfeit card fraud collapsed, and not because the chip is hard to read: reading a chip is trivial, and any developer with a fifteen-pound reader can pull a card number and expiry date off one in under a second. The chip works because there is a category of question it answers differently every time, and a category of secret it will never disclose however the question is phrased.

This chapter takes that conversation apart, command by command. By the end you should be able to read a terminal log, name each exchange, decode an Application File Locator by hand, and say precisely what static, dynamic and combined data authentication each prove and each fail to prove.

The plain version

Imagine two ways of proving who you are at a door.

The first way: you hand over a photocopy of your passport. The doorman reads it, sees your name and your photograph, and lets you in. It works. It is also completely useless the moment somebody else gets hold of a photocopier, because a copy of a copy is indistinguishable from the original. That is the magnetic stripe. Everything on it is written down. Anyone who can read it can write it out again, and the new one works exactly as well as the old one.

The second way: there is no photocopy. There is a small clerk in a locked room behind a letterbox. You cannot see him and you cannot go in. You slide questions through on slips of paper and he slides answers back. He is helpful — he will tell you the account number, the expiry date, the name — but he will not, under any circumstances, tell you the secret he uses to sign his answers. And crucially, some of the questions you may ask are ones where the *answer depends on the question*. If you ask "what is 8,391 plus your secret?" and he replies with a number, that number is worthless to anybody who asks him something different tomorrow.

That is the chip.

The conversation, one line at a time

Here is a real coffee. Four pounds twenty, a Tuesday morning, a card inserted into a countertop terminal. What follows is not a metaphor; it is the actual sequence, with the technical names given in brackets so that you can find them again in the second half of the chapter.

"What can you do?" The terminal has no idea what is in front of it. It might be a Visa debit card, a Mastercard credit card, a German girocard, or a transport card that is not a payment card at all. So it asks the chip for its directory of programs. The chip replies with a short list — say two entries, "Visa Credit" and "Visa Debit", each with a priority number — and the terminal picks the highest-priority one it also supports. (*Application selection, using the payment system directory.*)

"Right. I want to talk to Visa Debit." The terminal opens that program. The chip replies with a greeting that includes something surprising: a shopping list. It says, in effect, "before I will start a transaction, tell me nine things, in this exact order and at these exact sizes." (*SELECT of the application; the card returns a Processing Options Data Object List.*)

"Here is your list." The terminal fills it in: the amount is 000000000420, the country is 0826, the currency is 0826, today's date, the type of transaction, a five-byte block of its own findings so far, and — this one matters — a fresh four-byte random number it has just generated and has never used before. (*GET PROCESSING OPTIONS.*)

The chip now knows a transaction has begun. It replies with two things: a two-byte summary of what it is capable of, and a *map* listing which of its internal files the terminal should read, which records in

each, and how many of those records form the material to be authenticated. (*Application Interchange Profile and Application File Locator.*)

"Show me those files." Out come the card number, the expiry date, the cardholder name, the country of issue, rules about what the card may be used for, and a bundle of certificates. (*READ RECORD.*)

"Prove you are not a photocopy." The terminal has a card scheme's public key burned into it. The certificates chain back to that key: the scheme signed a certificate for the issuing bank, and the bank signed one for this chip. The terminal walks the chain, then throws the chip a challenge — a number it has just invented — and asks it to sign. Only this chip can produce the right signature, because only this chip holds the matching private key, and that key has never left the silicon.

"Here is the transaction. What is your verdict?" The terminal hands over the amount, the currency, the date, its own risk findings and another unpredictable number. The chip runs its own checks — how many transactions it has approved offline since it last spoke to its bank, whether the amount exceeds its limit — and produces a cryptogram computed from the transaction with a secret key it shares with the issuing bank. It also says what that cryptogram means: approved offline, declined offline, or *ask the bank*. (*GENERATE APPLICATION CRYPTOGRAM.*)

For a four-pound-twenty coffee most cards say "ask the bank", and the cryptogram goes off into the network. That journey is Volume III; what the cryptogram *is* and what key computes it is chapter 15.

The one thing the clerk will never do

Every command listed above is a question. There is no command in the whole specification that says "give me your private key", and none that says "give me the secret you share with your bank". The chip will sign things with those keys all day long. It will not disclose them.

That is not a policy, and not a setting an issuer can get wrong. No instruction in the instruction set would cause the chip to output them, and the program sits in read-only memory laid down at manufacture. To get the keys you would have to attack the silicon: decapsulate the die and either probe it or infer the key from variations in the power it draws while computing. Those attacks exist, and they are why chips are certified before a scheme will let them be issued. But they cost money, take equipment and destroy the card. Photocopiers do not.

Why the counter matters

One more detail worth carrying to dinner. The chip holds a counter that goes up by one every time a transaction starts and never goes down. It is two bytes wide, so it counts to 65,535, and every cryptogram includes the current value.

This is the quiet killer of replay. Suppose somebody records a complete, valid conversation — every question, every answer, cryptogram included — and plays it back tomorrow. The recorded cryptogram carries counter value 137, but the real card has been used eleven more times since, so the bank has seen 148. A counter that repeats or goes backwards is a card being replayed, and the issuer can see it in the message. The bank does not have to guess whether the transaction is genuine. It can check.

Where the plain version stops being true

The chip is not a computer in the sense you are picturing

It has no clock, no battery and no power of its own. It is parasitic on the terminal, which supplies voltage on one contact and a clock on another; the chip computes only while both are present. Pull the card out mid-transaction and it does not "notice" — it simply stops existing until somebody powers it

again. It has no idea whether the last transaction was four seconds ago or four years ago, it must ask the terminal for the date, and it cannot tell whether the terminal is lying.

Nor can the chip start anything. It has no way to say "excuse me" through the letterbox. It receives a command, computes, returns a response, and waits. This is why "the card decides whether to approve the transaction" is a useful but misleading simplification: the terminal decides what question to ask, and the card can only choose among the answers to *that* question — and, as we will see, it may return an answer weaker than the one requested but never a stronger one.

Offline authentication proves the card is genuine, not that the terminal is

The certificate chain runs one way. The chip proves itself to the terminal; nothing in the ordinary transaction flow requires the terminal to prove anything to the chip, and there is no practical way it could, because a terminal is a commodity device in a shop and the chip cannot check a revocation list or even know today's date.

The consequence surprises people: reading a chip is not an attack. Any reader may power any card, run selection, GET PROCESSING OPTIONS and READ RECORD, and obtain the primary account number, the expiry date, the cardholder name and the card's usage rules. Contactless cards will do this over the air. That is by design — those are the fields a merchant needs. The security does not come from hiding them. It comes from the fact that having them does not let you produce a cryptogram.

"The chip killed cloning" is true of one attack and one only

It killed the counterfeiting of magnetic stripes for use at chip-reading terminals. It did nothing about card-not-present fraud, and the fraud did not stop; it moved. UK Finance's twenty-year retrospective, published on 14 February 2026, records a 95 per cent reduction in losses to counterfeit card fraud over the two decades since Chip and PIN replaced signature — and in the same breath records £215 million lost to remote purchase fraud in the first half of 2025 alone.

There is a second qualification. For years a great many cards described as "chip cards" performed only *static* data authentication, which is a photocopy again — a photocopy with a genuine signature on it. The signature verifies; it is simply not fresh, because the signed material is identical on every transaction, and a device that has recorded it once can present it again. Whether a given card does this is an issuer decision, and the later sections explain exactly what the difference costs.

There is not one conversation; there are several specifications

The dialogue above is the contact interface, governed by the four EMV books. Contactless runs a related but distinct process, with its own entry-point specification and a family of kernel specifications, one per scheme, each with its own command sequence, data elements and decision logic. A Visa tap and a Mastercard tap do not execute the same steps. Treating "EMV" as a single protocol is the commonest error in engineering documentation about cards.

The technical version

What EMV actually is, and what it sits on

EMV is not a standard in its own right. It is a profile of the ISO/IEC 7816 series for integrated circuit cards. The current release, version 4.4, dated October 2022, comprises four books:

Book	Title
1	Application Independent ICC to Terminal Interface Requirements

Book	Title
2	Security and Key Management
3	Application Specification
4	Cardholder, Attendant, and Acquirer Interface Requirements

Book 3 section 1.3 states the relationship precisely: "This specification is based on the ISO/IEC 7816 series of standards and should be read in conjunction with those standards. However, if any of the provisions or definitions in this specification differ from those standards, the provisions herein shall take precedence." Version 4.4 also incorporates two consequential Specification Bulletins: no. 175, on Application Selection Registered Proprietary Data, and no. 243, which introduced Extended Data Authentication and Offline Data Encipherment. The Level 1 electrical and transport requirements now live in a separate document, the EMV Contact Interface Specification, cited by Book 3 as a normative reference.

Chapter 11 covered the physical layer: the ISO/IEC 7816-2 contact assignment, the 5 V supply, the 1 to 5 MHz clock, the answer to reset, and the protocols T=0 and T=1. This chapter starts one layer up.

The APDU: the shape of every question and every answer

Book 3 Figure 1 gives the command APDU as a mandatory four-byte header followed by a conditional body:

```
CLA  INS  P1  P2  [Lc]  [Data]  [Le]
```

and Figure 2 gives the response as a conditional body followed by a mandatory two-byte trailer:

```
[Data]  SW1  SW2
```

`Lc` is the number of bytes in the command data field; `Le` is the maximum expected back. Book 3 is unambiguous about one convention that trips up newcomers: "When Le is present and contains the value zero, the maximum number of available data bytes (up to 256) is expected. When required in a command message, Le shall always be set to '00'." The trailing `00` on almost every EMV command is therefore neither padding nor a null argument. It means "send me everything you have".

ISO/IEC 7816-4 defines four cases by whether data flows in each direction — case 1 neither, case 2 response only, case 3 command only, case 4 both — and EMV inherits them. Under T=0 they do not map cleanly onto single exchanges, which is why logs contain apparent duplicates: a card may answer `61 xx`, meaning "executed, xx bytes waiting, fetch them with GET RESPONSE", or `6C xx`, meaning "your Le was wrong, the correct value is xx". Book 1 Annex A works through all four cases and both procedure bytes. A library that ignores `6C xx` passes every test in the lab and fails intermittently in the field.

Book 3 Table 2 assigns the most significant nibble of the class byte: `'0'` is an inter-industry command defined by ISO/IEC 7816, `'8'` is a command proprietary to EMV, and any other value is out of scope. The least significant nibble carries secure messaging and logical channel indications, again per 7816-4. So `00 A4` is ISO's SELECT used as EMV requires; `80 A8` is an EMV-only command no general-purpose smart card would recognise.

Book 3 Table 3 gives the complete instruction set for financial transaction:

CLA	INS	Command
8x	1E	APPLICATION BLOCK

CLA	INS	Command
8x	18	APPLICATION UNBLOCK
8x	16	CARD BLOCK
0x	82	EXTERNAL AUTHENTICATE
8x	AE	GENERATE APPLICATION CRYPTOGRAM
0x	84	GET CHALLENGE
8x	CA	GET DATA
8x	A8	GET PROCESSING OPTIONS
0x	88	INTERNAL AUTHENTICATE
8x	24	PIN CHANGE/UNBLOCK
0x	B2	READ RECORD
0x	A4	SELECT
0x	20	VERIFY

That is the entire vocabulary: thirteen commands, of which a normal purchase uses five or six. The status bytes are the other half of the protocol. Those worth memorising:

SW1 SW2	Meaning
9000	Command successfully executed
61 xx	Executed; xx bytes available via GET RESPONSE
6C xx	Wrong Le; xx is the correct value
63 Cx	Verification failed; x tries remaining
6985	Conditions of use not satisfied
6A81	Function not supported
6A82	File not found
6A83	Record not found
6A88	Referenced data (data objects) not found

6A82 is not an error in the ordinary sense: it is how a card says "I do not have that application", and it is the expected response during selection.

The Application Identifier

An application is addressed by name, and the name is an Application Identifier. Its structure comes from ISO/IEC 7816-5: a Registered Application Provider Identifier of exactly five bytes, allocated by a registration authority, optionally followed by a Proprietary Application Identifier Extension of up to eleven bytes chosen freely by the provider. The AID is therefore 5 to 16 bytes long. On the card it is tag 4F, or the Dedicated File name, tag 84; in the terminal's configuration it is tag 9F06.

The RID identifies the scheme; everything after it is the scheme's own business — product, region, common debit variant. A working set:

AID	Application
A0000000031010	Visa Debit/Credit (Classic)
A0000000032020	V PAY
A0000000041010	Mastercard Credit/Debit (Global)
A0000000043060	Maestro
A0000000250000	American Express

AID	Application
A00000006510	JCB
A0000001523010	Discover / PULSE D-PAS
A0000002771010	Interac
A000000333010101	UnionPay Debit
D27600002545500100	ZKA girocard

Note the leading conventions — A0 for internationally registered providers, D2 for nationally registered ones — and note that the lengths differ. A terminal that assumes seven bytes fails on UnionPay and girocard, and that is a real and recurring defect.

Application selection

Selection begins with a directory. On contact the entry point is the Payment System Environment, a Directory Definition File named with the fourteen ASCII characters 1PAY.SYS.DDF01 — on the wire:

```
31 50 41 59 2E 53 59 53 2E 44 44 46 30 31
```

On contactless it is the Proximity Payment System Environment, 2PAY.SYS.DDF01, differing by one byte:

```
32 50 41 59 2E 53 59 53 2E 44 44 46 30 31
```

The terminal selects it by name per Book 1 Table 40: P1, the reference control parameter (Table 41), is '04' for selection by DF name, and P2, the selection options parameter (Table 42), is '00' for the first or only occurrence — '02' requests the next occurrence, which is how a terminal enumerates multiple applications sharing a partial AID.

A real trace, from a published analysis of a Visa contactless transaction:

```
C: 00 A4 04 00 0E 325041592E5359532E4444463031 00
R: 6F 37
     84 0E 325041592E5359532E4444463031
     A5 25
       BF0C 22
         61 20
           4F 07 A0000000031010
           50 0B 566973612043726564697
           87 01 01
           BF63 04 DF20 01 80
   9000
```

Read that structure carefully; it recurs everywhere. 6F is the File Control Information template; 84 the Dedicated File name, echoing what was selected; A5 the FCI Proprietary Template; BF0C the FCI Issuer Discretionary Data, where Book 3 section 5.1 requires any additional data element in a SELECT response to sit. 61 is a Directory Entry, holding 4F the ADF name, 50 the Application Label (566973612043726564697 is ASCII "Visa Credit") and 87 the Application Priority Indicator.

On the contact interface the PSE works slightly differently: the FCI Proprietary Template contains tag 88, the short file identifier of the directory elementary file, and the terminal issues READ RECORD against that SFI to walk the directory. Book 1 Table 46 gives the Payment System Directory Record Format and Table 47 the ADF Directory Entry Format.

If the PSE is absent the card returns 6A82 and the terminal falls back to trying each AID in its own list, one SELECT at a time, per Book 1 section 12.3.3. Against each it holds an Application Selection Indicator saying whether the card's AID must match exactly, including length, or only up to the length of the

terminal's — which is how a terminal holding the seven-byte Visa AID matches a card personalised with a longer, product-specific one. Book 1 section 12.4 governs final selection from the candidate list using the Application Priority Indicator, with cardholder confirmation where required.

Selecting the application itself returns a richer FCI:

```
C: 00 A4 04 00 07 A0000000031010 00
R: 6F 46
     84 07 A0000000031010
     A5 3B
       50 0B 5669736120437265646974
       87 01 01
       9F38 18 9F6604 9F0206 9F0306 9F1A02 9505
               5F2A02 9A03 9C01 9F3704
       5F2D 02 656E
       BF0C 08 9F5A 05 ...
   9000
```

`5F2D` is the Language Preference, `656E` being ASCII "en", `9F5A` the Application Program Identifier, and `9F38` the Processing Options Data Object List — the shopping list from the plain version, made concrete.

Data Object Lists: the card programs the terminal

A DOL is a concatenation of two-part entries, each a one- or two-byte tag followed by a one-byte length. It is *not* TLV data; it is a template telling the terminal what to concatenate, in what order, at what size. Book 3 section 5.4 explains why: to minimise processing in the card the constructed field carries no tags, so the card must know the layout in advance.

The PDOL above decodes to:

Tag	Length	Data element
`9F66`	4	Terminal Transaction Qualifiers
`9F02`	6	Amount, Authorised (Numeric)
`9F03`	6	Amount, Other (Numeric)
`9F1A`	2	Terminal Country Code
`95`	5	Terminal Verification Results
`5F2A`	2	Transaction Currency Code
`9A`	3	Transaction Date
`9C`	1	Transaction Type
`9F37`	4	Unpredictable Number

Thirty-three bytes. Book 3 section 5.4 also gives the rules every implementation gets wrong at least once:

If a tag in the DOL is unknown or names a constructed object, the terminal supplies that many hexadecimal zero bytes; likewise if the object is known but absent or inapplicable. If the DOL length is *shorter* than the actual data, the terminal truncates the leftmost bytes for numeric (`n`) format and the rightmost for everything else. If *longer*, it pads with leading zeroes for numeric, trailing `'FF's` for compressed numeric, trailing zeroes otherwise. Numeric goes one way and everything else the other; reversing that produces a cryptogram that fails at the issuer with no useful diagnostic.

The same mechanism appears three more times: CDOL1 (tag `8C`) and CDOL2 (tag `8D`) for GENERATE AC, the Transaction Certificate Data Object List for TC hashing, and the Dynamic Data Authentication Data Object List (tag `9F49`) for INTERNAL AUTHENTICATE.

GET PROCESSING OPTIONS

Book 3 section 6.5.8, Table 18:

Code	Value
CLA	'80'
INS	'A8'
P1	'00'; all other values are RFU
P2	'00'; all other values are RFU
Lc	var.
Data	PDOL related data
Le	'00'

The PDOL data is wrapped in a command template with tag '83'; if the card published no PDOL, the terminal sends the empty template 83 00. For a £4.20 purchase in the United Kingdom on 17 August 2026, filling in the PDOL above:

```
80 A8 00 00 23
   83 21
      36C04000        Terminal Transaction Qualifiers
      000000000420    Amount, Authorised          (£4.20)
      000000000000    Amount, Other
      0826            Terminal Country Code       (GB)
      0000000000      Terminal Verification Results
      0826            Transaction Currency Code   (GBP)
      260817          Transaction Date            (YYMMDD)
      00              Transaction Type            (purchase)
      7A19C4B2        Unpredictable Number
   00
```

Lc is '23', thirty-five bytes: two for the tag and length of the '83' template plus thirty-three of data. The amount is numeric format, two digits to the byte, right-justified with leading zeroes — 420 minor units, not 4.20. Chapter 9 explained why no floating point number ever touches this field.

The response comes back in one of two formats. Format 1 is a primitive object with tag '80' whose value is the Application Interchange Profile followed immediately by the Application File Locator, with no internal tags — you must know the first two bytes are the AIP and the rest the AFL. Format 2 is a constructed object with tag '77' containing ordinary TLV, AIP as '82' and AFL as '94'. Both are legal and both are deployed. From the same real trace:

```
R: 77 52
     82 02 0000                  Application Interchange Profile
     94 04 10040400              Application File Locator
     57 13 ...                   Track 2 Equivalent Data (PAN masked in source)
     5F20 02 202F                Cardholder Name
     5F34 01 00                  PAN Sequence Number
     9F10 07 06010A03A00000      Issuer Application Data
     9F26 08 1EB2BF209002EFD3    Application Cryptogram
     9F27 01 80                  Cryptogram Information Data (ARQC)
     9F36 02 0089                Application Transaction Counter (137)
     9F6C 02 0280                Card Transaction Qualifiers
     9F6E 04 20700000            Form Factor Indicator
   9000
```

That is a Visa contactless card, and it makes the earlier point concrete: this is not the contact flow. The card returned the cryptogram inside the GET PROCESSING OPTIONS response, before any GEN-

ERATE AC exists, because Visa's contactless kernel compresses the exchange to keep the tap under a few hundred milliseconds. Note too that the AIP is `'0000'`; the cardholder-verification signalling here rides in the Card Transaction Qualifiers, tag `9F6C` — precisely the field the published PIN-bypass attack against this kernel manipulates.

The Application Interchange Profile

Tag `82`, two bytes. Byte 1:

Bit	Meaning
b8	RFU
b7	SDA supported
b6	DDA supported
b5	Cardholder verification is supported
b4	Terminal risk management is to be performed
b3	Issuer authentication is supported
b2	On-device cardholder verification supported
b1	CDA supported

Byte 2 carries, among reserved bits, b8 for EMV mode and b1 for Relay Resistance Protocol support. Book 3 Annex C1, Table 41, is normative.

The Application File Locator

Tag `94`, always a multiple of four bytes, read four at a time:

Byte	Meaning
1	Short File Identifier in the five most significant bits; three least significant bits zero
2	First record to read
3	Last record to read
4	Number of records, starting at the first, included in offline data authentication

The AFL in the trace is `10 04 04 00`: one group. Byte 1 is `0x10`, `00010000` in binary; shift right three and you get `00010`, SFI 2. Records 4 to 4, so a single record. Zero records feed offline data authentication.

READ RECORD

Book 1 section 11.2 and Book 3 section 6.5.11. The command is `00 B2`, with P1 the record number and P2 the reference control parameter, in which the five most significant bits are the SFI and the three least significant are `100` — that is, `P2 = (SFI << 3) | 4`. For SFI 2, record 4:

```
C: 00 B2 04 14 00
R: 70 1A
     5A 08 ...               PAN (masked in source)
     5F28 02 ...             Issuer Country Code
     5F24 03 220731          Application Expiration Date
     9F07 02 C280            Application Usage Control
   9000
```

`0x14` is `00010100`: SFI 2, control bits `100`. The response is always wrapped in the READ RECORD Response Message Template, tag `70`. Short file identifiers are allocated by Book 3 Table 1:

SFI range	Meaning
1 to 10	Governed by the EMV specification
11 to 20	Payment system-specific
21 to 30	Issuer-specific

That boundary is not cosmetic; it is the most commonly mis-implemented rule in offline data authentication. Files with SFI 1 to 10 hold only data the card does not itself interpret, and when their records feed the authentication hash, *the tag `70` and its length byte are stripped* and only the value is used. For SFI 11 to 30 the *entire* record goes in, tag and length included. Get that wrong and every card with issuer-specific files fails offline authentication while every card inside the standard range passes — a defect that reaches production regularly, because test cards tend to use low SFIs.

Note also that a card may answer READ RECORD with `6C xx` if the terminal's Le was wrong, and the terminal must reissue with `Le = xx`.

The rest of the vocabulary

GET DATA (`80 CA`, P1 P2 carrying the tag of the wanted object, Book 3 section 6.5.7) retrieves single primitive objects living outside the file structure. Book 3 Table 32 lists them: the Application Transaction Counter (`9F36`), Last Online ATC Register (`9F13`), PIN Try Counter (`9F17`) and Log Format (`9F4F`).

GET CHALLENGE (`00 84 00 00 00`) asks the card for an eight-byte unpredictable number of its own, used when the card's CDOL demands an ICC Dynamic Number.

VERIFY (`00 20 00` P2) submits an offline PIN. Book 3 Table 24 codes P2 as the qualifier of reference data: `'80'` plaintext, `'88'` enciphered, with Table 25 giving the plaintext block format. Failure returns `63 Cx`, `x` being the tries left; at zero the application blocks. Chapter 17 covers cardholder verification in full.

INTERNAL AUTHENTICATE (`00 88 00 00` Lc Data `00`) submits authentication-related data — in practice the terminal's unpredictable number, laid out according to the DDOL — and returns the card's signature over it.

GENERATE APPLICATION CRYPTOGRAM (`80 AE` P1 `00` Lc Data `00`) is the decision command. P1, the reference control parameter, is coded by Book 3 Table 12:

b8 b7	Cryptogram requested
`0 0`	AAC — Application Authentication Cryptogram (decline)
`0 1`	TC — Transaction Certificate (offline approval)
`1 0`	ARQC — Authorisation Request Cryptogram (go online)
`1 1`	RFU

with b5 set to request a CDA signature and the rest RFU. So `P1 = '00'` asks the card to decline, `'40'` to approve offline, `'80'` to go online, and `'90'` to go online with a combined signature.

The response comes in the same two flavours as GET PROCESSING OPTIONS. Format 1, tag `'80'`, is Cryptogram Information Data, Application Transaction Counter, Application Cryptogram and op-

tionally Issuer Application Data, concatenated without tags. Format 2, tag `'77'`, is TLV, with `9F27`, `9F36` and `9F26` mandatory and `9F10` optional. The Cryptogram Information Data is coded by Book 3 Table 15:

b8 b7	Cryptogram returned
0 0	AAC
0 1	TC
1 0	ARQC
1 1	AAR — Application Authorisation Referral

with b6 and b5 for payment system-specific cryptograms, b4 indicating whether an advice message is required, and b3 to b1 a reason code: `000` no information given, `001` service not allowed, `010` PIN Try Limit exceeded, `011` issuer authentication failed.

A complete GENERATE AC exchange from a published contact-interface trace, the card's CDOL1 being `9F0206 9F0306 9505 5F2A02 9A03 9C01 9F3704 9F4C08 9F4502`:

```
C: 00 84 00 00 00
R: 8D51F46C9F405F71 9000            GET CHALLENGE returns ICC Dynamic Number

C: 80 AE 40 00 25
     000000000001          Amount, Authorised
     000000000000          Amount, Other
     0000000000            Terminal Verification Results
     0978                  Transaction Currency Code (EUR)
     090730                Transaction Date
     21                    Transaction Type
     2C764F65              Unpredictable Number
     8D51F46C9F405F71      ICC Dynamic Number
     D179                  Data Authentication Code
   00
R: 77 1E
     9F27 01 80                    ARQC
     9F36 02 0213                  ATC = 531
     9F26 08 2DF3833C61855BEA      Application Cryptogram
     9F10 07 06842300310208        Issuer Application Data
   9000
```

Read the two decisive values together. The terminal set `P1 = '40'`, asking for a Transaction Certificate — offline approval. The card returned Cryptogram Information Data `'80'`, an ARQC. The card overrode the terminal and forced the transaction online. That is card action analysis in one exchange: the terminal proposes, the card disposes, and the card may always return something weaker than requested but never something stronger.

Finally, `APPLICATION BLOCK`, `APPLICATION UNBLOCK`, `CARD BLOCK` and `PIN CHANGE/UNBLOCK` are the issuer script commands. The terminal never issues them on its own initiative; they arrive from the issuer inside the authorisation response and the terminal relays them. This is how a bank resets a PIN try counter on a card sitting in a wallet three thousand miles away.

Offline data authentication

Everything above establishes what the card says. Offline data authentication establishes whether to believe it, without going online.

The certificate chain

The terminal is loaded with the scheme's Certification Authority public keys, one or more per RID, each identified by a Certification Authority Public Key Index which the card supplies in tag `8F`. The terminal takes the first five bytes of the AID as the RID and, with the RID and the index, looks up the right CA public key in its own table.

The card carries an Issuer Public Key Certificate (tag `90`), an Issuer Public Key Remainder (tag `92`) where the key is too long to fit inside it, and an Issuer Public Key Exponent (tag `9F32`). The certificate is recovered with the CA public key using the message-recovery signature scheme of ISO/IEC 9796-2, and out comes a structure beginning `'6A'` and ending `'BC'`:

Field	Length (bytes)	Value
Recovered Data Header	1	`'6A'`
Certificate Format	1	`'02'`
Issuer Identifier	4	Leftmost 3–8 PAN digits, right-padded with `'F'`
Certificate Expiration Date	2	MMYY
Certificate Serial Number	3	Assigned by the CA
Hash Algorithm Indicator	1	
Issuer Public Key Algorithm Indicator	1	
Issuer Public Key Length	1	
Issuer Public Key Exponent Length	1	
Issuer Public Key or leftmost digits	NCA − 36	Right-padded with `'BB'` if short
Hash Result	20	
Recovered Data Trailer	1	`'BC'`

The terminal checks the header, format byte and trailer, recomputes and compares the hash over the recovered fields concatenated with the remainder and the exponent, checks the issuer identifier against the leading PAN digits, and checks the expiry. Only then does it hold a trusted issuer public key.

The key sizes are bounded. EMV Book 2 states that terminals "shall support keys up to 1984 bits (248 bytes) in length", and its Annex D1 sets the hierarchy: the Issuer Public Key length may be equal to or less than the CA Public Key length up to that 248-byte maximum, and the ICC Public Key and ICC PIN Encipherment Public Key lengths may be equal to or less than the Issuer Public Key length. Keys shrink as you go down the chain; they never grow.

Static Data Authentication

SDA uses the issuer key only. The card carries Signed Static Application Data in tag `93`, produced at personalisation by signing a hash of the card's static data with the issuer's private key. Recovery with the issuer public key yields:

Field	Length (bytes)	Value
Recovered Data Header	1	`'6A'`
Signed Data Format	1	`'03'`
Hash Algorithm Indicator	1	

Field	Length (bytes)	Value
Data Authentication Code	2	Issuer-assigned
Pad Pattern	NI - 26	`'BB'` repeated
Hash Result	20	
Recovered Data Trailer	1	`'BC'`

The authentication input is the concatenation of the records the AFL marked for authentication, assembled with the SFI rule described earlier, followed — where a Static Data Authentication Tag List (tag `9F4A`) is present — by the values of the objects it names. On success the terminal extracts the two-byte Data Authentication Code and stores it as tag `9F45`, which is why `9F45` appears in CDOL1: the card wants proof that the terminal really performed SDA.

What SDA proves: this card's static data was signed by an issuer whose certificate chains to the scheme. What it does not prove: that the card in the slot is the card the data came from. Every byte involved is fixed at personalisation; record it once and you can present it again on any device that speaks the protocol. SDA defeats a forger who alters the data. It does not defeat one who copies it wholesale.

Dynamic Data Authentication

DDA adds a key pair belonging to the individual chip. The card carries an ICC Public Key Certificate (tag `9F46`) signed by the issuer, plus the ICC Public Key Exponent (`9F47`) and Remainder (`9F48`). The recovered certificate has Certificate Format `'04'` and carries the full Application PAN in ten bytes, right-padded with `'F'`, rather than the issuer identifier of the `'02'` format.

Having verified the ICC public key, the terminal issues INTERNAL AUTHENTICATE carrying an unpredictable number of its own choosing. The card signs with its private key and returns Signed Dynamic Application Data, tag `9F4B`, which recovers to:

Field	Length (bytes)	Value
Recovered Data Header	1	`'6A'`
Signed Data Format	1	`'05'`
Hash Algorithm Indicator	1	
ICC Dynamic Data Length	1	LDD
ICC Dynamic Data	LDD	Generated by and/or stored in the ICC
Pad Pattern	NIC - LDD - 25	`'BB'` repeated
Hash Result	20	
Recovered Data Trailer	1	`'BC'`

The hash covers, among other things, the terminal's unpredictable number. That is the whole improvement: the signature is fresh, so a recording is useless, and producing it requires a private key that has never been transmitted. This is the point at which cloning stops being an economic proposition.

What standalone DDA does not do is bind the signature to the transaction. The card proves it is genuine; separately, it produces a cryptogram over the amount. A device sitting between card and terminal can let the authentication succeed and still interfere with the decision, because the two are not cryptographically joined.

Combined DDA/Application Cryptogram Generation

CDA joins them. Instead of a separate INTERNAL AUTHENTICATE, the terminal sets bit b5 of the GENERATE AC reference control parameter and the card returns the Signed Dynamic Application Data (`9F4B`) *in the GENERATE AC response*, alongside `9F27`, `9F36` and `9F10`. The ICC Dynamic Data inside that signature includes the Application Cryptogram itself and a hash over the transaction data, so one signature covers the card's identity, its decision and the transaction it applies to. Substituting any of the three invalidates all of them.

The terminal records the outcome in the Terminal Verification Results, tag `95`, five bytes, whose first byte is:

Bit	Meaning
b8	Offline data authentication was not performed
b7	SDA failed
b6	ICC data missing
b5	Card appears on terminal exception file
b4	DDA failed
b3	CDA failed
b2, b1	RFU

Those bits are then matched against the Issuer Action Codes and Terminal Action Codes to decide whether to approve, decline or go online. That is chapter 16.

Extended Data Authentication

Version 4.4 adds a fourth option. Specification Bulletin no. 243 introduced Extended Data Authentication and Offline Data Encipherment, built on elliptic curve cryptography rather than RSA: Book 3's abbreviation list now carries `XDA`, `ECC` and `EC-SDSA` for the Elliptic Curve Schnorr Digital Signature Algorithm, and Table 31 gives the data required for XDA alongside Table 29 for SDA and Table 30 for DDA and CDA.

The motive is arithmetic, not fashion. EMVCo's explanation is that in 2010 NIST announced it would not support EMV's longest RSA key length, 1984 bits, after 2030; longer keys mean longer computation, and a tap has a hard time budget. ECC reaches comparable strength at a fraction of the size. RSA key lengths remain under annual review.

Why the chip killed stripe cloning

Put the two technologies side by side and the asymmetry is total.

A magnetic stripe is a fixed pattern of magnetised particles encoding a fixed string of characters. The information required to make a working duplicate is exactly the information required to use the original, which is a way of saying the stripe has no secret at all. A skimmer read the tracks and a fifty-pound encoder wrote them to a blank. The counterfeit was not a good imitation of the card; it *was* the card, in every sense the acquiring system could test.

A chip has a secret and never says it. No command in Book 3 Table 3 outputs a private key or a shared symmetric key. The most complete possible read — every SELECT, every READ RECORD against every record named in every AFL, every GET DATA — yields the PAN, the expiry, the name, the certificates and the counters. It yields a card that can answer questions about itself. It does not yield

one that can sign a fresh unpredictable number under DDA, nor one that can compute an Application Cryptogram over a new amount, because both need key material no command will disclose.

Three further locks sit in the same door. The Application Transaction Counter makes replay visible to the issuer, because values that repeat or go backwards are detectable in the authorisation message. The unpredictable number makes replay useless to the terminal, because the challenge changes every time. And, as chapter 13 set out, the chip's Track 2 Equivalent Data (tag 57) is conventionally personalised with a different card verification value from the physical stripe's, so lifting that record onto a blank produces a stripe that fails verification at the issuer.

The result is in the fraud statistics. UK counterfeit card fraud losses were £129.7 million in 2004, up 17 per cent on £110.6 million in 2003, just as migration began: the Northampton trial ran in 2003, national rollout followed, and on 14 February 2006 signature ceased to be accepted for in-store chip card purchases. UK Finance's twentieth-anniversary assessment, published on that date in 2026, records a 95 per cent reduction in counterfeit card fraud losses over the intervening two decades. Chip and PIN now accounts for 30 per cent of UK card payments by value, averaging £93 a transaction against £17 for a contactless tap.

Three qualifications belong with that number. SDA-only cards were never protected in the way the summary implies, and a large installed base of them persisted for years. Magnetic stripe fallback — a terminal accepting the stripe when the chip fails to read — reopened the door wherever permitted, which is why schemes have progressively closed it. And the fraud did not vanish; it moved to the channel where no chip is present. That is Volume V.

The chip in one paragraph

An EMV chip is an ISO/IEC 7816 smart card running a payment application addressed by a 5-to-16-byte Application Identifier, speaking APDUs of the form CLA-INS-P1-P2-Lc-Data-Le and answering with data plus a two-byte status word. The terminal selects a directory (`1PAY.SYS.DDF01` on contact, `2PAY.SYS.DDF01` on contactless), builds a candidate list, selects an application, learns from the PDOL what transaction data the card requires, sends it in GET PROCESSING OPTIONS, gets back a capability profile and a file map, reads the named records, verifies the certificate chain up to a scheme public key it already holds, challenges the card to sign something fresh, and asks for a verdict with GENERATE AC. Static data authentication proves the data was signed; dynamic proves the card is present; combined proves the card, the decision and the transaction are one indivisible statement. Through all of it no instruction exists that will make the chip disclose its keys — which is why a card that answers questions cannot be photocopied, and a card that merely presents data always could.

14.98 Common wrong ideas

Wrong: The chip is a small computer with a clock and power of its own. **Right:** It is parasitic on the terminal for both voltage and clock and computes only while both are present, it cannot start anything, and it must ask the terminal for the date without any way of telling whether the terminal is lying.

Wrong: The card decides whether to approve the transaction. **Right:** The terminal decides which question to ask and the card may only choose among the answers to that question, and it may return an answer weaker than the one requested but never a stronger one.

Wrong: Reading data off a chip is an attack, so a contactless card that gives up an account number over the air is broken. **Right:** Any reader may power any card and obtain the account number, expiry, cardholder name and usage rules; the security does not come from hiding those fields but from the fact that holding them does not let you produce a cryptogram.

Wrong: Offline data authentication proves the card and the terminal to each other. **Right:** The certificate chain runs one way only, and nothing in the ordinary flow requires the terminal to prove anything, because a terminal is a commodity device and the chip cannot check a revocation list or even know today's date.

Wrong: Any card with a chip cannot be cloned. **Right:** A card performing only static data authentication carries a genuine signature over material identical on every transaction, so a device that recorded it once can present it again; freshness arrives only with dynamic authentication.

Wrong: "EMV" is a single protocol, so a Visa tap and a Mastercard tap run the same steps. **Right:** Contact runs the four EMV books while contactless runs an entry-point specification and a family of per-scheme kernels with their own commands, data elements and decision logic, and treating EMV as one protocol is the commonest error in engineering documentation about cards.

Wrong: A `6A82` in a trace is an error. **Right:** It is how a card says "I do not have that application", it is the expected response during selection, and it is what sends the terminal into trying each Application Identifier from its own list one at a time.

Wrong: Application Identifiers are seven bytes long. **Right:** An identifier is a five-byte registered provider identifier plus up to eleven bytes of proprietary extension, so lengths run from 5 to 16, and a terminal that assumes seven fails on UnionPay and girocard.

Wrong: A library that handles `9000` and `61 xx` has covered the protocol. **Right:** A card may also answer `6C xx`, meaning the expected-length byte was wrong and `xx` is the correct value, and an implementation that ignores it passes every test in the lab and fails intermittently in the field.

Wrong: Dynamic data authentication protects the transaction. **Right:** Standalone dynamic authentication proves the card is genuine without binding the signature to the amount, so a device between card and terminal can let it succeed and still interfere with the decision; only combined generation joins identity, decision and transaction into one signature.

14.99 Chapter summary in 20 lines

1. Behind the eight gold pads on a card, five of which do anything, sits a die a couple of millimetres square holding a processor, a program in read-only memory, non-volatile memory for data and keys, and hardware for arithmetic on very long numbers.
2. The structural difference from a magnetic stripe is that a stripe is a recording while a chip is a correspondent, so everything else in card security follows from the fact that a chip answers rather than presents.
3. Reading a chip is trivial and deliberate, because the account number, expiry, cardholder name and usage rules are exactly the fields a merchant needs.
4. What the chip will never do is disclose its private key or the symmetric key it shares with its issuer, because no instruction in the set would output them and the program sits in read-only memory laid down at manufacture.
5. EMV is not a standard in its own right but a profile of the ISO/IEC 7816 series, currently version 4.4 of October 2022 in four books, whose provisions take precedence wherever they differ from those standards.
6. Every exchange is a command of class, instruction and two parameter bytes with an optional body, answered with optional data and a mandatory two-byte status word.
7. The whole vocabulary is thirteen commands, of which an ordinary purchase uses five or six, and the status words carry as much of the protocol as the commands do.

8. An application is addressed by an Application Identifier of 5 to 16 bytes, and terminals that assume a fixed length break on real cards from real schemes.
9. Selection begins with a directory, `1PAY.SYS.DDF01` on contact and `2PAY.SYS.DDF01` on contactless, differing by one byte, and falls back to trying each identifier in turn when the directory is absent.
10. Selecting the application returns a Processing Options Data Object List, the card telling the terminal exactly what to concatenate, in what order and at what size, with no tags, so that the card is spared the work of parsing.
11. The Data Object List padding and truncation rules are asymmetric between numeric and non-numeric formats, and reversing them yields a well-formed command whose cryptogram fails at the issuer with no useful diagnostic.
12. GET PROCESSING OPTIONS returns an Application Interchange Profile saying what the card can do and an Application File Locator saying which records to read and how many of them feed offline data authentication.
13. READ RECORD retrieves those records, and the rule that low short file identifiers have their template tag and length stripped from the authentication input while high ones do not is the most commonly mis-implemented rule in the area.
14. Offline data authentication starts from a scheme public key already held in the terminal, walks a certificate chain through the issuer, and only then trusts anything the card says about itself.
15. Static data authentication proves that fixed data was signed by an issuer whose certificate chains to the scheme, which defeats a forger who alters the data and not one who copies it wholesale.
16. Dynamic data authentication adds a key pair belonging to the individual chip, which signs a number the terminal has just invented, so the signature is fresh and a recording is worthless.
17. Combined generation puts the cryptogram and a hash of the transaction inside that signature, so substituting any one of identity, decision or transaction invalidates all three.
18. Version 4.4 adds Extended Data Authentication on elliptic curves, because NIST will not support EMV's 1984-bit RSA after 2030 and a tap has a hard time budget.
19. GENERATE APPLICATION CRYPTOGRAM is the decision command, and a published trace in which the terminal asked for an offline approval and the card returned an online request is card action analysis in a single exchange.
20. The chip destroyed counterfeit stripe fraud — a 95 per cent reduction in the United Kingdom across two decades — because no command discloses a key, the transaction counter makes replay visible to the issuer and the unpredictable number makes it useless to the terminal, but the fraud did not vanish so much as move to the channel where no chip is present.

Chapter sources: EMV Integrated Circuit Card Specifications for Payment Systems v4.4 (October 2022), Book 3 — sections 1.3, 5.1, 5.3, 5.4, 6.1–6.3, 6.5, 9, 10.3, Tables 1–4, 10–18, 21–25, 29–32, 41, 46 and Figures 1, 2, 4, 5; EMV Book 1 v4.3 (November 2011), sections 10–12 and Tables 38–48; EMV Book 2, sections 5 and 6, 10.2.5.2 and Annex D1, for the certificate and signature formats and the 1984-bit bound; EMVCo Specification Bulletin no. 243 and EMVCo's FAQ "EMV Chip Specifications To Support Elliptic Curve Cryptography" (28 October 2021); EMV Contactless Specifications, Entry Point, section 3.3.1, for PPSE selection; ISO/IEC 7816-4 sections 5.3–5.4, ISO/IEC 7816-5 for RID and PIX, ETSI TS 101 220 for the 5-byte RID plus 11-byte PIX, ISO/IEC 9796-2 for the signature scheme. Traces from Basin, Sasse and Toro-Pozo, "The EMV Standard: Break, Fix, Verify", IEEE Symposium on Security and Privacy 2021, Appendix B, and the CardContact OpenSCDP EMV tutorial. Status words and AIDs cross-checked against EFTlab's references. Fraud figures from UK Finance, "20 years of change in payments — from Chip and PIN to mobiles" (14 February 2026), and APACS 2004 figures as reported by Pinsent Masons. Every hex trace printed here was re-parsed by hand against its stated lengths.

The Cryptogram

15.0 What this chapter gives you

1. You will be able to explain why a criminal holding a perfect byte-for-byte record of everything a card said last Tuesday cannot use that record to buy anything on Wednesday.
2. You will be able to say why the Application Cryptogram is a message authentication code and not a signature, and why the terminal that carries it cannot check a single bit of it.
3. You will be able to name the three cryptogram types and explain why a refusal is sealed as carefully as an approval.
4. You will be able to walk the three-level key hierarchy from an issuer master key, through a card-unique key derived with the account number and sequence number, to a session key derived with the transaction counter.
5. You will be able to assemble the byte string a cryptogram is computed over, pad it correctly, and explain why the mandatory `'80'` byte is appended even when the message is already a multiple of eight.
6. You will be able to read an Issuer Application Data field, pull out the Derivation Key Index, the Cryptogram Version Number and the Card Verification Results, and say why that version byte is the most load-bearing byte in the whole message.
7. You will be able to distinguish the two Authorisation Response Cryptogram methods and explain how the issuer's approval travels inside a MAC so that nobody in the chain can flip it.
8. You will be able to state precisely what a valid cryptogram proves and what it does not — not that a cardholder was present, not that a PIN was entered, not that the amount displayed matched the amount authenticated, and not which merchant was involved.
9. You will be able to give an honest account of the transaction counter: it makes replay detectable rather than impossible, and real issuer hosts apply a window rather than a hard ceiling for defensible reasons.
10. You will be able to diagnose a failing cryptogram from the pattern of which cards fail, knowing that the mathematics has not failed in the field and the byte strings fed to it fail constantly.

Every chapter of this volume so far has described something the card *has*. A number embossed on the front. A stripe with three tracks. A printed code on the signature panel. A chip that runs a program and answers questions. All of it is data, and all data can be copied.

This chapter is about the one thing on the card that cannot be copied, because it does not exist until the moment of the transaction and it is never the same twice. It is eight bytes long. It is called the Application Cryptogram, and it is the reason that a criminal holding a perfect byte-for-byte record of everything your card said last Tuesday cannot use that record to buy anything on Wednesday.

It is also the single most misunderstood object in payments. People who work with cards every day will tell you the chip "signs" the transaction, which is not quite what happens; that the cryptogram proves the cardholder was present, which it does not; or that the terminal checks it, which it emphatically cannot. Getting this right matters more than getting any other detail in this volume right, because when a cryptogram fails there is no diagnostic. The issuer's host sees a mismatch between sixteen hexadecimal characters it computed and sixteen it received, and it has no way of telling you which of the eleven inputs was wrong. Engineers lose weeks to this. The purpose of this chapter is that you should not.

The plain version

Imagine you and your bank have agreed on a secret recipe.

The recipe is not a password and it is not a code word. It is a *method*: a way of taking any piece of writing, stirring it through a fixed sequence of steps, and producing sixteen characters of gibberish at the end. The stirring is designed so that changing anything at all in the writing — one digit, one letter, a single space — changes every one of the sixteen output characters completely and unpredictably. There is no way to work backwards from the gibberish to the writing, and no way to produce the right gibberish for a given piece of writing unless you know the recipe.

You keep the recipe. The bank keeps the recipe. Nobody else has it.

Now you go to buy a coffee.

What gets written down

The till writes out a short statement of exactly what is about to happen. Not a description in words — a fixed list of facts, always the same facts in always the same order, because both you and the bank need to write out the identical statement or the gibberish will not match.

The statement for a coffee in Leeds on 17 August 2026 says:

- The amount is four pounds twenty.
- There is no cashback.
- The till is in the United Kingdom.
- The till's own safety checks found one thing worth mentioning: this purchase is above the amount the shop is allowed to approve on its own.
- The money is pounds sterling.
- The date is the seventeenth of August.
- This is a purchase, not a refund and not a cash withdrawal.
- Here is a number the till just made up on the spot: 9A1BC3D4. It has never used it before and will never use it again.

Then the card adds two facts of its own:

- Here is what I am capable of doing.
- **This is the sixty-first time I have ever been used.**

The card stirs that whole statement through the recipe and produces sixteen characters: `733084432BA24DAE`.

Why those last two facts are the clever part

The till's made-up number is there to defeat the eavesdropper. Suppose somebody has been recording everything your card says. They have last Tuesday's sixteen characters. To reuse them they would need today's statement to be identical to Tuesday's — but today's till invented a different number, so it is not identical, so Tuesday's gibberish is wrong. It is refused.

The card's own count is there to defeat something subtler: the shop itself. A dishonest shop could take a genuine statement and a genuine set of sixteen characters and send the pair to the bank twice, hoping to be paid twice. But the card counts every single use, and the count is inside the statement. The bank sees that use number sixty-one has already been paid for. The second copy is refused.

The count also does something else, which is quieter and more important. **The recipe changes every time the count changes.** It is not literally one fixed recipe; it is a family of recipes, and the count picks which member of the family is used today. So even if somebody somehow worked out the exact recipe used for purchase sixty-one, it would tell them nothing at all about purchase sixty-two.

Three possible answers

When the till hands the statement to the card, it is not just asking for a signature. It is asking a question, and the card gets to answer it. There are three answers, and each comes with its own set of sixteen characters.

The card can say **yes, on my own authority** — I know this cardholder, the amount is small, I have not been used much lately, go ahead without asking the bank. This answer is called a Transaction Certificate.

The card can say **ask the bank** — this needs a decision I am not entitled to make. This answer is called an Authorisation Request Cryptogram. The sixteen characters travel across the world to the bank, which recomputes them and compares.

The card can say **no**. This answer is called an Application Authentication Cryptogram, and — this is the part people find odd — it still comes with sixteen characters. The refusal is sealed too, so that nobody downstream can quietly turn a refusal into an approval.

And here is a detail worth carrying to dinner: the till *asks* for one of these three, and the card is allowed to give a worse answer than the one requested. A till can ask for "yes, on your own authority" and be told "ask the bank". It can ask for "ask the bank" and be told "no". It can never be told something better than it asked for. The terminal proposes; the card disposes.

The bank answers with a seal of its own

The bank recomputes the sixteen characters from its own copy of the recipe. They match. The bank checks the balance and decides to approve.

Now the bank has a problem. If it just sends back the word "approved", anybody in the chain — the shop's till, the shop's software, the phone line — could invent that word. So the bank does the same trick in reverse. It takes the card's sixteen characters, mixes in its answer, stirs the result through the shared recipe, and sends back its own short scramble.

The card receives it, does the same calculation, and if the two agree the card knows something quite specific: *the bank itself, and nobody in between, said yes to this exact transaction.* Only then will it act on instructions from the bank — reset a PIN counter, clear its offline spending allowance, or in the extreme case block itself.

That is the whole mechanism. Two seals, one from the card and one from the bank, over a statement that both of them wrote out identically, using a recipe that changes on every use.

Where the plain version stops being true

The recipe picture gets you to the right conclusion, and it hides four things that matter enormously in practice.

It is not a signature, and the terminal cannot check it

The word "sign" implies that anyone can verify and only one party can produce. That is asymmetric cryptography, and the Application Cryptogram is not that. It is a **Message Authentication Code** computed with a *symmetric* key. Card and issuer hold the same secret. Anybody who can verify a cryptogram can also forge one.

This has an immediate practical consequence that surprises people: **the terminal cannot verify the cryptogram at all.** It has no key and never will. When a terminal receives eight bytes back from a GENERATE AC command it has no idea whether they are correct, random, or fabricated. It copies them into the authorisation message and forwards them. Everything the terminal does to satisfy itself that the card is genuine — static, dynamic and combined data authentication — is a *separate* mechanism using public keys, which was Chapter 4's subject. The cryptogram is not part of it.

Because the key is symmetric, the security of the whole scheme rests on the key never leaving tamper-resistant hardware at either end: a chip on one side, a hardware security module on the other. A single issuer master key extracted from a poorly managed host is not one compromised card. It is every card in that key's range, forever.

There is no such thing as "the data the card signs"

The plain version implies a fixed list. There is not one. EMV Book 2 section 8.1.1 gives a table of ten elements and calls it the "recommended minimum set". It is a recommendation. What is actually included, and in what order, is fixed by the application's **Cryptogram Version Number**, which is a scheme-proprietary value published in Visa's, Mastercard's, Amex's, JCB's and UnionPay's own specifications and not in EMV's.

Two cards from the same issuer with different Cryptogram Version Numbers compute cryptograms over different byte strings. An issuer host that assumes one layout for all its cards will silently fail on the others. This is, by a wide margin, the most common cause of "the cryptogram does not verify and we cannot work out why".

It says nothing about the cardholder, the merchant, or what you agreed to

A valid ARQC proves that a chip holding the correct key participated in a transaction with those exact parameters. It proves nothing else.

It does not prove the cardholder was present. It does not prove a PIN was entered — cardholder verification is reported separately, in the Cardholder Verification Method Results and in the card's own Card Verification Results, and an attacker who can suppress the PIN check can still obtain a perfectly valid cryptogram. It does not prove the amount displayed to the customer matches the amount in the command; it authenticates the amount the *terminal* supplied. And it does not identify the merchant: neither the merchant identifier nor the terminal identifier is in the recommended minimum set.

The counter only stops replay if somebody checks it

The Application Transaction Counter makes replay *detectable*. It does not make it impossible. Detection requires the issuer to remember the highest counter value it has seen for that card and refuse anything at or below it.

Real issuer hosts are much less strict than that, and for defensible reasons: authorisations arrive out of order, offline transactions consume counter values that never reach the host at all, reversals and pre-authorisations complicate the sequence, and a card used at an unattended terminal may burn counter values with no corresponding message. Most hosts therefore apply a window rather than a hard ceiling, and some check nothing at all. A gap in the counter is evidence of offline activity, not of fraud; a repeat is evidence of something wrong, but not necessarily of an attack.

The technical version

The three cryptogram types

The terminal requests a cryptogram with the GENERATE APPLICATION CRYPTOGRAM command — class `'80'`, instruction `'AE'`, P2 `'00'` — and states which type it wants in P1, the reference control parameter. The top two bits carry the request; bit 5 additionally asks the card to return a combined dynamic signature.

P1 bits b8 b7	Requested
`0 0`	AAC — Application Authentication Cryptogram (decline)
`0 1`	TC — Transaction Certificate (approve offline)
`1 0`	ARQC — Authorisation Request Cryptogram (go online)
`1 1`	RFU

The card replies with the cryptogram and with a single byte, the **Cryptogram Information Data**, tag `9F27`, stating which type it actually generated. The top two bits use the same encoding as P1. Bits b6 and b5 flag a payment-system-specific cryptogram, bit b4 indicates that an advice message is required, and bits b3 to b1 carry a reason or advice code: `000` no information given, `001` service not allowed, `010` PIN Try Limit exceeded, `011` issuer authentication failed.

The industry is not consistent about `1 1` in the Cryptogram Information Data. Older editions of Book 3, and a good deal of kernel documentation still in circulation, label it AAR — Application Authorisation Referral, the card asking for a voice referral. Referral processing does not appear in EMV 4.3 Book 2 at all, current terminal specifications do not implement it, and current tag references treat the value as reserved. Treat any card that returns it as broken rather than as requesting a referral.

The response arrives in one of two shapes. Format 1, template tag `'80'`, is a primitive object: Cryptogram Information Data, then Application Transaction Counter, then Application Cryptogram, then optionally Issuer Application Data, concatenated with no tags and no lengths. Format 2, template tag `'77'`, is TLV-encoded, with `9F27`, `9F36` and `9F26` mandatory and `9F10` optional. A Common Core Definitions application must use format 2 and must include a 32-byte `9F10`.

The key hierarchy

Three levels, and everything else in this chapter depends on keeping them straight.

At the top sits the **Issuer Master Key for Application Cryptograms**, IMK-AC. One per issuer per key range, held in the issuer's hardware security module, never in a card. Under two-key Triple DES it is 128 bits; the AES option supports 128, 192 and 256.

From it, at personalisation, the issuer derives one **ICC Application Cryptogram Master Key**, MK-AC, per card. EMV Book 2 Annex A1.4 gives three optional methods, all taking the Application PAN and the PAN Sequence Number as input:

Method	Cipher	Rule
Option A (A1.4.1)	Triple DES only	Concatenate PAN with PAN Sequence Number; take the rightmost 16 digits as an 8-byte number `Y`, left-padded with hexadecimal zeroes if shorter. `ZL := DES3(IMK)[Y]`, `ZR := DES3(IMK)[Y XOR 'FFFFFFFFFFFFFFFF']`, and `MK` is `ZL` followed by `ZR`, with odd parity then forced on every byte.
Option B (A1.4.2)	Triple DES only	If the PAN is 16 digits or fewer, use Option A. Otherwise hash PAN concatenated with PAN Sequence Number under SHA-1, take the first 16 decimal digits of the 20-byte result as `Y`, and if there are fewer than 16 decimal nibbles, decimalise the remaining ones using the table A->0, B->1, C->2, D->3, E->4, F->5, appending from the left. Then continue from Option A step 2.
Option C (A1.4.3)	AES only	Concatenate PAN with PAN Sequence Number, left-pad with hexadecimal zeroes to a 16-byte number `Y`, and encipher under AES.

None of the three is mandatory; Book 2 says so explicitly. Issuers may adopt another method, and some do.

From MK-AC, at transaction time, both card and issuer derive a **session key**, SK-AC. Book 2 Annex A1.3.1 specifies the Common Session Key option. The diversification value `R` for Application Cryptogram and ARPC generation is the two-byte Application Transaction Counter followed by n-2 bytes of `'00'`. Where the key is longer than the block — Triple DES with a 128-bit key, or AES with 192 or 256 bits — two blocks are built from `R`:

```
F1 = R with byte 3 replaced by 'F0'
F2 = R with byte 3 replaced by '0F'
SK = leftmost k bits of { ALG(MK)[F1] || ALG(MK)[F2] }
```

Where key and block are the same size — AES-128 — the derivation collapses to `SK := AES(MK)[R]`. The same session key is used for every command in a single transaction, which is why the second GENERATE AC does not need a fresh derivation.

Not every application uses this. Visa Cryptogram Version 10 performs no session key derivation at all: the card master key *is* the cryptogram key. Visa CVN 18 and CVN 22, and Mastercard CVN 14 and CVN 15, use the Common Session Key method above. Mastercard CVN 12 and CVN 13 use an older Mastercard derivation that folds the terminal's Unpredictable Number into the session key as well as the counter.

A worked derivation

Take an issuer master key of `0123456789ABCDEFFEDCBA9876543210` — a test key, not a real one — a PAN of `4111111111111111` and a PAN Sequence Number of `01`.

Option A concatenates them to `41111111111111101`, eighteen digits, and takes the rightmost sixteen:

```
Y      = 1111111111111101
MK_AC  = A768CE9E1551FDE3A45B4C7C3DE9DC52
```

The card is on its sixty-first use, so the Application Transaction Counter is `003D`. The diversification value and the two derivation blocks are:

```
R      = 003D000000000000
F1     = 003DF00000000000
F2     = 003D0F0000000000
SK_AC  = C471A83E37AEF3FE5D4CEF548575DD9C
```

Everything from here uses `SK_AC` and nothing else. `MK_AC` is not touched again until the next transaction.

What goes into the MAC

Book 2 section 8.1.1 gives the recommended minimum set. Eight elements come from the terminal, two from the card:

Source	Element
Terminal	Amount, Authorised (Numeric)
Terminal	Amount, Other (Numeric)
Terminal	Terminal Country Code
Terminal	Terminal Verification Results
Terminal	Transaction Currency Code
Terminal	Transaction Date
Terminal	Transaction Type
Terminal	Unpredictable Number
Card	Application Interchange Profile
Card	Application Transaction Counter

For a Common Core Definitions application with Cryptogram Version `'5'` or `'6'`, Book 2 replaces that table with a mandatory eleven-element list in a fixed order — the ten above, in exactly that order, followed by the Issuer Application Data. Every real scheme implementation does something equivalent: it appends the card's own **Card Verification Results**, which live inside the Issuer Application Data, to the end of the terminal's eight elements and the card's two.

The terminal does not choose which elements to send. The card publishes **CDOL1**, tag `8C`, a Card Risk Management Data Object List naming the tags and lengths the terminal must concatenate into the first GENERATE AC command, and **CDOL2**, tag `8D`, for the second. The terminal fills the template in the order given. The Data Object List padding and truncation rules of Book 3 section 5.4 apply in full, and they are asymmetric: numeric-format elements are truncated from the left and padded with leading zeroes, everything else is truncated from the right and padded with trailing zeroes, and compressed numeric is padded with trailing `'FF'`s. Getting that backwards produces a well-formed command that yields a cryptogram nobody can verify.

Here is the actual byte string for our £4.20 coffee, purchased in the United Kingdom on 17 August 2026 at a terminal whose floor limit the amount exceeds:

Tag	Element	Bytes	Value
`9F02`	Amount, Authorised (Numeric)	6	`000000000420`
`9F03`	Amount, Other (Numeric)	6	`000000000000`
`9F1A`	Terminal Country Code	2	`0826`
`95`	Terminal Verification Results	5	`0000008000`
`5F2A`	Transaction Currency Code	2	`0826`
`9A`	Transaction Date	3	`260817`
`9C`	Transaction Type	1	`00`
`9F37`	Unpredictable Number	4	`9A1BC3D4`
`82`	Application Interchange Profile	2	`1D00`
`9F36`	Application Transaction Counter	2	`003D`
—	Card Verification Results	4	`03A00000`

Three of those values deserve a sentence each. The amounts are numeric format, twelve digits packed as six bytes, so `£4.20` is `000000000420` and not four hundred and twenty of anything — the currency's minor-unit exponent is not in the string, which is why the Transaction Currency Code has to be. The Terminal Verification Results byte 4 has bit 8 set, which is "transaction exceeds floor limit": that single bit is the reason the terminal asked for an ARQC rather than a TC. And the Application Interchange Profile byte 1 bit 3 is set, meaning "issuer authentication is supported" — the card will accept an EXTERNAL AUTHENTICATE command later.

Concatenated, that is thirty-seven bytes:

```
000000000420 000000000000 0826 0000008000 0826 260817 00
9A1BC3D4 1D00 003D 03A00000
```

The MAC algorithm

Book 2 Annex A1.2.1 specifies the computation for an 8-byte block cipher. It is ISO/IEC 9797-1 with the message padded according to ISO/IEC 7816-4, which the specification notes is equivalent to padding method 2 of ISO/IEC 9797-1: append a mandatory `'80'` byte, then the smallest number of `'00'` bytes that brings the length to a multiple of eight. The `'80'` is mandatory even when the message is already a multiple of eight. Our thirty-seven bytes become forty:

```
...9F36 003D  03A00000  80 0000
```

Split into five 8-byte blocks `X1..X5`. The session key is treated as a left half `KSL` and a right half `KSR`. Chain the blocks in CBC mode using **single DES with `KSL` only**, starting from an initial value of eight zero bytes:

```
Hi := ALG(KSL)[ Xi XOR Hi-1 ],  for i = 1..B,  H0 = '0000000000000000'
```

Then apply the output transformation. Book 2 permits either ISO/IEC 9797-1 Algorithm 1, which is simply `HB+1 := HB`, or Algorithm 3:

```
HB+1 := ALG(KSL)[ ALG-1(KSR)[ HB ] ]
```

Algorithm 3 — encrypt with the left half, decrypt with the right, encrypt with the left again, applied only to the final chaining value — is what every deployed scheme uses, and it is what Book 2 mandates for the Common Core Definitions Cryptogram Version `'5'`. It is the construction the banking world calls the Retail MAC or the ANSI X9.19 MAC. The Application Cryptogram is the s most significant bytes of `HB+1` with $s = 8$, so the whole of it.

For our example:

```
ARQC = 733084432BA24DAE
```

Eight bytes. Sixty-four bits. An attacker with no key guesses correctly about once in eighteen quintillion attempts, and the issuer will have declined and blocked the card long before that.

The AES option, Annex A1.2.2, is a different construction: ISO/IEC 9797-1:2011 Algorithm 5, which is CMAC, over 16-byte blocks, with the two CMAC subkeys `K1` and `K2` derived from the session key and the constant `'10000111'`. Padding is added only when the message is *not* already a multiple of sixteen, and the final block is masked with `K1` when no padding was added and `K2` when it was. Cryptogram Version `'6'` of the Common Core Definitions mandates AES CMAC with $s = 8$, and requires Option C master key derivation to go with it. Cryptogram Version `'5'` mandates Triple DES, Algorithm 3, and Option B.

The Application Transaction Counter

Tag `9F36`, two bytes, binary, maintained entirely by the card. The terminal cannot write it and the issuer cannot reset it. It increments once per transaction — in practice on the first GENERATE AC, or on GET PROCESSING OPTIONS in some contactless kernels — and it does not decrement.

Two bytes means the counter tops out at `FFFF`, 65,535. What a card does on reaching that ceiling is application-defined rather than specified by EMV; the counter is not permitted to be reused, so an application that reaches it has reached the end of its life. At a hundred transactions a month a card would need fifty-four years to get there, so it is not a practical constraint for a payment card, though it is a real one for some transit and stored-value applications.

The counter does three jobs at once, and they are worth separating.

It **diversifies the session key**. This is its cryptographic job and it is the reason it appears in `R`. Every transaction uses a different key, so a compromise of one transaction's key compromises exactly one transaction.

It **is inside the signed data**, as one of the ten recommended minimum elements. So it cannot be tampered with in flight, and its value binds the cryptogram to a specific position in the card's lifetime.

It **is visible to terminal risk management**. The terminal reads it directly with GET DATA — `80 CA 9F 36 00` — along with the **Last Online Application Transaction Counter Register**, tag `9F13`, which records the counter value at the last transaction that actually went online. The difference between the two is the number of transactions the card has completed offline since it last spoke to its issuer. The terminal compares that difference against the card's **Lower Consecutive Offline Limit**, tag `9F14`, and **Upper Consecutive Offline Limit**, tag `9F23`, and sets the corresponding Terminal Verification Results bits, which are what force the transaction online. Velocity checking is exactly this subtraction and nothing more.

At the issuer, the counter is the replay control. The honest statement of what it gives you is: an ARQC arriving with a counter value already seen for that card is either a duplicate or a forgery, and an ARQC arriving with a counter far above the last one seen indicates offline activity the issuer has not been told about. Both are signals. Neither is proof. The gap between "counter arrived out of sequence" and "fraud" is filled by the issuer's own risk rules, not by cryptography.

Issuer Application Data

Tag `9F10`, up to 32 bytes, described by EMV only as "proprietary application data for transmission to the issuer in an online transaction". EMV does not define its contents. The schemes do, and this is where an issuer host learns *how* to verify the cryptogram it has just been handed.

The Visa VIS layout, for Cryptogram Version Numbers 10 and 18, is:

Bytes	Field
1	Length Indicator
2	Derivation Key Index (DKI)
3	Cryptogram Version Number
4–7	Card Verification Results (4 bytes)
8+	Issuer Discretionary Data (optional)

A real one, taken from a Visa test transaction, is `9F10 07 06 00 0C 03A01000`: length indicator `'06'`, derivation key index `'00'`, Cryptogram Version Number `'0C'` — decimal 12, one of Visa's issuer-proprietary versions — and a four-byte Card Verification Results of `03A01000`, whose leading `'03'` is itself a length byte.

The Mastercard M/Chip 4 layout is different in both order and size:

Bytes	Field
1	Key Derivation Index
2	Cryptogram Version Number
3–8	Card Verification Results (6 bytes)
9–10	DAC / ICC Dynamic Number
11–18	Plaintext / Encrypted Counters (optional)

A real one: `9F10 12 02 12 A0000F240000 0000 00000000000000FF`.

A Common Core Definitions application always returns exactly 32 bytes, beginning with a length indicator and a **Common Core Identifier** whose high nibble is the IAD Format Code and whose low nibble is the Cryptogram Version. The remaining layout is given in Book 3's CCD annex.

Three things in that structure carry the whole weight of interoperability.

The **Derivation Key Index** tells the issuer host which of its master keys to load. Issuers rotate master keys across BIN ranges and over time; without the index the host would have to try them all.

The **Cryptogram Version Number** tells the host which algorithm to run — which master key derivation option, whether to derive a session key at all, which elements to concatenate and in what order, and which ARPC method the card expects. It is the single most load-bearing byte in the message.

The **Card Verification Results** are the card's own account of what happened. They report whether offline data authentication was performed and how it ended, whether offline PIN verification succeeded, whether the PIN Try Limit was exceeded, whether the offline counters were exceeded, how the *previous* transaction ended, and whether issuer authentication succeeded last time. The bit layouts are scheme-proprietary and are not published in EMV. Together with the Terminal Verification Results, the Card Verification Results are the two halves of the story: the terminal's version and the card's version of the same transaction, arriving in the same message, and sometimes disagreeing in ways that are extremely informative.

Verification at the issuer

The cryptogram and its companions travel to the issuer inside the ISO 8583 authorisation request, in data element 55, as a TLV list — `9F26` the cryptogram, `9F27` the Cryptogram Information Data, `9F36` the counter, `9F10` the Issuer Application Data, `9F37` the Unpredictable Number, `95` the Terminal Verification Results, `82` the Application Interchange Profile, and the rest. Volume III takes that message apart field by field.

The issuer host does six things, and all the cryptographic ones happen inside a hardware security module:

1. Parse `9F10`. Read the Derivation Key Index and the Cryptogram Version Number.
2. Select the issuer master key identified by the index, as a key token the host itself cannot read.
3. Derive MK-AC from the PAN and PAN Sequence Number using the option the Cryptogram Version requires.
4. Derive SK-AC from the Application Transaction Counter, unless the version says not to.
5. Reassemble the message: the same elements, in the same order, with the same lengths, that the card used. The host is reconstructing the terminal's work from the fields in the authorisation message, and any element it takes from the wrong place — the clearing amount rather than the authorised amount, its own date rather than the terminal's — produces a mismatch.
6. Compute the MAC and compare with the received `9F26`.

If they match, the card is genuine and the transaction data is intact. If they do not, the host has learned only that. Book 2 attaches a recommendation at this point that is easy to skip and important: an ARPC should only be returned if ARQC verification succeeded, and if an issuer insists on returning one anyway, it must not compute it from the ARQC it received from the network.

The ARPC and issuer authentication

The issuer's reply carries an **Authorisation Response Cryptogram**, and Book 2 section 8.2 defines two methods. Both use the same session key SK-AC that verified the ARQC.

Method 1 produces an 8-byte ARPC from the ARQC and the 2-byte Authorisation Response Code, tag `8A`, which is the ISO 8583 response code carried as two alphanumeric characters:

```
X    := ARC || '00' '00' '00' '00' '00' '00'
Y    := ARQC XOR X
ARPC := DES3(SK_AC)[Y]
```

For our transaction, with an ARC of `3030` — the two ASCII characters `0` and `0`, meaning approved:

```
ARQC = 733084432BA24DAE
X    = 3030000000000000
ARPC = 735C24FD87F1BF0E
```

The card verifies by deciphering the ARPC under SK-AC and XORing the result with its own ARQC; if what falls out is the ARC followed by six zero bytes, the issuer is authenticated.

Method 2 produces a 4-byte ARPC and carries far more information back to the card. Instead of a response code it uses a 4-byte binary **Card Status Update** and up to 8 bytes of Proprietary Authentication Data:

```
Y    := ARQC || CSU || Proprietary Authentication Data
ARPC := MAC(SK_AC)[Y], ISO/IEC 9797-1 Algorithm 3, s = 4
```

with the Issuer Authentication Data then formed as `ARPC || CSU || Proprietary Authentication Data`. Whether the proprietary data is included at all is governed by the "Proprietary Authentication Data Included" bit in the Card Status Update itself; if that bit is zero, the field is zero bytes long and, as Book 2 warns explicitly, anything sent in it is not protected by issuer authentication and the card should ignore it.

For our transaction, with a Card Status Update of `00800000`:

```
ARPC = 8F793CCC
Issuer Authentication Data = 8F793CCC 00800000
```

The Card Status Update is where the issuer reaches into the card, and its coding is published rather than proprietary: section C8 of Book 3's Common Core Definitions annex sets it out in Table CCD 11, for cryptogram versions `'5'` and `'6'`. The second byte carries the instructions: bit 8 "Issuer Approves Online Transaction", bit 7 "Card Block", bit 6 "Application Block", bit 5 "Update PIN Try Counter", bit 4 "Set Go Online on Next Transaction", bit 3 "CSU Created by Proxy for the Issuer", and bits 2 and 1 together the "Update Counters" instruction, which tells the card to leave its offline counters alone, set them to the upper offline limits, reset them to zero, or add this transaction to them. When bit 5 is set, the new PIN try count is taken from bits 4 to 1 of the first byte, whose top bit is the Proprietary Authentication Data Included flag; the third byte is reserved and the fourth is issuer-discretionary, defaulting to zero.

Note what bit 8 of byte 2 means. Under Method 2 the issuer's approval is *inside* the MAC. A card that verifies the ARPC and finds that bit clear will return an AAC and decline, no matter what the terminal was told. Under Method 1 the equivalent information is the ARC, also inside the calculation. Either way, the approval decision cannot be flipped in transit — which is precisely the class of attack that unauthenticated response codes leave open.

The Issuer Authentication Data reaches the card as tag `91`, 8 to 16 bytes, and there are two routes for it. If the Application Interchange Profile has byte 1 bit 3 set — "issuer authentication is supported" — the terminal sends an **EXTERNAL AUTHENTICATE** command:

Code	Value
CLA	`'00'`
INS	`'82'`
P1 P2	`'0000'`
Lc	8–16
Data	Issuer Authentication Data
Le	absent

It may be sent only once per transaction; a second attempt returns `6985`. Whatever the outcome, the terminal sets byte 1 bit 5 of the Transaction Status Information, "issuer authentication was performed". If issuer authentication fails, byte 5 bit 7 of the Terminal Verification Results is set, and the card's

CDOL2 will pull that byte into the second GENERATE AC, so the failure is itself covered by the second cryptogram.

If that Application Interchange Profile bit is *not* set, there is no EXTERNAL AUTHENTICATE. The issuer's data must instead be named in CDOL2 as tags `91` and `8A`, and reaches the card as part of the second GENERATE AC command. Mastercard M/Chip 4 takes the EXTERNAL AUTHENTICATE route; Visa VIS 1.4.x takes the second-GENERATE-AC route. An issuer host that assumes one of these for a portfolio containing both will find that half its script processing silently never executes.

The second GENERATE AC

Whatever happened online, the transaction is not finished until the terminal asks the card once more, using CDOL2, for a completion cryptogram. The terminal requests a TC if the issuer approved and it is willing to complete, or an AAC if not. The card performs its own second round of risk management, and again may only respond with something equal to or weaker than what was asked. The commonest surprise in this step is a card that returns an AAC after an approved authorisation: almost always the card failed to verify the ARPC, or verified it and found the approval bit clear.

Because the session key is derived from the Application Transaction Counter, and the counter does not increment between the two commands, both cryptograms in a transaction use the same SK-AC. They differ because the data differs — CDOL2 typically adds the Authorisation Response Code and the updated Card Verification Results.

A TC is not a courtesy. It is the evidence the acquirer presents in clearing that the transaction completed as authorised, and it is what an issuer asks for when a disputed transaction is being reconstructed. Schemes have been steadily tightening the requirement that the cryptogram appear in the clearing message and not only in the authorisation.

Combined data authentication

There is one arrangement in which the cryptogram and the public-key world do meet. Under **Combined DDA/Application Cryptogram Generation** — CDA — the card does not return the cryptogram in clear. It returns tag `9F4B`, Signed Dynamic Application Data, an RSA signature computed over data that includes the cryptogram itself.

For a Common Core Definitions application supporting CDA, the response to GENERATE AC for a TC or an ARQC contains exactly four objects: `9F27` the Cryptogram Information Data, `9F36` the counter, `9F4B` the signature, and `9F10` the 32-byte Issuer Application Data. Tag `9F26` is absent — the cryptogram is inside the signature. The leftmost bytes of the signed ICC Dynamic Data are the ICC Dynamic Number and its length, then the Cryptogram Information Data, then the 8-byte TC or ARQC, then a 20-byte Transaction Data Hash Code covering everything exchanged in the transaction. The signed data format byte is `'05'`, the recovered header is `'6A'`, the trailer is `'BC'`, and the padding pattern is `'BB'`. A card that responds with an AAC does not sign; it returns `9F26` in the clear, and the terminal declines.

CDA closes a specific hole: without it, an attacker sitting between card and terminal can let a genuine card generate a genuine ARQC and then alter the plaintext response, or substitute the card's answer to the second GENERATE AC. With it, altering anything invalidates a signature the terminal *can* check, because the terminal holds the certificate chain. This is the one place where the terminal's opinion of the cryptogram counts for something.

Failure modes

When a cryptogram does not verify, the issuer host learns nothing except that it did not verify. In practice the cause is almost always one of a short list, and it is worth having the list to hand.

Symptom	Usual cause
Every card in a BIN range fails	Wrong issuer master key, or wrong Derivation Key Index mapping
One card type fails, others succeed	Cryptogram Version Number not handled; wrong element list or wrong session key rule
Fails only for long PANs	Option A used where Option B is required, or vice versa
Fails only for non-zero PAN Sequence Numbers	Sequence number omitted from, or wrongly padded in, master key derivation
Fails only with cashback	Amount, Other omitted, or taken from the wrong field
Fails only in some currencies	Amount converted to major units, or the currency code taken from the account rather than the transaction
Fails intermittently at one estate	Unpredictable Number not reproduced byte-for-byte, or Data Object List padding applied the wrong way round
Verifies, but the card declines afterwards	ARPC method mismatch, or the approval bit in the Card Status Update not set
Verifies, but the counter is rejected	Offline transactions the host never saw, or an over-strict replay window

Every one of those is a data-assembly error, not a cryptographic one. The mathematics in this chapter has not failed in the field. The byte strings fed to it fail constantly.

The cryptogram in one paragraph

At personalisation, the issuer derives a card-unique key from its master key and the card's PAN. At transaction time, the card derives a use-unique session key from that card key and its own transaction counter, and computes an eight-byte Retail MAC over a fixed list of terminal-supplied and card-supplied facts — amount, currency, date, country, the terminal's random number, the card's counter and its own results. The type of cryptogram it returns is its verdict: TC to approve, ARQC to ask, AAC to refuse. The terminal cannot check any of it and merely forwards it. The issuer repeats the whole derivation from the PAN, the counter and the Cryptogram Version Number in the Issuer Application Data, and compares. If it approves, it computes a second cryptogram over its own answer so the card can be certain the answer is genuine, and only then will the card act on what the issuer says. Nothing in that chain is a secret the card gives away, which is why recording it is worthless, and why the eight bytes at the centre of it are the strongest single thing on the card.

15.98 Common wrong ideas

Wrong: The chip signs the transaction. **Right:** It computes a message authentication code with a symmetric key the issuer also holds, so anybody who can verify a cryptogram can also forge one, which is why that key must never leave tamper-resistant hardware at either end.

Wrong: The terminal checks the cryptogram before forwarding it. **Right:** The terminal has no key and never will, so when eight bytes come back from GENERATE AC it has no idea whether they are correct, random or fabricated, and it merely copies them into the authorisation message.

Wrong: A valid cryptogram proves the cardholder was present and entered a PIN. **Right:** It proves that a chip holding the correct key participated in a transaction with those exact parameters and nothing else; cardholder verification is reported separately, and an attacker who suppresses the PIN check can still obtain a perfectly valid cryptogram.

Wrong: There is a fixed list of data the card computes the cryptogram over. **Right:** Book 2 offers a recommended minimum set of ten elements, and what is actually included and in what order is fixed by the scheme-proprietary Cryptogram Version Number, which is by a wide margin the commonest cause of unexplained verification failures.

Wrong: The transaction counter makes replay impossible. **Right:** It makes replay detectable, and only if the issuer checks; hosts apply windows rather than hard ceilings because authorisations arrive out of order and offline transactions consume counter values that never reach the host at all.

Wrong: A cryptogram that fails to verify indicates a cryptographic problem. **Right:** In practice every failure is a data-assembly error — the wrong master key, an unhandled cryptogram version, an omitted cashback amount, a currency converted to major units, or Data Object List padding applied the wrong way round.

Wrong: A decline is just a decline, so it needs no cryptogram. **Right:** The refusal is sealed too, precisely so that nobody downstream can quietly turn a refusal into an approval.

Wrong: A terminal that requests an offline approval and receives one has been obeyed. **Right:** The terminal proposes and the card disposes, returning the requested type or a more restrictive one and never a less restrictive one, so a request to approve offline may come back as an online request or an outright refusal.

Wrong: One session key derivation rule covers an issuer's whole portfolio. **Right:** Visa Cryptogram Version 10 derives no session key at all and uses the card master key directly, Visa 18 and 22 and Mastercard 14 and 15 use the common session key method, and Mastercard 12 and 13 fold the terminal's unpredictable number into the derivation as well.

Wrong: Once the issuer approves, the transaction is finished. **Right:** The terminal must ask the card once more using the second data object list, and a card that failed to verify the response cryptogram, or verified it and found the approval bit clear in the Card Status Update, will refuse after an approved authorisation.

15.99 Chapter summary in 20 lines

1. Everything else on a card is data and all data can be copied, but the Application Cryptogram cannot, because it does not exist until the moment of the transaction and is never the same twice.
2. It is eight bytes long and it is a message authentication code computed with a key the card and the issuer share, not a signature anybody can verify.
3. Because the key is symmetric, the terminal cannot check the cryptogram at all and simply forwards eight bytes whose correctness it has no way of assessing.
4. The security of the whole scheme therefore rests on that key never leaving tamper-resistant hardware, since a single master key extracted from a badly managed host compromises every card in its range forever.
5. Three keys stand in a hierarchy: an issuer master key, a card-unique key derived from it with the account number and sequence number, and a session key derived from that with the card's own transaction counter.

6. Book 2 gives three master key derivation options and mandates none of them, and which one a card uses is announced by the scheme's Cryptogram Version Number rather than by EMV.
7. The session key changes with every transaction because the counter does, so compromising one transaction's key compromises exactly one transaction.
8. The data covered by the MAC is a fixed concatenation of eight facts supplied by the terminal, two supplied by the card, and in every real implementation the card's own Card Verification Results as well.
9. The terminal does not choose those elements: the card publishes two Card Risk Management Data Object Lists and the terminal fills them in the order given, under padding rules that run one way for numeric formats and the other for everything else.
10. The computation is CBC chaining under the left half of the session key followed by an encrypt-decrypt-encrypt on the final block, the construction the banking world calls the Retail MAC, with an AES option for newer cryptogram versions.
11. The type of cryptogram returned is the card's verdict: a Transaction Certificate approves offline, an Authorisation Request Cryptogram asks the issuer, and an Application Authentication Cryptogram refuses.
12. The terminal requests a type and the card may return that type or a more restrictive one, never a less restrictive one.
13. The transaction counter does three jobs at once: it diversifies the session key, it sits inside the signed data, and it is visible to terminal risk management as the gap from the Last Online ATC Register.
14. What the counter honestly gives an issuer is a signal and not a proof, since a repeated value means duplicate or forgery and a large jump means unreported offline activity, and the distance from signal to fraud is covered by risk rules.
15. The Issuer Application Data is where the host learns how to verify what it has been handed, carrying the Derivation Key Index, the Cryptogram Version Number and the Card Verification Results in scheme-specific layouts.
16. The Cryptogram Version Number is the single most load-bearing byte in the message, because it decides the derivation option, whether a session key exists, which elements are concatenated and which response method the card expects.
17. The issuer parses that field, loads the master key by index, derives the card key and session key, reassembles the terminal's byte string from the authorisation message and compares — and a mismatch tells it nothing except that there was a mismatch.
18. The issuer replies with its own cryptogram under one of two methods, so that its approval sits inside a MAC and cannot be flipped in transit, which is exactly the attack that unauthenticated response codes leave open.
19. Under combined data authentication the cryptogram is not returned in the clear at all but inside an RSA signature the terminal can check, the one place where the terminal's opinion of the cryptogram counts for anything.
20. When a cryptogram fails, the cause is almost always a data-assembly error rather than a cryptographic one, because the mathematics in this chapter has not failed in the field and the byte strings fed to it fail constantly.

Sources: EMV Integrated Circuit Card Specifications for Payment Systems, Book 2 (Security and Key Management) v4.3, sections 8.1–8.3 and Annexes A1.2, A1.3, A1.4 and the Common Core Definitions part; EMV Book 3 (Application Specification) v4.3, sections 5.4, 6.5.5 and 6.5.7 and the Data Elements Dictionary; ISO/IEC 9797-1 and ISO/IEC 7816-4; EMVCo tag references at emvlab.org; AWS Payment Cryptography and IBM z/OS ICSF documentation for scheme cryptogram version behaviour; CardContact OpenSCDP EMV traces. Worked figures were computed from the Book 2 algorithms using the stated test key and are reproducible.

Online and Offline

16.0 What this chapter gives you

1. You will be able to answer the question the previous chapter skipped — whether anybody actually phones the bank — and say what happens on the occasions when nobody can.
2. You will be able to explain why small purchases keep working during an outage while large ones stop, and why the cut-off is a number sitting in one particular till rather than a national figure.
3. You will be able to perform the three comparisons of Terminal Action Analysis by hand, in order, as a bitwise OR of two policies followed by a bitwise AND with the terminal's own observations.
4. You will be able to decode a five-byte Terminal Verification Results bit by bit, and say why its fifth byte cannot be decoded without knowing which kernel produced it.
5. You will be able to state the asymmetric defaults for absent action codes — all ones for the issuer's online and default lists, all zeros for every terminal action code — and explain why they point in opposite directions.
6. You will be able to explain why setting a terminal floor limit to zero does not straightforwardly mean "always go online".
7. You will be able to distinguish an offline approval from a store-and-forward deferral, and say why calling both "offline" is how merchants end up carrying risk they did not know they had.
8. You will be able to sort a network failure into its three shapes — the request never sent, sent with no answer, and an answer arriving too late — and say which of them makes a reversal mandatory whatever the local decision was.
9. You will be able to read a Terminal Type byte and explain why a supermarket till, a fuel pump, an aircraft seatback and a transit gate behave differently, including why the pump must commit before it knows the amount.
10. You will be able to run the four diagnostics that recover most of this ground when an estate is sending everything online, approving too much offline, failing cryptograms during outages, or producing duplicate debits afterwards.

Chapter 15 ended with the card producing eight bytes and the issuer producing eight bytes back. That description skipped a question, and the question is the one that decides whether your card works in a tunnel, on an aeroplane, at three in the morning when your bank's authorisation host is being patched, and at a petrol pump that has no idea how much fuel you are about to take.

The question is: does anybody phone the bank?

Most people assume the answer is always yes. It is not. A meaningful fraction of card transactions in the world are approved without any message ever reaching the issuer, and much of the *decision* is made locally even when a message does travel. The chip in your card is a small computer running its

own risk policy, and the terminal is running a different risk policy, written by a different organisation, with different money at stake. Both are expressed in the same format: a five-byte bitmap. Both are consulted on every transaction. Neither can see the other's reasoning.

This chapter is about those two policies, the bitmap they share, and what happens when the line to the bank is not there.

The plain version

Imagine a new employee on a supermarket till in Leeds. On her first day she is handed a laminated card from head office saying, in effect: *here are the situations in which you must ring us before taking the money, here are the situations in which you must refuse outright, and here is what to do if the phone is dead.*

That is the shop's policy. Now imagine that every customer also hands over a second laminated card, printed by their own bank, saying the same three things in the same three categories but with the bank's own opinions on them. The employee now has two policies to satisfy at once, and she does not get to choose between them.

She also has a tick sheet of forty boxes, each a specific observation: *the card would not prove it was genuine. The card has expired. The customer could not enter a PIN because the keypad is broken. The amount is above the limit the shop may approve on its own.* She fills it in as she goes, and most boxes stay empty on most transactions.

At the end she lays the tick sheet next to the two policy cards and asks three questions in order.

First: *do any of the ticked boxes appear on either "refuse outright" list?* If yes, she refuses, and no phone call happens.

Second, if not: *do any of the ticked boxes appear on either "ring us" list?* If yes, she rings.

Third, if she cannot ring — the line is dead, the shop has no line, the call timed out: *do any of the ticked boxes appear on either "if the phone is dead" list?* If yes, she refuses. If no, she approves it herself.

That is the whole mechanism. Two policies, one tick sheet, three questions.

A real transaction, with real numbers

A customer buys £42.50 of groceries. The till's limit for approving things on its own — head office calls it the floor limit — is £30.00.

The employee starts ticking. The amount is over £30, so the box marked *transaction exceeds floor limit* gets a tick. The card also tells her it has been used sixty-one times in its life and last spoke to its own bank on use number fifty-five. That is six uses ago, and the card's instructions say it prefers to check in after five, so a second box is ticked: *the card has gone too long without ringing home.* Everything else is fine, and thirty-eight of the forty boxes stay empty.

Question one: does either "refuse outright" list mention either of those? Neither does. She does not refuse.

Question two: does either "ring us" list mention them? The bank's card says yes to the floor limit. It happens not to mention the six-uses box at all — that bank does not consider it urgent — but one match is enough. She rings, the bank approves, and the transaction goes through in about half a second.

The same transaction with the phone line cut

Now run it again with the shop's broadband down. Questions one and two go the same way: no outright refusal, and the answer to "should I ring?" is still yes. But she cannot ring, so she falls through to question three and reads the third list.

The bank's third list *does* mention the floor limit, so the transaction is declined. Not because the customer has no money, not because the card is stolen, but because that customer's bank decided in advance that a £42.50 offline purchase it cannot see is a risk it does not want to take.

Had the basket been £8.40, no box would have been ticked, no phone call would have been needed, and the sale would have gone through with the line still dead. That is the most useful thing to understand about card payments during an outage: small purchases keep working and large ones stop, and the cut-off is not a national figure but a number sitting in that specific till for that specific scheme.

Two risk managers, and a spot check

One twist surprises people, and since it was the subject of the previous chapter it needs only a sentence here. The employee does not actually approve anything. She *proposes*. The card gets the final word, and it may be more cautious than she was but never less: she can say "I'd approve this offline" and the chip can answer "no, ring them", but she can never say "ring them" and have the chip answer "don't bother". There are two risk managers in the conversation, not one.

One last wrinkle explains something you may have noticed: occasionally a £2 coffee takes longer and clearly goes online for no visible reason. That is deliberate. A terminal may pick transactions at random and send them online even when nothing is wrong, because if small transactions *never* went online a thief would learn to keep every purchase under the floor limit and would never once be checked. The chance is weighted, so a purchase near the floor limit is far more likely to be selected than a tiny one.

If you take one thing to dinner, take this. Your card and the shop's till each carry a written policy about when the bank must be consulted, both policies are consulted every time, the strictest wins, and there is a third policy that only comes out when the line is down.

Where the plain version stops being true

The tick-sheet story gets you to the right answer for a supermarket till on an ordinary day. It hides four things, and each of them is where real implementations go wrong.

The two policies are merged, not consulted separately, and the merge only ever adds

The employee reads two lists and matches against either. That is nearly right, but the actual operation is a bitwise OR of the two policies followed by a bitwise AND with the tick sheet. Because the two are ORed before the comparison, *neither party can ever remove a condition the other has set*: an issuer cannot tell a terminal to stop going online for something the acquirer insists on, and an acquirer cannot relax a condition the issuer has demanded. Every configuration change either adds strictness or does nothing. There is no negotiation, no priority, no override.

That catches out people diagnosing live estates. When a terminal goes online it cannot tell you *why*: it has a five-byte mask, a five-byte tick sheet and a non-zero AND result, and it records neither which bit caused it nor which policy contributed that bit. To find out, you fetch the sheet and both policies out of the transaction record and do the arithmetic yourself.

An absent policy is also not a neutral policy. It is a defined value, and the value differs depending on which list is missing. An issuer that ships a card with no "ring us" list has not made it permissive; it

has made it maximally strict, because the fallback for the missing online list is all ones. An acquirer that leaves the terminal's lists unconfigured has done the opposite, because the fallback for a missing terminal action code is all zeros. The two defaults point in opposite directions, and that asymmetry causes a great many surprises in the field.

The terminal never approves, and the tick sheet is not a report

Calling the tick sheet a record of what went wrong is misleading twice over. First, not every ticked box is a problem. "Online PIN entered" is one of the forty boxes and is set on perfectly successful transactions; so is "transaction selected randomly for online processing". The sheet records *observations*, some neutral and some adverse, and it is the two policies — not the sheet — that decide which matter.

Second, the tick sheet is not a report at all. It is an *input*: its five bytes are handed to the chip inside the command that asks for a cryptogram, and they form part of the data the cryptogram is computed over. Two things follow. A terminal that sets a bit after sending the sheet has produced a transaction whose cryptogram covers a different sheet from the one in the authorisation message, and the issuer's verification will fail with no diagnostic. And the issuer is verifying the terminal's *claims* about its own behaviour: nothing prevents a compromised terminal from ticking boxes untruthfully, and the cryptogram merely proves the card saw the same lies the issuer did.

"Floor limit" is at least four different limits owned by four different parties

The plain version has one number, £30, sitting in the till. Real transactions are governed by several limits at once, and confusing them is the most common error in this whole area.

There is the terminal floor limit, loaded by the acquirer, specific to a particular scheme's application on that terminal, doing exactly what the story says. There are contactless reader limits, separate values with separate names governing whether a tap is allowed at all, whether it must go online, and whether it must carry a cardholder verification. There are the card's own offline limits, set by the issuer, which count *transactions* and in some schemes *cumulative amounts* rather than looking at a single purchase. And there are the scheme and regulatory limits — the figure a cardholder thinks of as "the contactless limit" — enforced somewhere else entirely.

The card's limits are the ones that usually bite, and they are worded more precisely than most people realise. The chip carries two consecutive-offline limits, and they are not "soft" and "hard" versions of the same thing. One states the issuer's preference for a terminal that *has* online capability; the other for a terminal that has *not*. Reading them as a graduated pair will make your velocity behaviour wrong at exactly the unattended terminals where it matters.

Setting the terminal floor limit to zero, incidentally, does not straightforwardly mean "always go online". It means the floor-limit observation is recorded every time; whether that produces an online request still depends on whether either policy lists it under "ring us". With a hand-configured pair that both leave the bit clear it does not, and you have a terminal with a zero floor limit approving everything offline.

Offline approval and store-and-forward are different things, and "the network went down" is at least three things

The plain version treats "cannot ring" as a single event with a single response. It is not.

Offline approval is a *decision*: nobody is going to ask the issuer, the card has sealed an approval in a cryptogram, and the money will be claimed later through clearing. Store-and-forward is a deferral: the terminal holds a cryptogram that says "please ask the bank", cannot ask now, and intends to ask later. The customer has already left with the goods, and if the answer comes back "no" three hours

later there is no customer to hand it to. The two put the loss in different places under different rules, and calling both "offline" is how merchants end up carrying risk they did not know they had.

The failure itself also has at least three shapes with three different correct responses. The request may never have left the terminal, in which case nothing has happened anywhere and the terminal falls through to its third policy. It may have left with no answer coming back, in which case the issuer may well have approved and debited a shadow balance, and the terminal is obliged to send a reversal *regardless of what it decides locally*, precisely because it does not know. Or an answer may arrive after the terminal has given up, leaving an approval in the issuer's records for a transaction the terminal has declined and the card has refused to certify. Treating those three as one case is the classic way to produce duplicate debits during an outage, and it is a failure of the acquirer's software, not of EMV.

The technical version

Two decisions, in this order

EMV places the online/offline determination in two processes with two owners, both running on every contact transaction that reaches the cryptogram stage. **Terminal Action Analysis** is EMV Integrated Circuit Card Specifications for Payment Systems, Book 3 (Application Specification), section 10.7: the terminal's decision, producing a *request* for one of three cryptogram types via the reference control parameter of the GENERATE APPLICATION CRYPTOGRAM command. **Card Action Analysis** is Book 3 section 10.8, the card's decision; the chip may return the requested type or a more restrictive one, never a less restrictive one, and reports what it did in the Cryptogram Information Data, tag 9F27. Book 4, Table 3, sets out what the terminal must do with each answer. Everything else in this chapter feeds those two decisions.

Terminal Verification Results

The tick sheet is the Terminal Verification Results, tag 95, five bytes, binary, generated by the terminal and defined in Book 3, Annex C5. Bits run b8 (most significant) to b1 within each byte; reserved-for-future-use bits are coded as zero and neither party checks them.

Byte	Bits, b8 downwards
1	b8 offline data authentication was not performed; b7 SDA failed; b6 ICC data missing; b5 card appears on terminal exception file; b4 DDA failed; b3 CDA failed; b2 SDA selected
2	b8 ICC and terminal have different application versions; b7 expired application; b6 application not yet effective; b5 requested service not allowed for card product; b4 new card
3	b8 cardholder verification was not successful; b7 unrecognised CVM; b6 PIN Try Limit exceeded; b5 PIN entry required and PIN pad not present or not working; b4 PIN entry required, PIN pad present, but PIN was not entered; b3 online PIN entered
4	b8 transaction exceeds floor limit; b7 lower consecutive offline limit exceeded; b6 upper consecutive offline limit exceeded; b5 transaction selected randomly for online processing; b4 merchant forced transaction online

Byte	Bits, b8 downwards
5	b8 default TDOL used; b7 issuer authentication failed; b6 script processing failed before final GENERATE AC; b5 script processing failed after final GENERATE AC

Byte 5 is where contact and contactless diverge. In EMV Contactless Specifications for Payment Systems, Book C-2 (Kernel 2), byte 5 additionally carries the relay resistance outcomes: bits for "relay resistance threshold exceeded" and "relay resistance time limits exceeded", and a two-bit field in b2 and b1 encoding whether the relay resistance protocol was not supported, not performed or performed. A decoder handling both interfaces cannot decode byte 5 without knowing which kernel produced it. Note too that byte 1 b2, "SDA selected", is an observation rather than a failure: it exists so an issuer can express a policy about static data authentication being used where dynamic authentication was available.

Action codes: three lists, two owners, five bytes each

Every one of the six action codes has exactly the same five-byte layout as the Terminal Verification Results, bit for bit.

The **Issuer Action Codes** live on the card and are read during the READ RECORD stage: `9F0D` for Default, `9F0E` for Denial, `9F0F` for Online. Book 3 is precise about scope. Denial specifies the issuer's conditions that cause denial *without attempt to go online*; Online, the conditions that cause a transaction to be transmitted online; Default, the conditions that cause rejection *if it might have been approved online but the terminal is unable to process the transaction online*.

The **Terminal Action Codes** are acquirer-configured and terminal-resident. On the contact interface they have no EMV tag because they never cross the card interface; Book 4, Table 7 lists all three as application-dependent terminal data elements, required if non-zero values are used. Contactless kernels carry them as configuration objects — Kernel 2 holds Terminal Action Code – Denial in tag `DF8121`.

The defaults when a code is absent must be memorised, because they are asymmetric.

Action code	Value used if absent
Issuer Action Code – Denial	`00 00 00 00 00`
Issuer Action Code – Online	`FF FF FF FF FF`
Issuer Action Code – Default	`FF FF FF FF FF`
Any Terminal Action Code	`00 00 00 00 00`

Book C-5 (Kernel 5) states the three issuer defaults explicitly for cards that do not carry the codes, matching Book 3, and Book 4 confirms that in the absence of a Terminal Action Code the default is all bits set to 0. Book 4 attaches a strong recommendation to that default: the bits for "offline data authentication was not performed", "SDA failed", "DDA failed" and "CDA failed" should be set to 1 in both Terminal Action Code – Default and Terminal Action Code – Online. Those are byte 1 b8, b7, b4 and b3, so the minimum implied is `CC 00 00 00 00`. An acquirer shipping all-zero terminal action codes has followed the letter of the specification and ignored its advice.

The three comparisons

Book 3 section 10.7 defines the decision as three bitwise tests in a fixed order. In each, the two five-byte action codes are ORed and the result ANDed with the Terminal Verification Results; a non-zero result means the condition applies.

Denial first: if any bit is set in `TVR AND (TAC-Denial OR IAC-Denial)`, the terminal declines offline and requests an AAC, and no online message is attempted. Online second, and only if the terminal has online capability: if any bit is set in `TVR AND (TAC-Online OR IAC-Online)` the terminal requests an ARQC, otherwise a TC. Default third, used when the terminal is offline-only so the online test never ran, or is online-capable but unable to go online: if any bit is set in `TVR AND (TAC-Default OR IAC-Default)` the terminal requests an AAC, otherwise a TC.

Book 4 section 6.3.6 confirms the bookkeeping. On an offline approval the terminal sets the Authorisation Response Code to "Offline approved"; on an offline decline, to "Offline declined"; and when it transmits online it sets no value at all, leaving the field for the response message.

A complete worked example

Take a supermarket till in Leeds on 17 August 2026. Terminal Type `9F35` is `22`, attended, merchant-controlled, offline with online capability. The floor limit is £30.00, so `9F1B` is `00 00 0B B8`; the basket is £42.50, so Amount, Authorised, `9F02`, is `00 00 00 00 42 50`; Transaction Currency Code `5F2A` and Terminal Country Code `9F1A` are both `0826`; Transaction Date `9A` is `26 08 17` and Transaction Type `9C` is `00`. The card reports an Application Transaction Counter, `9F36`, of `00 3D` — sixty-one — a Last Online ATC Register, `9F13`, of `00 37`, and a Lower Consecutive Offline Limit, `9F14`, of `05`.

Terminal risk management runs. £42.50 exceeds £30.00, so TVR byte 4 b8 is set. The counter is six ahead of the last online register, which exceeds five, so byte 4 b7 is set. Data authentication succeeded, the application is in date, the PIN verified offline. The resulting TVR is:

```
95 05 00 00 00 C0 00
```

The card presents these issuer action codes:

```
9F0E 05 00 10 18 00 00      IAC - Denial
9F0F 05 FC 68 BC 98 00      IAC - Online
9F0D 05 FC 40 AC 80 00      IAC - Default
```

The acquirer has loaded `CC 00 00 00 00` as both Terminal Action Code – Online and Terminal Action Code – Default, and all zeros as Terminal Action Code – Denial.

Denial test: `(00 00 00 00 00) OR (00 10 18 00 00)` is `00 10 18 00 00`, ANDed with the TVR gives all zeros. No denial.

Online test: `(CC 00 00 00 00) OR (FC 68 BC 98 00)` is `FC 68 BC 98 00`, ANDed with `00 00 00 C0 00` gives `00 00 00 80 00` — non-zero, from the floor-limit bit alone. Note what did *not* contribute: byte 4 of this issuer's online code is `98`, that is b8, b5 and b4, and does not include b7. This issuer does not regard the lower consecutive offline limit as a reason to go online. The terminal requests an ARQC.

Now suppose the line is dead, and the terminal falls through to the default test: `(CC 00 00 00 00) OR (FC 40 AC 80 00)` is `FC 40 AC 80 00`, ANDed with `00 00 00 C0 00` gives `00 00 00 80 00`. The terminal requests an AAC and sets the Authorisation Response Code to `Z3`. Same basket, same card, same till, different answer, and the only thing that changed is which of the three lists was consulted.

Terminal risk management: what sets those bits

None of this runs unless the card asks for it. Application Interchange Profile, tag `82`, byte 1 b4 is "terminal risk management is to be performed". A terminal must also perform it whenever its own floor limit checking demands, but the AIP bit is the card's request.

Floor limit checking. Terminal Floor Limit, tag `9F1B`, is four bytes binary, terminal-sourced, held per application identifier — one physical terminal can carry different floor limits for different schemes. Book 4, Table 7 makes it required on an offline terminal or an offline terminal with online capability. The terminal may keep a log of recent transactions keyed on Application PAN and Application PAN Sequence Number so a customer cannot defeat the limit by splitting a purchase; where a log is kept, the current amount is summed with the logged amounts before comparison.

Random transaction selection. Required on offline terminals with online capability, using three acquirer-configured values: Target Percentage to be Used for Random Selection, Threshold Value for Biased Random Selection, and Maximum Target Percentage to be Used for Biased Random Selection. Below the threshold the selection probability is the target percentage; between the threshold and the floor limit it is interpolated linearly:

```
interpolation      = (amount - threshold) / (floor limit - threshold)
transaction target = ((max target - target) x interpolation) + target
```

The terminal generates a random number in the range 1 to 99 and selects the transaction for online processing if that number is less than or equal to the transaction target percent, setting TVR byte 4 b5. With a threshold of £10.00, a target of 10 per cent, a maximum target of 40 per cent and a floor limit of £30.00, a £22.50 purchase interpolates at 0.625 and yields a target of 28 per cent. That is why a purchase at £29.99 is very nearly always selected.

Velocity checking. The card supplies Lower Consecutive Offline Limit, tag `9F14`, and Upper Consecutive Offline Limit, tag `9F23`, one byte each. These are not soft and hard versions of one control. Book 3 is explicit: the lower limit is the issuer's preference for the maximum number of consecutive offline transactions *in a terminal with online capability*, the upper limit the same preference *in a terminal without online capability*. If either is absent from the card, velocity checking is skipped entirely. The terminal retrieves Application Transaction Counter `9F36` and Last Online ATC Register `9F13` with GET DATA — class `80`, instruction `CA`, tag in P1-P2 — and the difference between them is the number of transactions since the card last went online. Exceeding the lower limit sets TVR byte 4 b7; exceeding the upper limit sets byte 4 b6 as well.

Exception file. If the Application PAN matches an entry in the terminal's exception file, TVR byte 1 b5 is set. Exception files are largely historical in online-capable estates and largely useless in offline-only ones, because they cannot be updated.

The terminal also maintains Transaction Status Information, tag `9B`, two bytes, recording which *processes ran* rather than what they found: byte 1 b8 offline data authentication performed, b7 cardholder verification performed, b6 card risk management performed, b5 issuer authentication performed, b4 terminal risk management performed, b3 script processing performed, byte 2 reserved. The TSI says a step happened; the TVR says what it observed. A terminal reporting "issuer authentication failed" in the TVR while leaving "issuer authentication was performed" clear in the TSI has a bug.

Card Verification Results

The card runs its own risk management and keeps its own tick sheet: the Card Verification Results. It is not a standalone tag. It lives inside Issuer Application Data, tag `9F10`, binary, zero to thirty-two bytes, returned by the card in the GENERATE AC response.

There is no single layout. The Issuer Application Data is proprietary to the payment system and its internal structure is selected by the Cryptogram Version Number carried inside it. EMV defines a Card Verification Results only for Common Core Definitions applications, in Book 3, Annex C; everything else is published by the schemes.

Two layouts cover most of the estate. For Visa VSDC with Cryptogram Version Numbers 10 and 18, the Issuer Application Data is a length indicator, a one-byte Derivation Key Index, a one-byte Cryptogram Version Number, a four-byte Card Verification Results and optional Issuer Discretionary Data, so `06 00 0C 03 A0 10 00` carries a Card Verification Results of `03 A0 10 00`. For Mastercard M/Chip, an eighteen-byte Issuer Application Data is a one-byte Key Derivation Index, a one-byte Cryptogram Version Number, a *six*-byte Card Verification Results, a two-byte DAC or ICC Dynamic Number and eight bytes of counters.

What it reports, across all the schemes, is the card's own account of the transaction and its recent history: the cryptogram type returned at each GENERATE AC; whether offline PIN verification was performed and succeeded; whether the PIN try limit is exceeded; whether the previous transaction went online and completed; whether issuer authentication was attempted and failed; whether the card's velocity counters are exceeded; whether an issuer script was received and failed; and counts of consecutive offline transactions and, in some schemes, cumulative offline amounts. Bit positions differ between schemes and between cryptogram versions within a scheme, so there is no generic Card Verification Results decoder; any tool claiming to be one is guessing at the version. And for most cryptogram versions the Issuer Application Data is included in the data over which the Application Cryptogram is computed, so the Card Verification Results is sealed. That asymmetry matters: the terminal's tick sheet is a claim the terminal makes about itself, whereas the card's is signed.

The GENERATE AC exchange

The first GENERATE AC is formatted according to Card Risk Management Data Object List 1, tag `8C`, supplied by the card, which for a typical application includes `9F02` Amount Authorised, `95` Terminal Verification Results, `5F2A` Transaction Currency Code, `9A` Transaction Date, `9C` Transaction Type and `9F37` Unpredictable Number. The second is formatted from Card Risk Management Data Object List 2, tag `8D`, whose critical addition is Authorisation Response Code, tag `8A`, two bytes. The terminal must never modify an Authorisation Response Code obtained from a response message; Book 4, Annex A6, is explicit. But when the transaction is *not* authorised online the terminal generates the code itself, and Book 4, Table 35, defines exactly four values:

Authorisation Response Code	Value
Offline approved	`Y1`
Offline declined	`Z1`
Unable to go online, offline approved	`Y3`
Unable to go online, offline declined	`Z3`

The distinction between `Y1` and `Y3`, and between `Z1` and `Z3`, is precisely the distinction between the second comparison and the third: `Y1` and `Z1` mean the terminal decided offline because its policies told it to, `Y3` and `Z3` mean it decided offline because it had no choice. Issuers price offline risk on that difference, and a terminal reporting `Z1` when it meant `Z3` is misrepresenting an outage as a policy decline.

Where an online response arrives containing Issuer Authentication Data, tag `91`, the terminal delivers it to the card — through EXTERNAL AUTHENTICATE or inside the second GENERATE AC, depend-

ing on the application — and sets TSI byte 1 b5; if the card rejects it, TVR byte 5 b7 is set. Book 4 section 12.2.2 covers the case that trips people up: when the response contains no Issuer Authentication Data at all, the terminal must *not* execute EXTERNAL AUTHENTICATE and must set the TSI bit to 0, continuing on the Authorisation Response Code alone. That is a downgraded authorisation, not a failure, and the TVR bit must stay clear.

Why a fuel pump, an aircraft seatback and a supermarket till behave differently

EMV encodes the answer in one byte. Terminal Type, tag `9F35`, is two digits: operational control and environment. Book 4, Annex A1, gives the full table.

	Financial institution	Merchant	Cardholder
Attended, online only	11	21	—
Attended, offline with online capability	12	22	—
Attended, offline only	13	23	—
Unattended, online only	14	24	34
Unattended, offline with online capability	15	25	35
Unattended, offline only	16	26	36

There are no attended cardholder-controlled types. Terminal types `14`, `15` and `16` with cash disbursement capability — Additional Terminal Capabilities, tag `9F40`, byte 1, cash bit set to 1 — are ATMs; nothing else is, and any logic that decides "is this an ATM" from the merchant category code rather than from those two data elements is doing it wrong.

The supermarket till is `22`: attended, merchant-controlled, offline with online capability, with a modest floor limit, a working keypad, a person behind it and a network it expects to be up. All three comparisons are available to it, and on an ordinary day it goes online above the floor limit and approves the rest itself.

The fuel pump is `25` — unattended, merchant-controlled, offline with online capability — with merchant category code `5542`, automated fuel dispensers, as distinct from `5541`, service stations. Its problem is not connectivity. It is that *the amount is not known when the decision has to be made*: the pump must be authorised to dispense before anyone knows how much fuel will come out.

That breaks the arithmetic. The Amount, Authorised, tag `9F02`, that goes into the cryptogram is an *estimate*, and the cryptogram covers the estimate and only the estimate. The final amount, which is what actually clears, was never seen by the chip. Every scheme therefore wraps automated fuel dispensers in its own rules rather than leaving it to EMV: the Visa Core Rules and Visa Product and Service Rules carry a section on Automated Fuel Dispenser Transactions with tables specifying status check authorisations, maximum amount initial authorisations and maximum allowed amounts, while Mastercard classifies the same devices through the cardholder-activated terminal levels in DE 61, subfield 10 of the authorisation message.

Everything else follows from that estimate: a hold for an amount unrelated to the fuel taken, a reconciliation between hold and clearing, and a figure on the cardholder's statement that never appeared on the pump. It is not a bug. It is the only way to run a device that must commit before it knows.

The aircraft seatback is `26` or `36` — unattended, offline only. "Offline only" is a specification term with a precise meaning, given in Book 4, Table 1: the transaction can only be completed offline by the terminal. On such a terminal the second comparison never happens; Terminal Action Code – Online and Issuer Action Code – Online are read and never used. The decision reduces to the denial test and

then the default test, which is exactly why Issuer Action Code – Default falls back to all ones when absent, and why a card carrying no Issuer Action Code – Default is declined by an aircraft for the smallest observation.

Mastercard's terminology for this environment is CAT level 4, in-flight commerce, alongside level 1 for automated dispensing machines, level 2 for self-service terminals, level 3 for limited amount terminals and level 9 for mobile point-of-sale acceptance devices; the Mastercard Switch Rules manual sets out the requirements for each. Two further differences bite on an aircraft: the exception file can be no fresher than the last time someone updated it on the ground, and there is no attendant to compare a signature, so cardholder verification collapses to whatever the card's CVM List permits an unattended device to do — in practice No CVM Required or offline PIN.

Transit gates are a fourth case, frequently confused with the third. A gate is unattended and often offline at the moment of the tap, but it is usually not an offline-only terminal: it is an online-capable terminal deliberately deferring the online request under a scheme-sanctioned transit model, which is not an EMV offline approval. The regulatory position differs too. Article 12 of Commission Delegated Regulation (EU) 2018/389, the regulatory technical standards on strong customer authentication, exempts unattended terminals *for the purpose of paying a transport fare or a parking fee* with no monetary ceiling, whereas an ordinary contactless purchase falls under Article 11 with its ceiling of EUR 50 for a single transaction, EUR 150 cumulative since the last strong authentication, and no more than five consecutive transactions. A seatback screen selling a gin and tonic is not paying a transport fare, and does not get Article 12. The United Kingdom's own figures were £100 single and £300 cumulative until the Financial Conduct Authority removed the mandatory caps on 19 March 2026, leaving issuers with adequate fraud controls to set their own.

Offline approval and its risk

The risk in offline approval is not that the cryptogram is weak; it is exactly as strong as an online one. The risk is that *nobody with a balance in front of them has looked at it*. Four things are unknown at that moment: whether the account has funds; whether the card has been reported lost or stolen since the exception file was last updated; whether the cryptogram is even valid, because as Chapter 15 established the terminal has no key and cannot check; and whether this is the fifteenth offline transaction on that card in an hour at fifteen terminals, none of which can see the others.

The controls against all four are the card's own. Offline data authentication establishes that the chip is genuine before any offline approval is contemplated, which is why Book 4 recommends so firmly that the authentication-failure bits be set in Terminal Action Code – Default and – Online, and why offline approval following a failed authentication is the single most dangerous configuration in EMV. Beyond that, the consecutive offline limits force a card back online after a defined number of unseen transactions, and scheme-proprietary cumulative amount limits do the same on value. The gap between the Application Transaction Counter and the Last Online ATC Register is literally a measure of the issuer's current exposure on that card.

Liability follows configuration, and configuration is auditable after the fact: the Terminal Verification Results, Terminal Type, Terminal Capabilities and cryptogram all travel in the clearing record. A terminal that approved offline above its scheme-permitted floor limit, or after data authentication failed, has departed from the rules visibly, and scheme chargeback rules attach consequences to exactly that.

Store-and-forward

Store-and-forward is what an online-capable terminal does when it cannot reach the acquirer and its policies still permit completion: it stores the full transaction, cryptogram included, and transmits it when connectivity returns. Some acquirers call it deferred authorisation. It is not offline approval, because the answer is still outstanding.

The cryptogram in a stored transaction is an ARQC nobody has verified, or a TC generated against an Authorisation Response Code of `Y3`, and either way the issuer receives it late. Hosts enforcing a strict Application Transaction Counter ceiling will reject it, because by then the card has almost certainly been used elsewhere and the host has seen higher counter values. This is one of the concrete reasons issuer hosts apply a counter *window* rather than a hard ceiling.

The obligation to send a reversal is not discretionary. Book 4 section 12.2.3 covers the response that is not received, is received too late, or is malformed: after any permitted repeats the terminal processes the transaction as unable to go online, runs the default comparison, and issues the second GENERATE AC. Then, where online data capture is performed by the acquirer, the terminal *shall send a reversal message regardless of the final decision on the transaction*, so that if the host did receive the request and send a response, that transaction is cancelled. If the card's final answer was a TC, the terminal must also create a financial record for the acquirer.

Third, storing a transaction before authorisation means holding cardholder data before authorisation, the one window in which PCI DSS permits sensitive authentication data to exist at all, and only under specific protection; such an implementation must be designed against the standard's pre-authorisation storage requirements, not retro-fitted to them. Fourth, presentment windows are set by scheme rules and measured in days, and a queue surviving a long outage may hold transactions that are no longer presentable.

The messages are ordinary ISO 8583 — authorisation 0100 and 0110, financial 0200 and 0210, advice 0220, reversal 0400 with 0410 or advice-form 0420 with 0430. Whether a stored transaction goes as a financial request or as an advice depends on the acquirer's specification, and determines whether the issuer is asked a question or told a fact.

When the network dies mid-transaction

Sort the failures by where the transaction was when the line went.

Before the request was sent. Book 4 section 12.2.1: the terminal compares the Terminal Verification Results with Terminal Action Code – Default and Issuer Action Code – Default, decides, and issues the second GENERATE AC with an Authorisation Response Code of `Y3` or `Z3`. Nothing has happened at the issuer, no reversal is required, and the card's answer in the Cryptogram Information Data is final.

After the request was sent, with no response, or with a response that arrives too late. Section 12.2.3. The terminal may repeat the request if the acquirer requires it — repeat requirements are explicitly outside the scope of EMV — and then must treat the transaction as unable to go online. Same default comparison, same `Y3` or `Z3`. But the reversal is now mandatory under online data capture, because the issuer may have approved and debited. The late response is the case most often implemented wrongly, because the response *is there* and a naive implementation will process it after the card has already been given a `Z3` and returned an AAC. The transaction is then declined at the card and approved at the issuer, and only the reversal reconciles them.

Response arrives without Issuer Authentication Data. Section 12.2.2, and not a network failure at all. The terminal does not execute EXTERNAL AUTHENTICATE, sets the TSI issuer-authentication bit to

0, leaves the TVR issuer-authentication-failed bit clear, and proceeds on the Authorisation Response Code.

The card leaves mid-transaction. Common on a contactless reader: the cardholder lifts the device between command and response. Kernel 2 long addressed it with a Torn Transaction Log, governed by Max Lifetime of Torn Transaction Log Record, tag `DF811C`, an integer in seconds, and Max Number of Torn Transaction Log Records, tag `DF811D`, together with the RECOVER AC command, which retrieves from the card the last transaction it completed so that a returning card can be matched against a stored torn record and the transaction finished rather than restarted — which would burn a second Application Transaction Counter value. That machinery is now historical. EMVCo's Specification Bulletin number 261 of October 2021, applying to Book C-2 version 2.10 and effective from the first of January 2022, states that torn transaction recovery is no longer supported in Kernel 2, retires the Torn Transaction Log, deletes the RECOVER AC section on the grounds that the command is no longer used, and strikes the torn data objects, tags `DF811C` and `DF811D` among them, from the data dictionary. The revision log of Book C-2 version 2.11, published in June 2023, records that bulletin as incorporated, and the current Kernel C-2 conformance statement offers implementers only mag-stripe mode and data exchange as options. A reader still running an older kernel will meet the mechanism; a newly approved one will not.

The issuer is down but the network is up. Not a terminal problem at all, and the terminal will never know. The scheme answers on the issuer's behalf — Visa calls it Stand-In Processing, Mastercard calls it Stand-In Processing Service — applying issuer-supplied parameters and reconciling afterwards. To the terminal this is an ordinary online authorisation, typically arriving without Issuer Authentication Data, which returns us to the downgraded authorisation case and a TSI with the issuer authentication bit at zero. If a large share of an estate's transactions show issuer authentication not performed, stand-in is the first place to look, not a terminal fault.

What to check when it goes wrong

Every terminal runs the denial test. Only online-capable terminals run the online test; offline-only terminals go straight from denial to default. And every terminal reaches the default test eventually, which is the point most often missed: an online-only terminal is not exempt from it, it simply expects never to get there. When it does, it needs a sensible Terminal Action Code – Default loaded, and online-only estates are precisely the ones that never bothered. With the fallback of all zeros and a card whose Issuer Action Code – Default is present and permissive, such a terminal will approve offline in exactly the circumstances its operator assumed impossible.

Four diagnostics recover most of the ground in this chapter. If an estate is sending everything online, pull the Terminal Verification Results and both online action codes from a sample of transactions and do the AND yourself; almost always the card carries no Issuer Action Code – Online, so the all-ones default applies, or data authentication is silently failing and byte 1 lights up every time. If an estate is approving too much offline, check whether the Terminal Action Codes were ever loaded, because all-zero terminal action codes plus a permissive card will pass every certification test and lose money in production. If cryptograms fail at the issuer only during outages, check whether the terminal is building the GENERATE AC data from a stale copy of the Terminal Verification Results after it has finished setting bits. And if duplicate debits appear after an outage, check the reversal path first: when online data capture is performed and the terminal was unable to obtain a valid response, a reversal is sent regardless of what the card finally decided.

The deeper point behind all four is the one this chapter opened with. There is no single authority in a card transaction: there is a terminal with an acquirer's policy, a card with an issuer's policy, a

merged bitmap neither party can see in full, and a network that is sometimes there. Everything that looks arbitrary about card acceptance — why the pump held £100, why the aeroplane refused a perfectly good card, why the corner shop kept working when the supermarket stopped — falls out of that arrangement, one five-byte comparison at a time.

16.98 Common wrong ideas

Wrong: The terminal reads the issuer's list and the acquirer's list and matches against either. **Right:** The two five-byte codes are ORed together and the result ANDed with the Terminal Verification Results, so neither party can ever remove a condition the other has set and every configuration change either adds strictness or does nothing.

Wrong: A card carrying no Issuer Action Code – Online is permissive. **Right:** The fallback for a missing issuer online or default list is all ones, making such a card maximally strict, whereas an unconfigured Terminal Action Code falls back to all zeros — the two defaults point in opposite directions.

Wrong: The Terminal Verification Results is a report of what went wrong. **Right:** It is an input, handed to the chip inside the command that asks for a cryptogram and covered by that cryptogram, and several of its bits — "online PIN entered", "transaction selected randomly for online processing" — are set on perfectly successful transactions.

Wrong: A terminal can tell you why it went online. **Right:** It holds a five-byte mask, a five-byte sheet of observations and a non-zero AND result, and records neither which bit caused it nor which policy contributed that bit, so you must fetch all three from the transaction record and do the arithmetic yourself.

Wrong: The floor limit is one number. **Right:** There is the acquirer's per-application terminal floor limit, a set of contactless reader limits, the card's own offline limits set by the issuer, and the scheme and regulatory limits, and confusing them is the most common error in this whole area.

Wrong: The lower and upper consecutive offline limits are soft and hard versions of the same control. **Right:** The lower is the issuer's preference for a terminal that has online capability and the upper for one that has not, and reading them as a graduated pair makes velocity behaviour wrong at exactly the unattended terminals where it matters.

Wrong: Setting the terminal floor limit to zero means every transaction goes online. **Right:** It means only that the floor-limit observation is recorded every time, and whether that produces an online request still depends on whether either policy lists that bit, so a hand-configured pair that both leave it clear gives a zero floor limit approving everything offline.

Wrong: Store-and-forward is a kind of offline approval. **Right:** Offline approval is a decision nobody will revisit, while store-and-forward holds a cryptogram that still says "please ask the bank" and defers the question, and the two put the loss in different places under different rules.

Wrong: When no response arrives, the terminal simply decides locally and moves on. **Right:** The issuer may well have approved and debited a shadow balance, so under online data capture a reversal must be sent regardless of what the terminal and the card finally decided, and skipping it is the classic way to produce duplicate debits during an outage.

Wrong: An online-only terminal never needs a Terminal Action Code – Default. **Right:** Every terminal reaches the default test eventually, and an online-only estate that never loaded one falls back to all zeros and, against a permissive card, approves offline in exactly the circumstances its operator assumed impossible.

16.99 Chapter summary in 20 lines

1. A meaningful fraction of card transactions in the world are approved without any message ever reaching the issuer, and much of the decision is made locally even when a message does travel.
2. Two organisations write risk policies for the same transaction: the acquirer's, loaded into the terminal, and the issuer's, carried on the card.
3. Each policy is three five-byte lists — denial, online and default — laid out bit for bit like the terminal's own record of what it observed.
4. That record is the Terminal Verification Results, five bytes of specific findings, most of which stay clear on most transactions and some of which are neutral observations rather than faults.
5. Terminal Action Analysis runs three comparisons in a fixed order, each ORing the two matching action codes and ANDing the result with those observations.
6. Denial comes first and produces an offline decline with no online attempt; online comes second, and only where the terminal has online capability; default comes third, for an offline-only terminal or one that cannot reach the network.
7. Because the two policies are ORed before the comparison, neither party can relax what the other has demanded, and there is no negotiation, no priority and no override.
8. The defaults for absent codes are asymmetric — a missing issuer online or default list counts as all ones, a missing terminal action code as all zeros — and Book 4 separately recommends that the four data-authentication failure bits be set in the terminal's online and default codes.
9. The observations are not a report but an input, since their five bytes go into the command that asks for a cryptogram, so a terminal that sets a bit after sending them produces a cryptogram the issuer cannot verify.
10. They are also a claim the terminal makes about its own behaviour, whereas the card's account of itself, the Card Verification Results inside the Issuer Application Data, is sealed by the cryptogram.
11. Whatever the terminal requests, Card Action Analysis lets the chip return the same type or a more restrictive one and never a less restrictive one, so there are two risk managers and not one.
12. Terminal risk management sets the decisive bits: floor limit checking, random selection biased towards amounts near the limit, velocity checking on the gap between the transaction counter and the Last Online ATC Register, and the exception file.
13. In the worked example a £42.50 basket at a till with a £30 floor limit goes online on the floor-limit bit alone, and the identical basket with the line dead is declined by the default comparison — same card, same till, a different list consulted.
14. "Floor limit" is really several limits owned by different parties, and the card's two consecutive-offline limits are preferences for terminals that do and do not have online capability rather than a soft and hard pair.
15. Terminal Type, one byte of operational control and environment, is why a supermarket till, a fuel pump, an aircraft seatback and a transit gate behave differently.
16. A fuel pump's difficulty is not connectivity but that the amount is unknown when the decision must be made, so the cryptogram covers an estimate and every scheme wraps such devices in rules of its own.
17. An offline-only terminal never runs the online comparison at all, which is exactly why the issuer's default list falls back to all ones and why a card carrying none is refused by an aircraft for the smallest observation.
18. Offline approval is cryptographically as strong as an online one and commercially riskier, because nobody with a balance in front of them has looked at it and the only controls are the card's own.
19. Store-and-forward is a deferral rather than a decision, and the three shapes of network failure have three correct responses, one of which makes a reversal mandatory whatever the terminal and card

decided locally.

20. There is no single authority in a card transaction — a terminal with an acquirer's policy, a card with an issuer's policy, a merged bitmap neither party can see in full, and a network that is sometimes there — and everything that looks arbitrary about card acceptance falls out of that arrangement.

Sources: EMV Integrated Circuit Card Specifications for Payment Systems, Book 3 (Application Specification) and Book 4 (Cardholder, Attendant, and Acquirer Interface Requirements), EMVCo; EMV Contactless Specifications for Payment Systems, Book C-2 (Kernel 2) and Book C-5 (Kernel 5), EMVCo; Visa Core Rules and Visa Product and Service Rules; Mastercard Switch Rules and Mastercard Developers authorisation message documentation; Commission Delegated Regulation (EU) 2018/389; UK Finance and Financial Conduct Authority guidance on contactless limits; ISO 8583 and ISO 18245.

CHAPTER

7

Proving It Is You

17.0 What this chapter gives you

1. You will be able to explain why the Application Cryptogram is a proof and the three bytes of tag `9F34` are only a claim, and why that difference is where every serious EMV attack has landed.
2. You will be able to walk a CVM List from the top for a given amount, terminal and currency, and say which rule wins and why nothing below it is ever read.
3. You will be able to decode a real tag `8E` object byte by byte: amounts X and Y, the method codes, the condition codes and the continuation bit in byte 1.
4. You will be able to tell offline PIN from online PIN by their key hierarchy, block format, comparator, counter and liability, instead of calling both of them "PIN".
5. You will be able to build an ISO 9564-1 format 0 PIN block by hand and say exactly which twelve PAN digits go into the PAN block.
6. You will be able to diagnose a `6985` on every VERIFY, a terminal that never prompts for a PIN, and an online PIN rejected across a whole estate.
7. You will be able to explain why three wrong guesses at a supermarket till lock the chip while the same card still works at a cash machine.
8. You will be able to say why the schemes could drop signature in 2018 without losing any evidence at all.
9. You will be able to explain why a phone pays for a £400 television at a reader that refuses a £120 plastic card.
10. You will be able to read a contactless trace kernel-aware, so that tag `9F66` does not decode as confident nonsense.

The previous chapter was about proving the card is real. This one is about proving *you* are.

They are different problems solved by different machinery, which is the first thing to get straight. The Application Cryptogram is eight bytes of mathematics that a chip and an issuer host compute independently and compare: either the bytes match or they do not. Cardholder verification has no such property. It produces three bytes, tag `9F34`, and those three bytes are not a computation. They are a *statement*, written by the terminal, about what the terminal believes happened at the keypad. Nobody in the chain is obliged to check whether the statement is true, and for most of the last thirty years nobody did.

Once you see that cardholder verification is a claim rather than a proof, the rest follows: why the card publishes a wish list rather than a rule, why the terminal gets the casting vote, why signature survived for decades without anyone ever comparing two signatures, why the two great EMV attacks of 2010

and 2020 both struck this exact seam, and why the industry's answer was not to fix the seam but to move verification onto a device the criminal does not have.

The plain version

Imagine a members' club with a door.

The club has a rule about identification, but it is not a single rule. It is a list, carried by every member, in order of preference. The first entry the doorman can actually perform, under the circumstances that apply tonight, is the one that gets used. Nothing further down the list is consulted.

A member's card might read like this:

1. Ask the bank to check my four-digit number. Use this one only at cash machines.
2. Ask my own chip to check my four-digit number, sending it in a sealed envelope. Use this whenever the door can do it.
3. Ask my own chip to check my four-digit number, sent openly. Use this whenever the door can do it.
4. Take my signature. Use this whenever the door can do it.
5. Let me in with nothing. Use this only if it is not a cash transaction.

The doorman has his own list, much shorter: it says what he is equipped to do. Some doors have a keypad and some do not. Some have a sealed-envelope machine and some do not. Some can only take a signature.

The whole of cardholder verification is these two lists being walked down together, top to bottom, until they touch.

A worked example, in pounds

You are buying £47.50 of groceries in a Leeds supermarket. Attended till, keypad, full capability. Your card's list has two amounts written at the top of it, which the conditions can refer to: **X is £30.00** and **Y is £100.00**. Neither matters here, because none of your card's five rules happen to mention them, but they are there.

Walk the list.

Rule 1 says *ask the bank, but only at an unattended cash machine.* This is a supermarket till. The condition is not met, so the rule is not applicable. Move on. Note carefully: this is not a failure. The rule simply does not apply tonight.

Rule 2 says *ask the chip, sealed, whenever the door can do it.* Can this till do it? Yes: it has an encrypting keypad and knows how to seal a number for a chip. Condition met, method supported. **This rule wins.** The till stops walking. It asks you for four digits, seals them, hands them to the chip, and the chip says yes or no.

Rules 3, 4 and 5 are never read. The signature line is never reached. This is why two cards in the same shop, on the same day, for the same amount, can behave completely differently: they carry different lists.

Now change one thing. Suppose the till is old and cannot do the sealed-envelope trick. Rule 2 becomes inapplicable and rule 3 wins: same digits, same chip, but sent in the open, which is fine because the wire between keypad and chip is a few centimetres long and inside a tamper-resistant box.

Change one more thing. Suppose the till has no keypad, only a screen and a printer. Rules 2 and 3 both drop out and rule 4 wins: signature. And if the till cannot even print, rule 5 wins: you are let in with nothing at all, which sounds alarming until you remember it is exactly what happens every time you tap for a coffee.

Four ways to prove it, and a fifth

There are, in the classical scheme, four families.

The chip checks the PIN. Your card stores your PIN. Not your bank: your card. The till sends the digits to the chip over the contacts, the chip compares them to what it holds, and answers. The bank is not involved and does not need to be, which is why this method exists: it works when the shop's line to the bank is down, on an aeroplane, in a car park machine at three in the morning. It comes in two flavours, one where the digits travel in the open between keypad and chip and one where they are sealed with the chip's own public key first.

The bank checks the PIN. The chip is not involved at all. The till locks your digits into a box that only the bank's cryptographic hardware can open and sends it with the authorisation request. This is what happens at every cash machine and at most tills outside Europe.

Signature. The till prints a slip, you sign it, and somebody is theoretically supposed to compare it to the strip on the back of the card.

Nothing. The card and the terminal agree, in writing, that this transaction needs no verification.

And the fifth, which arrived with the phone: **the device checks you.** Your phone or watch takes your face, fingerprint or passcode, satisfies itself, and tells the till that verification has already been done. The till never sees anything. Nor, importantly, does the bank.

Three strikes

The chip keeps a counter, usually starting at three. Every wrong PIN takes one off it; a correct PIN puts it back. At zero the chip refuses to check any more PINs, permanently, until the bank sends a specific instruction to reset it — which it can only do while you are online, which is why "put your card in a cash machine and try again" is genuinely useful advice.

The counter lives on the card, not on the account. Three wrong guesses at a supermarket till will lock the chip, and yet the same card will still work at a cash machine using the bank-checks-the-PIN route. Two counters, two places.

The box the PIN travels in

Whenever a PIN moves anywhere, it moves inside a fixed-size scrambled box: sixteen hexadecimal characters, eight bytes, containing in order a digit saying which recipe was used, a digit saying how many digits the PIN has, the PIN itself, and filler.

For a PIN of 1234 the raw box is `2 4 1234 FFFFFFFFFF`: recipe 2, four digits, one-two-three-four, ten `F`s of padding. That is the box used when the chip does the checking, and it is deliberately dull.

For a PIN going to the bank the box is built differently, and this is the clever bit: the account number is mixed in before it is locked. Two people with the same PIN on two different cards produce completely different boxes, so an attacker who collects a million boxes cannot spot that a hundred thousand of them are `1234`. The box for PIN 1234 on card 43219876543210987 is `0412AC89ABCDEF67`, and on any other card it would be something else entirely.

The three-byte receipt

When it is over, the till writes down three bytes: *what method was used*, *under what condition*, and *did it work*. `01 00 02` means "the chip checked a plaintext PIN, the condition was always, and it succeeded". That is the whole record, and the only record.

Here is the sentence to carry to dinner: **nothing in the system forces those three bytes to be true.** The till writes them. The card does not check them. The bank receives them and, historically, believed them.

Where the plain version stops being true

The list-walking picture is correct as far as it goes. Four things it hides matter enormously.

The card proposes; the terminal disposes; and nobody audits

The CVM List is a *preference*, not an instruction. The terminal performs the walk, decides which rule was applicable, performs the method, and authors the result. The card contributes only the list.

Two consequences follow, and both have been exploited. First, if the terminal lies about the outcome, the card has no independent way of knowing. In the 2010 Cambridge attack (Murdoch, Drimer, Anderson and Bond, *Chip and PIN is Broken*, IEEE Symposium on Security and Privacy), a small device between card and terminal swallowed the VERIFY command and answered `9000` — success — on the card's behalf. The terminal recorded a successful PIN. The card, never having been asked, recorded that no PIN had been performed. Both statements went to the issuer in the same message, and issuers were not comparing them.

Second, the result *does* often get bound into the cryptogram — tag `9F34` appears in most CDOL1s — and that binding creates a false sense of safety. The cryptogram authenticates the terminal's claim. It does not make the claim true.

There is also a switch that turns the whole subject off. The Application Interchange Profile, tag `82`, has a bit meaning "cardholder verification is supported". If it is clear, the terminal does not walk the list at all, whatever the list says.

Offline PIN and online PIN are not two flavours of one thing

Calling them both "PIN" hides that they share almost nothing.

Offline PIN travels a few centimetres over the card contacts, is protected either not at all or by the card's own RSA or elliptic-curve public key, is compared by the chip, is counted by the chip, and is never seen by the issuer. Online PIN travels thousands of kilometres, is protected by symmetric keys shared between terminal, acquirer, switch and issuer, is compared inside a hardware security module in a data centre, is counted by the issuer's host, and is never seen by the card.

Different PIN block formats, different key hierarchies, different failure modes, different counters, and in most markets different liability. A terminal estate can support one and not the other; an issuer can permit one and not the other. Treating them as interchangeable is the commonest conceptual error in this area.

The list barely applies to the transaction you actually did today

Amounts X and Y are expressed in the *application* currency, and conditions `'06'` through `'09'` are explicitly conditioned on the transaction being in that currency. Present a sterling-denominated card at a euro-denominated terminal and every amount-based rule silently evaporates. That is not an edge case; it is every foreign holiday.

More fundamentally, most cardholder verification decisions in a modern economy are contactless, and contactless decisions are dominated not by tag `8E` but by limits configured in the reader and by kernel-specific data objects. When your card asks for a PIN after a run of taps, that decision came from counters inside the card application and from reader limits, not from the walk described above.

Signature did not die of insecurity. It died of pointlessness

The protocol's definition of a successful signature CVM is not "the signatures matched", nor even "a signature was obtained". It is, effectively, "the terminal is capable of handling signature". The capability bit is set, the condition is met, the method is deemed performed, and the result byte is set to successful. There is no comparison step anywhere in EMV, because comparison is a human act the specification cannot describe.

Signature was therefore always a null CVM dressed as a real one, and what killed it was the schemes noticing that it created queues, receipts and disputes while providing no evidence. Note the mirror-image error too: "No CVM required" is not the absence of security. It is a deliberate, listed, cryptogram-covered choice by the issuer, appropriate below a threshold the issuer picked.

The technical version

The objects involved

Tag	Name	Length	Source
`82`	Application Interchange Profile	2	ICC
`8E`	Cardholder Verification Method (CVM) List	10 to 252	ICC
`95`	Terminal Verification Results (TVR)	5	Terminal
`99`	Transaction PIN Data	var.	Terminal
`9F17`	PIN Try Counter	1	ICC
`9F2D, 9F2E, 9F2F`	ICC PIN Encipherment Public Key certificate, exponent, remainder	var.	ICC
`9F33`	Terminal Capabilities	3	Terminal
`9F34`	Cardholder Verification Method (CVM) Results	3	Terminal
`9F35`	Terminal Type	1	Terminal
`9F66`	Terminal Transaction Qualifiers (kernels 3, 6, 7)	4	Terminal
`9F6C`	Card Transaction Qualifiers (kernel 3)	2	ICC

The relevant Application Interchange Profile bits, from EMV 4.4 Book 3 Annex C1, are byte 1 bit 5 (`0x10`), "Cardholder verification is supported", and byte 1 bit 2 (`0x02`), "On device cardholder verification is supported". That second bit was reserved for the contactless specifications in EMV 4.3 and only became a named contact-specification bit in 4.4. Byte 2 carries `0x80` "Contactless EMV mode supported", `0x40` "Mobile phone", `0x20` "Contactless transaction" and `0x01` "Relay Resistance Protocol supported".

The CVM List, tag '8E'

The value field is a fixed eight-byte header followed by a variable number of two-byte Cardholder Verification Rules:

Field	Length	Format
Amount X	4	binary
Amount Y	4	binary
CV Rule 1 … CV Rule n	2n	binary

Both amounts are binary integers in the minor units of the application currency, with the decimal point implied by the Application Currency Exponent, tag `9F44`. A list is always at least ten bytes and always an even length; anything else is malformed.

CV Rule byte 1 carries the method in bits 6 to 1, a continuation flag in bit 7, and a reserved bit 8. Bit 7 set means *apply the succeeding CV Rule if this CVM is unsuccessful*; bit 7 clear means *fail cardholder verification if this CVM is unsuccessful*. The method codes, from EMV 4.4 Book 3 Annex C3 Table 43:

Code	Meaning
`'00'`	Fail CVM processing
`'01'`	Plaintext PIN verification performed by ICC
`'02'`	Enciphered PIN verified online
`'03'`	Plaintext PIN verification performed by ICC and signature (paper)
`'04'`	Enciphered PIN verification performed by ICC
`'05'`	Enciphered PIN verification performed by ICC and signature (paper)
`'06'–'0F'`	Biometric CVMs, in pairs: facial `'06'`/`'07'`, finger `'08'`/`'09'`, palm `'0A'`/`'0B'`, iris `'0C'`/`'0D'`, voice `'0E'`/`'0F'`, the even code verified offline by the ICC and the odd code verified online
`'1E'`	Signature (paper)
`'1F'`	No CVM required
`'20'–'2F'`	Reserved for use by the individual payment systems
`'30'–'3E'`	Reserved for use by the issuer
`'3F'`	Not available for use

The biometric codes `'06'` to `'0F'` are an EMV 4.4 addition. In 4.3 that whole range was reserved. If your parser labels `'08'` as RFU, it is a 4.3 parser.

CV Rule byte 2 is the condition:

Value	Condition
`'00'`	Always
`'01'`	If unattended cash
`'02'`	If not unattended cash and not manual cash and not purchase with cashback
`'03'`	If terminal supports the CVM
`'04'`	If manual cash

Value	Condition
'05'	If purchase with cashback
'06'	If transaction is in the application currency and is under X value
'07'	If transaction is in the application currency and is over X value
'08'	If transaction is in the application currency and is under Y value
'09'	If transaction is in the application currency and is over Y value
'0A'–'7F'	RFU
'80'–'FF'	Reserved for use by the individual payment systems

Only condition '03' explicitly tests terminal support, but the terminal must skip any rule whose method it cannot perform regardless, so '03' is the condition you will see on almost every rule that is not amount- or environment-specific.

Decoding a real list

Take this object:

```
8E 12 00 00 0B B8 00 00 27 10 42 01 44 03 41 03 5E 03 1F 02
```

Eighteen bytes of value: eight of header, ten of rules.

Amount X is 00000BB8 = 3000, which with a sterling application currency and exponent 2 is £30.00. Amount Y is 00002710 = 10000, or £100.00.

Rule	Byte 1	Bit 7	Method	Byte 2	Condition
1	42	set	'02' Enciphered PIN verified online	01	If unattended cash
2	44	set	'04' Enciphered PIN verification performed by ICC	03	If terminal supports the CVM
3	41	set	'01' Plaintext PIN verification performed by ICC	03	If terminal supports the CVM
4	5E	set	'1E' Signature (paper)	03	If terminal supports the CVM
5	1F	clear	'1F' No CVM required	02	If not unattended cash, manual cash or cashback

Now give the terminal a Terminal Capabilities of E0 F8 C8. Byte 1 E0 is manual key entry, magnetic stripe and IC with contacts. Byte 3 C8 is SDA, DDA and CDA. Byte 2, the CVM capability field, is F8:

Mask	Bit	Meaning
0x80	b8	Plaintext PIN for ICC verification
0x40	b7	Enciphered PIN for online verification
0x20	b6	Signature (paper)
0x10	b5	Enciphered PIN for offline verification (RSA ODE)
0x08	b4	No CVM required
0x04	b3	Online Biometric
0x02	b2	Offline Biometric
0x01	b1	Enciphered PIN for offline verification (ECC ODE)

F8 sets the top five: all four classical methods plus no-CVM, and no biometrics. Bits b3, b2 and b1 are EMV 4.4 definitions; in 4.3 they were RFU.

For a £47.50 attended purchase in sterling with Terminal Type '22' (operational control: merchant; environment: attended, offline with online capability), the walk goes: rule 1 skipped, condition not met; rule 2 applicable and supported, so **offline enciphered PIN is performed** and the walk stops. If the PIN matches, the terminal writes CVM Results 44 03 02. Note that byte 1 carries the *entire* CV Rule byte 1 that was applied, continuation bit included: 44, not 04.

The VERIFY command and offline plaintext PIN

Offline PIN is carried by the ISO/IEC 7816-4 VERIFY command: class '00', instruction '20', P1 '00'. P2 is the qualifier of the reference data, and EMV assigns two values: '80' for plaintext PIN and '88' for enciphered PIN. A complete plaintext VERIFY for PIN 1234 is:

```
00 20 00 80 08 24 12 34 FF FF FF FF FF
```

Lc is 08 and the data is an ISO 9564-1 format 2 PIN block: control field '2', length nibble '4', the four PIN digits, then 'F' padding to eight bytes. Format 2 exists precisely for this case. ISO describes it as being for local use with offline systems only, and it deliberately contains no account number: the card already knows its own PAN, and there is no benefit in diversifying a block that never leaves the reader.

Status words matter here more than usual:

SW1 SW2	Meaning
9000	PIN correct
63Cx	PIN incorrect; x attempts remain
6983	Authentication method blocked (try counter exhausted)
6984	Referenced data invalidated
6985	Conditions of use not satisfied

6985 is by far the commonest complaint from engineers writing their first EMV client, and it almost always means one of two things: the command sequence is wrong — VERIFY may be issued any time after Read Application Data and before terminal action analysis completes, not before GET PROCESSING OPTIONS — or the card does not support plaintext offline PIN and you should be sending '88'.

Read the PIN Try Counter *before* attempting VERIFY, with GET DATA:

```
80 CA 9F 17 00
```

If it returns zero, EMV requires the terminal not to attempt offline PIN, to set "PIN Try Limit exceeded" in the TVR, and to treat the method as unsuccessful — which, because bit 7 of the rule is usually set, sends the walk to the next rule rather than failing outright. Resetting the counter requires the issuer script command PIN CHANGE/UNBLOCK, instruction `'24'` under the proprietary class, delivered after online authorisation.

Offline enciphered PIN

The RSA form works as follows. The terminal issues GET CHALLENGE, class `'00'`, instruction `'84'`, P1 and P2 both `'00'`, and the card returns an eight-byte unpredictable number. The terminal then assembles a block for encipherment:

Field	Length	Value
Data Header	1	`'7F'`
PIN Block	8	ISO 9564-1 format 2 PIN block
ICC Unpredictable Number	8	The value returned by GET CHALLENGE
Random Pad Pattern	N(IC) - 17	Generated by the terminal

N(IC) is the byte length of the modulus of the key used, so the assembled block is exactly one RSA block. The key is the ICC PIN Encipherment Public Key, recovered from the certificate in tag `9F2D` with the exponent in `9F2E` and, where the modulus does not fit in the certificate, the remainder in `9F2F`. If the card carries no separate encipherment key, the ICC Public Key from `9F46`, `9F47` and `9F48` is used instead. The result goes in the data field of VERIFY with P2 `'88'` and Lc equal to N(IC). The card decrypts, checks that the recovered header is `'7F'`, checks that the recovered unpredictable number equals the one it issued, and only then parses the PIN block.

That unpredictable number is the entire replay defence. Without it, a recording of one enciphered VERIFY would work forever.

EMV 4.4 adds an elliptic-curve form, Offline Data Encipherment, signalled by Terminal Capabilities byte 2 bit 1. It has no random pad, because the ECC construction supplies its own freshness. EMVCo's worked example (document EMV-SWG-NJ24r95, version 1.0a, December 2022) gives the plaintext for PIN 1234 as:

```
Data Header               7F
PIN block (ISO format 2)  24 12 34 FF FF FF FF FF
ICC Unpredictable Number  4B 61 72 6D 65 6C 69 74
```

The terminal generates an ephemeral P-256 key pair, does Diffie-Hellman against the card's public key, derives four AES keys via CMAC and AES, encrypts in counter mode under K1, MACs under K2 with CMAC truncated to eight bytes, and sends the ephemeral public key's x-coordinate concatenated with the ciphertext. The card repeats the derivation with its private key, verifies the MAC, decrypts, checks the trailing unpredictable number and the leading `'7F'`, and — in EMVCo's own words — parses the eight bytes following the header as an ISO-2 PIN block.

Online PIN and PIN block formats

Online PIN never touches the card. It is captured on a PCI PTS POI approved device, formatted into a PIN block, encrypted under a key shared with the acquirer, and carried in the authorisation message. Between capture and the issuer's hardware security module it is translated — decrypted and re-encrypted under the next zone's key — inside HSMs at each hop, and must never exist in the clear outside a secure cryptographic device. That is the substance of the PCI PIN Security Requirements,

currently version 3.1 (March 2021), and of the PCI PTS POI Modular Security Requirements, currently version 7.0.

The formats are defined by ISO 9564-1. The first nibble of the block identifies the format.

Format	Construction	PAN mixed in	Notes
0	`0` | `L` | PIN | `F` padding, XORed with a PAN block	Yes	Equivalent to ANSI X9.8; the workhorse
1	`1` | `L` | PIN | random padding	No	For use where no PAN is available
2	`2` | `L` | PIN | `F` padding	No	Offline use with smart cards only
3	`3` | `L` | PIN | random `A`–`F` fill, XORed with a PAN block	Yes	Format 0 with random rather than fixed fill
4	`4` | `L` | PIN | `A` fill | random, combined with a PAN block under AES	Yes	16 bytes; introduced in ISO 9564-1:2017 for AES

Formats 0 to 3 are eight bytes, matching the 64-bit block size of TDEA. Format 4 is sixteen, matching AES.

Format 0 is worth building by hand once. For PIN 1234 and PAN 43219876543210987:

```
PIN block   0 4 1 2 3 4 F F F F F F F F F F
PAN block   0 0 0 0 9 8 7 6 5 4 3 2 1 0 9 8
XOR         0 4 1 2 A C 8 9 A B C D E F 6 7
```

The PAN block is four zero nibbles followed by the twelve rightmost PAN digits *excluding the check digit*. Getting that exclusion wrong is a classic bug: it produces a block that decrypts to garbage at the issuer and a `55` invalid-PIN response for every cardholder on the estate.

Format 4 is a different animal. The PIN block is `4` | length | PIN | fill digits `A` | sixteen random nibbles, making 32 nibbles. The PAN block is a length indicator (`'0'` to `'7'`, meaning PAN length 12 to 19), the PAN, and zero padding, also 32 nibbles. The PIN block is encrypted with AES, the result XORed with the PAN block, and that encrypted with AES again. For PIN 1234 on PAN 432198765432109870 the clear PIN block begins `44 12 34 AA AA AA AA AA` and the PAN block is `64321987654321098700000000000000`. Migration has been slow enough that PCI SSC published a dedicated information supplement on implementing it and suspended its own compliance dates.

How the issuer actually checks

The issuer's HSM decrypts the PIN block, extracts the PIN, and compares it. It does not store the PIN. There are two dominant methods, both older than EMV by decades.

IBM 3624 with offset. The HSM encrypts validation data — conventionally the PAN, padded to sixteen characters with a chosen pad character — under a PIN verification key, takes the leftmost four hexadecimal digits of the result, and maps them through a decimalisation table to produce the *natural PIN*. Because a natural PIN cannot be chosen by the cardholder, an *offset* is stored: the digit-by-digit modulo-10 difference between the chosen PIN and the natural PIN. At verification the HSM recomputes the natural PIN, applies the offset, and compares. The decimalisation table is a parameter, and a well-known family of attacks in the early 2000s exploited HSM APIs that let an attacker supply a degenerate one.

Visa PVV. A Transformed Security Parameter is built from the eleven rightmost PAN digits excluding the check digit, a single-digit PIN Verification Key Indicator in the range 0 to 6, and the four PIN digits: sixteen digits, eight bytes. That is encrypted under the PIN verification key pair and the result scanned left to right for decimal digits; if four are not found, the scan repeats over the remaining digits with ten subtracted. The four-digit PIN Verification Value is stored on the account.

Neither scheme stores a PIN and neither can recover one. Both store a four-digit residue, which is why an issuer can tell you your PIN is wrong but genuinely cannot tell you what it is.

The CVM Results, tag '9F34'

Three bytes, defined in EMV 4.4 Book 4 Annex A4 Table 33.

Byte	Content
1	CVM Performed: the CV Rule byte 1 that was applied, or `'3F'` meaning "CVM not performed"
2	CVM Condition: the CV Rule byte 2 that was applied
3	CVM Result: `'00'` unknown, `'01'` failed, `'02'` successful

Value `'3F'` in byte 1 is the interesting asymmetry. In a CV Rule it is explicitly "not available for use"; in CVM Results it is the legitimate way of saying nothing was done. A parser sharing one lookup table between rules and results will reject valid CVM Results.

EMVCo's own ECC worked example carries `9F34` as CDOL1 data with the value `01 00 02`: plaintext PIN verified by the ICC, condition Always, successful.

The Terminal Verification Results, byte 3

The whole of TVR byte 3 is cardholder verification. Under EMV 4.4:

Mask	Bit	Meaning
`0x80`	b8	Cardholder verification was not successful
`0x40`	b7	Unrecognised CVM
`0x20`	b6	PIN Try Limit exceeded
`0x10`	b5	PIN entry required and PIN pad not present or not working
`0x08`	b4	PIN entry required, PIN pad present, but PIN was not entered
`0x04`	b3	Online CVM captured
`0x02`	b2	Biometric required but Biometric capture device not working
`0x01`	b1	Biometric required, Biometric capture device present, but Biometric Subtype entry was bypassed

Bit 3 was named "Online PIN entered" in EMV 4.3 and renamed "Online CVM captured" in 4.4 to accommodate online biometrics. The name changed; the bit did not. An unrecognised CVM code

sets bit 7 and the method is treated as unsuccessful, which is how a card personalised with a scheme-proprietary code fares on a terminal that has never heard of it.

These bits are what issuer and terminal action codes actually test. An Issuer Action Code – Denial with byte 3 bit 8 set means "decline anything where cardholder verification failed", and that is where the CVM decision becomes a financial outcome.

Contactless: a different machine wearing the same words

The contactless world is not the contact world with an aerial. EMV Contactless Book B defines an Entry Point that selects a kernel, and Books C-1 through C-7 define kernels 1 to 7, each belonging to a payment system. Kernel 2 is Mastercard's; kernel 3 is Visa's. Book C-8, Kernel 8, was published by EMVCo in October 2022 as a common kernel licensable on the same royalty-free terms as the contact specification, with elliptic-curve card authentication and explicit support for biometric and mobile cardholder verification methods.

The tag collision problem. The most dangerous fact in contactless decoding is that the same tag means different things in different kernels. Tag `9F66` is the Terminal Transaction Qualifiers for Entry Point and for kernels 3, 6 and 7, but kernel 2 defines `9F66` as PUNATC(Track2). Tag `9F6B` is the Card CVM Limit in kernel 3 and Track 2 Data in kernel 2. Tag `9F6E` is the Form Factor Indicator in kernel 3, Third Party Data in kernel 2 and Enhanced Contactless Reader Capabilities in kernel 4. A decoder that does not know which kernel produced the trace will produce confident nonsense.

Kernel 3, the Visa model. The reader supplies Terminal Transaction Qualifiers, tag `9F66`, four bytes, usually through the PDOL:

Byte	Bit	Meaning
1	b8	MSD supported
1	b7	VSDC supported
1	b6	qVSDC supported
1	b5	EMV contact chip supported
1	b4	Offline-only reader
1	b3	Online PIN supported
1	b2	Signature supported
1	b1	Offline Data Authentication for Online Authorisations supported
2	b8	Online cryptogram required
2	b7	CVM required
2	b6	(Contact chip) Offline PIN supported
3	b8	Issuer Update Processing supported
3	b7	Consumer Device CVM supported

The card replies with Card Transaction Qualifiers, tag `9F6C`, two bytes:

Byte	Bit	Meaning
1	b8	Online PIN Required
1	b7	Signature Required

Byte	Bit	Meaning
1	b6	Go Online if Offline Data Authentication Fails and Reader is online capable
1	b5	Switch Interface if Offline Data Authentication fails and Reader supports VIS
1	b4	Go Online if Application Expired
1	b3	Switch Interface for Cash Transactions
1	b2	Switch Interface for Cashback Transactions
2	b8	Consumer Device CVM Performed
2	b7	Card supports Issuer Update Processing at the POS

That is the whole contactless CVM negotiation for Visa: two bits, one saying "ask for a PIN", one saying "the device already did it". The CVM List is not consulted.

Kernel 2, the Mastercard model. Here reader configuration drives the decision. Three tags matter, all six-byte numeric amounts:

Tag	Name
DF8124	Reader Contactless Transaction Limit (No On-device CVM)
DF8125	Reader Contactless Transaction Limit (On-device CVM)
DF8126	Reader CVM Required Limit

The kernel instantiates a single working Reader Contactless Transaction Limit from DF8125 if the card indicates on-device cardholder verification is supported and from DF8124 otherwise — which is exactly why a phone can pay for a £400 television at a reader that will refuse a £120 plastic card. Separately, if the amount reaches DF8126, the kernel instantiates the CVM capability field of Terminal Capabilities with "CVM Required", forcing a method to be selected.

The card's own counters do the rest: consecutive-transaction and cumulative-amount counters live in the card application, not the terminal, and when they trip the card asks for verification. That is why the PIN prompt appears at an apparently random shop.

Consumer Device CVM

CDCVM is what happens when the phone, watch or laptop verifies you and the terminal takes its word. EMVCo defines it as authentication performed on the consumer's own device: passcode, password, pattern, or a biometric such as fingerprint, iris, voice or facial recognition.

Mechanically, in kernel 3 the device sets Card Transaction Qualifiers byte 2 bit 8, "Consumer Device CVM Performed", and the reader stops asking. In kernel 2 the device signals on-device CVM support and the reader promotes itself to the higher limit. In both cases the terminal performs no verification whatsoever.

The security consequence is precisely stated by EMVCo's own CDCVM task force chair: *unlike an online PIN, a CDCVM cannot be seen by the issuer.* The issuer receives an assertion that some unspecified

mechanism, on some unspecified device, from some unspecified manufacturer, succeeded. Hence three pieces of EMVCo scaffolding: the **EMV CDCVM Security Requirements**, a **Security Evaluation Process** for software-based mobile payment, and an **EMV CDCVM Solution ID and Database** assigning each registered solution a short identifier plus metadata — name, provider, type, operating system — so an issuer can at least know *which* CDCVM it is being asked to trust. EMVCo has also worked with GlobalPlatform on the Secure Element Broker Interface and with the FIDO Alliance since 2016 so that FIDO's biometric certification covers EMVCo's performance requirements.

The regulatory position is more generous than the technical one. Under the PSD2 regulatory technical standards, Commission Delegated Regulation (EU) 2018/389, Article 11 permits contactless payments without strong customer authentication where the individual amount does not exceed EUR 50, the cumulative amount since the last strong authentication does not exceed EUR 150, or the number of consecutive contactless transactions since the last strong authentication does not exceed five. CDCVM satisfies strong customer authentication in its own right, which is why digital wallets have never been subject to those caps.

In the United Kingdom the corresponding caps were £100 per transaction and £300 cumulative, in force from 15 October 2021. From **19 March 2026** the Financial Conduct Authority removed both, allowing firms with sufficiently strong fraud controls to set their own limits or none at all; in practice the major banks retained £100 when the rules changed, and raising it also requires changes at acceptance terminals and in the industry rules that govern them. Mobile wallets were never inside that regime.

The death of signature, precisely

Mastercard announced in October 2017 that from April 2018 merchants in the United States and Canada would no longer be required to obtain a cardholder signature for in-store credit and debit transactions, and confirmed in 2018 that from April 2019 issuers globally would no longer be required to include a signature panel on Mastercard products. Visa, American Express and Discover made comparable changes for North America over the same period.

It could be dropped so cleanly because it was never verified in the protocol. CVM code `'1E'` under condition `'03'` asks only whether the terminal supports signature; if the capability bit is set, the method is applicable, the terminal obtains the signature and reports success. There is no field for a comparison, no field for its result, and no obligation on anyone to perform one. Removing signature removed a receipt, a queue and a dispute category. It removed no evidence, because there was none.

What the attacks actually taught

Three pieces of work define the state of understanding.

Chip and PIN is Broken (Murdoch, Drimer, Anderson and Bond, IEEE S&P 2010) inserted a wedge between card and terminal that intercepted VERIFY and returned `9000` without forwarding it. The terminal recorded a successful offline PIN in `9F34`. The card, never asked, recorded in its own Card Verification Results that offline PIN verification had not been performed. Both went to the issuer in the same authorisation message. Issuers comparing the terminal's claim with the card's own account would have caught it; most did not.

The EMV Standard: Break, Fix, Verify (Basin, Sasse and Toro-Pozo, IEEE S&P 2021) did the same to contactless. A man-in-the-middle Android application modified the Card Transaction Qualifiers in flight: clearing byte 1 bit 8, so the terminal no longer believed online PIN was required, and setting byte 2 bit 8, so it believed the Consumer Device CVM had been performed. Demonstrated on real terminals with Visa Credit, Visa Electron and V Pay cards. The CTQ is not integrity-protected, which is the whole of the vulnerability.

Card Brand Mixup (same authors, USENIX Security 2021) made a Mastercard card behave as a Visa transaction so that the Visa PIN bypass applied. It was demonstrated on Mastercard credit, Mastercard debit and Maestro cards, including a transaction above 400 US dollars. Mastercard's response was to check at authorisation that the Application Identifier is consistent with the card brand.

In every case the cryptogram was valid and the card genuine. What failed was that the record of *who was standing there* is authored by one party rather than agreed between two, and is not cryptographically bound to the act it describes.

A short field guide to failures

Symptom	Usual cause
6985 on every VERIFY	Command out of sequence, or card does not support that flavour of PIN
6984 or 6983 immediately	PIN Try Counter already at zero; read 9F17 first
Terminal never prompts for PIN	AIP byte 1 bit 5 clear, or no rule's method is in Terminal Capabilities byte 2
CVM works at home, not abroad	Conditions '06'–'09' inert because transaction currency differs from application currency
Parser rejects a valid 9F34	Byte 1 value '3F' treated as an invalid CV Rule instead of "CVM not performed"
9F66 decodes as nonsense	Kernel 2 trace parsed with a kernel 3 tag table
Phone allowed above the card limit	DF8125 in force rather than DF8124; working as designed
Enciphered offline PIN always fails	ICC Unpredictable Number not echoed byte-for-byte, or wrong public key
Online PIN rejected across an estate	PAN block built from the wrong twelve digits, or check digit not excluded
Biometric CVM codes shown as RFU	EMV 4.3 tables in use; codes '06'–'0F' are 4.4

The chapter in one paragraph

The card carries an ordered wish list, tag 8E, of ways it will accept being satisfied that you are you, with two amounts and a condition attached to each entry. The terminal walks that list from the top, skipping entries whose condition is unmet or whose method it cannot perform, and stops at the first one it can do. If the method is offline PIN, the digits go to the chip in an ISO 9564-1 format 2 block inside a VERIFY command and the chip answers, decrementing its own try counter on failure. If the method is online PIN, the digits never touch the chip; they go into a format 0 or format 4 block, encrypted under keys shared with the acquirer, and are checked in a hardware security module against a stored offset or PIN verification value. If the method is signature, nothing is verified at all. Whatever happens, the terminal writes three bytes — method, condition, outcome — and forwards them. Contactless replaces the walk with reader limits and two bits in a kernel-specific object, and the phone replaces the whole exercise with a face the terminal cannot see and the issuer cannot verify. The eight bytes of the cryptogram are proof. These three bytes are testimony. Every serious attack on EMV in the last two decades has been an attack on the difference.

17.98 Common wrong ideas

Wrong: the card decides how you are verified. **Right:** the card publishes an ordered preference in tag 8E; the terminal performs the walk, decides which rule was applicable, performs the method and authors the result.

Wrong: because tag 9F34 is bound into the cryptogram, its contents must be true. **Right:** the cryptogram authenticates the terminal's claim without making the claim true, which is exactly what the 2010 Cambridge wedge exploited.

Wrong: offline PIN and online PIN are two flavours of one thing. **Right:** they share almost nothing — different block formats, different key hierarchies, different comparators, different counters and in most markets different liability.

Wrong: three wrong PINs lock the account. **Right:** the try counter lives on the card, so the chip refuses further offline PIN checks while the same card still works at a cash machine on the online route.

Wrong: signature was dropped because it was insecure. **Right:** code '1E' under condition '03' asks only whether the terminal supports signature; there is no comparison step anywhere in EMV, so removing it removed a queue and a receipt but no evidence.

Wrong: "No CVM required" is the absence of security. **Right:** it is a deliberate, listed, condition-bound and cryptogram-covered choice by the issuer, appropriate below a threshold the issuer picked.

Wrong: the amounts X and Y govern your spending wherever you are. **Right:** conditions '06' to '09' only apply when the transaction is in the application currency, so every amount-based rule silently evaporates on a foreign holiday.

Wrong: the PIN prompt after a run of taps came from walking the CVM List. **Right:** it came from counters inside the card application and from limits configured in the reader; kernel 3 settles the question with two bits and kernel 2 with DF8124, DF8125 and DF8126.

Wrong: a value of '3F' in tag 9F34 byte 1 is invalid. **Right:** '3F' is "not available for use" in a CV Rule but is the legitimate way of saying "CVM not performed" in CVM Results, and a parser sharing one table between the two will reject valid data.

Wrong: Consumer Device CVM gives the issuer what an online PIN gives it. **Right:** a CDCVM cannot be seen by the issuer, which receives only an assertion — hence EMVCo's security requirements, evaluation process and solution database so an issuer at least knows which mechanism it is trusting.

17.99 Chapter summary in 20 lines

1. Proving the card is genuine and proving the cardholder is entitled are separate problems solved by different machinery.
2. The Application Cryptogram is eight bytes computed independently by chip and issuer and compared; cardholder verification produces three bytes of terminal-authored testimony in tag 9F34.
3. The card carries an ordered wish list, the CVM List at tag 8E, of the ways it will accept being satisfied that you are you.
4. That list opens with two amounts, X and Y, in the minor units of the application currency, followed by two-byte rules pairing a method with a condition.
5. The terminal walks the list from the top, skipping any rule whose condition is unmet or whose method it cannot perform, and stops at the first one it can do.
6. Nothing below the winning rule is ever read, which is why two cards in the same shop for the same amount can behave completely differently.
7. Byte 1 of a rule carries the method and a continuation bit deciding whether failure moves to the next rule or fails cardholder verification outright.
8. There are four classical families — offline PIN checked by the chip, online PIN checked by the issuer, signature, and no CVM — plus the biometric codes added in EMV 4.4 and, in practice, the device.

9. Offline PIN travels a few centimetres in an ISO 9564-1 format 2 block inside a VERIFY command and is compared by the chip against a counter the chip itself keeps.
10. Online PIN never touches the chip: it goes into a format 0 or format 4 block, is translated inside HSMs at every hop, and is checked in the issuer's own hardware security module.
11. Issuers store no PIN at all, only an IBM 3624 offset or a Visa PIN Verification Value, which is why they can tell you the PIN is wrong but genuinely cannot tell you what it is.
12. The PIN Try Counter lives on the card rather than the account, and only an issuer script delivered after online authorisation can reset it.
13. Enciphered offline PIN binds the card's own unpredictable number into the encrypted block, and that number is the entire replay defence.
14. Signature was never verified anywhere in the protocol: success is declared on terminal capability alone, so it was always a null CVM dressed as a real one.
15. That is why it could be dropped from 2018 onwards so cleanly — removing it removed a receipt, a queue and a dispute category, and no evidence.
16. Whatever happens, the terminal writes down the rule applied, the condition applied and the outcome, and nothing in the system forces those three bytes to be true.
17. The 2010 Cambridge wedge answered VERIFY on the card's behalf and the 2021 contactless work rewrote the Card Transaction Qualifiers in flight; in both the cryptogram was valid and the card genuine.
18. Contactless replaces the walk entirely, with reader limits in kernel 2 and two bits of a kernel-specific object in kernel 3.
19. Consumer Device CVM moves verification onto a device the terminal cannot see and the issuer cannot verify, which is why wallets were never inside the contactless caps.
20. Every serious attack on EMV in two decades has been an attack on the gap between eight bytes that are proof and three bytes that are only testimony.

Sources: EMV ICC Specifications for Payment Systems v4.4, Book 3 sections 6.5 and 10.5 and Annexes C1 and C3, Book 4 Annexes A1 and A4, and Book 2 on PIN encipherment; EMV Contactless Books B, C-2, C-3 and C-8; EMVCo document EMV-SWG-NJ24r95 v1.0a, December 2022, for the ECC worked examples; EMVCo on CDCVM, October 2020; ISO 9564-1:2017 and ISO/IEC 7816-4; PCI PIN Security Requirements v3.1 and PCI PTS POI v7.0; Commission Delegated Regulation (EU) 2018/389 Article 11; FCA and UK Finance contactless guidance, March 2026; Mastercard newsroom, 2017 and 2018; Murdoch, Drimer, Anderson and Bond (IEEE S&P 2010) and Basin, Sasse and Toro-Pozo (IEEE S&P 2021, USENIX Security 2021). Bit tables cross-checked against openemv/emv-utils, lumag/emv-tools, CardContact OpenSCDP and emvlab.org.

Contactless

18.0 What this chapter gives you

1. You will be able to explain where a card with no battery gets its power, and why the honest description is mutual inductance in the near field rather than radio reception.
2. You will be able to say why a card works at two centimetres and does nothing at fifteen, and why that short range is a power-budget consequence and not a privacy guarantee.
3. You will be able to describe how a card that cannot transmit still answers, by shorting a load across its own coil and letting the reader read the dips in its own field.
4. You will be able to name every layer a single tap traverses, from ISO/IEC 14443-2 up through T=CL, ISO/IEC 7816-4 APDUs, the EMV entry point and the kernel books.
5. You will be able to explain why the tap limit is a lost-and-stolen control with no relationship to the radio, and name the three independent places that enforce it.
6. You will be able to trace the UK limit from £10 in 2007 to £100 and £300 in 2021 and to the removal of the mandated caps on 19 March 2026, and say what "the limit" now means.
7. You will be able to separate eavesdropping, which reaches metres and yields a transaction bound to one amount and one counter, from relaying, which creates a new transaction.
8. You will be able to explain why a Frame Waiting Time of nearly five seconds leaves the protocol no obstacle whatever to a relay.
9. You will be able to say what the Relay Resistance Protocol measures, why both entropies end up inside the cryptogram, and why it still works less well than the design suggests.
10. You will be able to explain why a ticket gate cannot ask your bank, what it verifies before it opens, and why your statement shows one number, a day late, matching no fare on any poster.

There is a moment, at the till, when nothing appears to happen. You hold a card near a lit square of plastic, the square beeps, and you walk away. No slot, no keypad, no paper. The whole event is shorter than the sentence describing it.

That moment is the most heavily specified two-thirds of a second in retail. It involves an international standard for radio, a second layer for how bytes are framed on that radio, a third for how smart cards talk at all, a fourth from EMVCo describing which of eight payment kernels should wake up, and a fifth of scheme rules and regulatory technical standards deciding whether you should have been asked for a PIN. Almost all of it is invisible, and almost all of the public understanding of it is wrong in the same three ways.

This chapter is about all five layers, in order, from the magnetic field upward.

The plain version

Start with the card. It has no battery. Cut one open and you will find a chip the size of a grain of rice and a loop of very thin wire, or a printed spiral of aluminium, running around the inside edge of the plastic. That is the whole machine. Nowhere for a battery to go.

So where does the power come from? The reader.

Think of a guitar. Pluck a string on one guitar, hold a second guitar close by, and the matching string on the second guitar starts to hum on its own. Nobody touched it. The energy came across the air, and the second string was built to be sympathetic to exactly that note.

The reader in the shop is the plucked string. It hums a note — not a sound, but a magnetic note, oscillating 13,560,000 times every second — constantly, all day, into empty air, whether or not anyone is there. The loop of wire in your card is the sympathetic string. Bring it into the hum and a small current begins to flow around the loop. That current is rectified, smoothed and fed to the chip, and the chip wakes up. It has no idea what time it is or how long it slept. It only knows it now has power.

Now the harder half. Once the card is awake, how does it answer?

You might assume it hums back. It cannot: it has barely enough power to run its own arithmetic, let alone to broadcast. So it does something stranger. It interferes with the reader's own hum.

Imagine standing at a window at night, looking at a lighthouse. You cannot signal back — you have no lamp. But if you could put your hand in front of the lighthouse's beam, and take it away, and put it back, in a rhythm, then someone standing next to the lighthouse would see its own light flicker. You would be talking using their light.

That is what the card does. It repeatedly shorts and unshorts a load across its own coil. Each time it does, it drags a tiny amount of energy out of the reader's field, and the reader — which constantly monitors how much energy its own field is costing it — sees a dip. Dip, no dip, dip, dip, no dip. That is the card talking. It is speaking in the reader's voice.

Why you have to hold it so close

The hum falls away brutally fast. Not gently, the way sound fades as you walk from a speaker, but savagely: double the distance and you have roughly an eighth of the strength. This is why a card works at two centimetres and does nothing at fifteen. Nobody imposed that rule. It is the shape of the physics.

A worked example: £8.40 for coffee

You order a flat white and a pastry. £8.40. The barista taps the total into the till and the till pushes the number to the reader, whose little screen lights up. You present your card. Roughly this happens, in this order.

The card enters the field and powers up. The reader has been calling out, in effect, "anyone there?" — repeatedly, several times a second, all day long. Your card answers with a short serial number so the reader can tell it apart from any other card in range. The reader picks it and says "you, then."

The reader asks for the card's list of payment applications, and picks the highest-priority one it also supports.

The reader hands over the facts of the sale: eight pounds forty, pounds sterling, United Kingdom, today's date, and a random number it has just invented and will never use again.

The card does its sum. Using a secret key that has never left the chip and can never be read out of it, it computes an eight-byte answer depending on the amount, the currency, the date, that random number, and a counter inside the card that goes up by one every time the card is used. It hands the answer back with the counter's current value.

The reader beeps. You leave. The reader sends that answer to your bank, and your bank — which holds the other copy of the secret key — does the identical sum and checks it got the identical answer.

Those eight bytes are the whole security model, and here is the point people miss: they are exactly the eight bytes you would get if you had pushed the card into the slot and typed your PIN. Same chip, same key, same maths. The radio changed nothing about the proof that the card is real.

So why is there a tap limit?

Because the proof that the *card* is real is not the same as the proof that *you* are entitled to use it.

When you type a PIN you are answering a second question. Contactless, below a threshold, skips that question for the sake of speed. So the threshold is not protecting you against anything happening in the air. It protects you against a much older and duller crime: someone taking your card out of your coat pocket and spending it in a newsagent before you notice.

The limit is therefore built as a leaky bucket. Two numbers, not one. In the United Kingdom, from 15 October 2021, the rules said: no single tap above £100, and no more than £300 of taps in total since the last time you actually proved who you were.

Work it through with a thief. They lift your card at 09:00 and buy £95 of gift cards. Running total: £95. Again: £190. Again: £285. On the fourth £95 attempt the running total would reach £380, so the answer comes back: not without a PIN. The thief does not have the PIN. The bucket has emptied, and it will not refill until somebody types four digits or inserts the card into a machine.

Three hundred pounds is the size of the hole the industry decided to leave open. Everything above it meets the same wall as always.

Two thieves and a very long wire

There is one attack the physics genuinely does invite, and it matters because it is the only one where the radio is the point.

Suppose two thieves. One stands near you on a train platform with a device in a bag; the other stands at a till a mile away. The two devices are connected by a phone call. When the shop's reader asks its question, the thief in the shop relays it down the line to the thief on the platform, whose device asks your pocket. Your card, which cannot see, answers honestly. The answer goes back down the phone and into the shop reader.

Nobody forged anything. Nobody broke any cryptography. The thieves simply made a very long extension lead between a real reader and a real card. This is a relay attack, and the defences are not about secrecy. They are about time and about the PIN — which is exactly why the tap limit matters more than people think.

Buses and barriers

Finally, the special case that breaks the whole model: the ticket gate.

A shop's reader can wait a second or two while the message goes to your bank and back. A ticket gate cannot. Behind you there are forty people walking at three miles an hour, and the gate has a few

hundred milliseconds to decide before somebody's nose meets a barrier. There is no time to ask your bank.

So it does not ask. It takes the card's cryptographic answer, checks it against a locally held list of cards known to be bad, opens the gate, and files the answer away. Then it does the same at your exit gate, and again tomorrow. At the end of the day a computer adds up everything you did, works out the cheapest legal price for that set of journeys — applying the daily cap, the weekly cap, the off-peak rates — and sends your bank a single request for one total.

This is why the entry on your statement is one number matching no fare on any poster, and why it arrives a day or two late. The gate was never charging you. It was collecting evidence.

Where the plain version stops being true

The card is not "powered by radio waves", and the distinction has consequences. The guitar analogy is closer than it sounds: this is resonant magnetic coupling in the near field, not radio reception. At 13.56 MHz a full wavelength is about 22 metres and the boundary between near field and far field sits at roughly 3.5 metres. Everything between card and reader happens deep inside that boundary, where energy is exchanged by mutual inductance between two coils rather than radiated away. That is why the working range is centimetres. But it also means the plain version's reassurance is wrong in a specific way: the range at which a card can be *powered and transacted with* is not the range at which it can be *overheard*. Radiated leakage is a separate phenomenon with a much longer reach, and published work has recovered the reader-to-card channel at several metres. Short range is not a privacy guarantee; it is a power-budget consequence.

The £100 was never a technical fact, and as of 2026 it is not a rule either. Nothing in any radio standard, chip specification or EMV kernel knows what a pound is. The limit lives in three places at once — configured amounts inside the terminal, counters inside the card chip, velocity rules inside the issuer's authorisation host — and it exists because a regulator wrote it down. The UK's £100 and £300 came from Article 11 of the onshored Strong Customer Authentication regulatory technical standards. On 19 December 2025 the Financial Conduct Authority announced it was removing those mandated caps, and the change took effect on 19 March 2026, leaving firms free to set their own limits provided they have adequate fraud controls. Most major UK issuers did not immediately move. So the correct statement in 2026 is not "the limit is £100" but "your issuer's limit is probably still £100, and it is now their choice."

"Contactless is less secure" and "contactless is exactly as secure" are both wrong. The cryptogram is identical; the interface changes nothing about proving the card is genuine. But the seam is real and it is not where people look. It is in cardholder verification: the data elements that tell a terminal *whether* verification happened are, in some scheme configurations, not covered by the cryptogram. That is a specific, documented, demonstrated flaw with a paper trail, and it has nothing to do with anyone reading your card through your trousers.

The gate has not taken an IOU. The transit picture implies the barrier trusts you and sorts it out later. It does not trust you. Before that gate opened it verified a digital signature generated by your card's own private key and obtained a full transaction cryptogram over data it supplied. What is deferred is not the proof but the *authorisation*, the question of whether your bank will honour it. Those are different things, they fail differently, and the transit operator carries the gap between them as a real financial risk with a name.

The technical version

The radio: ISO/IEC 14443

The physical and link layers of every contactless payment card in the world are defined by ISO/IEC 14443, a four-part standard originally titled *Identification cards — Contactless integrated circuit cards — Proximity cards* and retitled in later editions to *Cards and security devices for personal identification — Contactless proximity objects*. Part 1 covers physical characteristics; Part 2, radio frequency power and signal interface; Part 3, initialization and anticollision; Part 4, transmission protocol.

The standard's vocabulary matters, because EMV documents use it throughout. The reader is a **PCD**, a Proximity Coupling Device. The card is a **PICC**, a Proximity Integrated Circuit Card. Neither term says anything about payments; the same standard carries transport tickets, passports and door badges.

Part 2 sets the numbers everything else hangs from.

Parameter	Value
Carrier frequency, fc	13,56 MHz ± 7 kHz
Minimum operating field, Hmin (Class 1)	1,5 A/m rms
Maximum operating field, Hmax (Class 1)	7,5 A/m rms
Subcarrier frequency, fs	fc/16, approximately 847 kHz
Initialisation bit rate	fc/128, approximately 106 kbit/s
Higher bit rates	fc/64 approx. 212, fc/32 approx. 424, fc/16 approx. 848 kbit/s
Elementary time unit	1 etu = 128 / (D x fc); 9,44 µs at D = 1

The field-strength window is the most under-appreciated engineering constraint in the system. A card must work at 1,5 A/m and survive 7,5 A/m: five to one in field terms and considerably more in delivered power. The chip therefore carries a shunt regulator whose job at close range is to throw energy away as heat and whose job at long range is to run a public-key operation on almost nothing. Later editions of Part 2 define six PICC classes with different antenna sizes and correspondingly different windows, from Class 1 at 1,5 to 7,5 A/m up to Class 6 at 4,5 to 18 A/m.

The two signalling variants are not compatible and both are mandatory for payment readers.

Type A. Reader to card: amplitude shift keying at 100 % modulation depth — the field is switched fully off in short pauses — with modified Miller coding. Card to reader: on-off keying of the 847 kHz subcarrier, Manchester coded. The 100 % pauses mean the card loses power entirely during each pause and must ride through on a reservoir capacitor.

Type B. Reader to card: amplitude shift keying at roughly 10 %, with the modulation index specified as between 8 % and 14 %, NRZ-L coded. The field never drops out, so the card's supply is steadier. Card to reader: BPSK modulation of the same subcarrier.

In both directions the card's reply is **load modulation**. The card switches an impedance across its coil; the reader observes the reflected change in its own antenna's load, appearing as sidebands at fc ± fs. The card never generates a carrier of its own. This matters enormously to relay attacks and their defeat: the card's timing is not free-running but locked to the reader's carrier, because that carrier is also the card's clock.

Above Part 2 sits Part 3. For Type A, the reader continuously issues REQA (or WUPA for halted cards), a card in the field responds with ATQA, and a bitwise anticollision loop over the UID resolves any

collision, cascading through up to three levels for 4-, 7- or 10-byte UIDs; the reader then issues SELECT and receives SAK. Type B uses REQB/WUPB, ATQB and ATTRIB with a slot-marker scheme instead. Both terminate at the same place: one selected card, everything else told to be quiet. Two cards in a wallet do not "confuse" the reader in the way folklore suggests; they cause a collision the standard resolves deterministically, ending in a clean read or a "present one card only" prompt.

Part 4 defines the half-duplex block transmission protocol, universally called **T=CL**. The reader sends RATS (Request for Answer To Select); the card returns ATS, which declares its capabilities. Two of those set the shape of everything above.

The first is frame size. FSDI and FSCI index a fixed table: 16, 24, 32, 40, 48, 64, 96, 128 and 256 bytes. Anything longer must be chained.

The second is the **Frame Waiting Time**:

FWT = (256 x 16 / fc) x 2^FWI

with FWI in the range 0 to 14 and a maximum FWT of approximately 4 949 ms. If the card needs longer than the FWT it has declared, it sends S(WTX) with a Waiting Time eXtension Multiplier in the range 1 to 59, and the reader grants a temporarily extended FWT, capped at FWTMAX.

That maximum figure — nearly five seconds — should be held in mind. It is the reason the protocol, as designed, offers no obstacle whatever to a relay.

On top of T=CL, the card speaks ISO/IEC 7816-4 command and response APDUs, exactly as it would over the contacts. From the payment application's point of view the radio is a transport. On top of that sits EMV.

EMVCo layers its own requirements over ISO/IEC 14443 in the **EMV Level 1 Specifications for Payment Systems, EMV Contactless Interface Specification**, which reached Version 3.1 in December 2020. Specification Bulletin 245, announced by EMVCo on 2 June 2021, required all compliant contactless terminals to decode IQ modulation — to demodulate the in-phase and quadrature components of the card's load modulation rather than amplitude alone. The stated motivations were reliability, positioning tolerance and transit gate throughput, which tells you where the pressure comes from.

Power harvesting, in detail

The card's antenna is a planar coil, typically a small number of turns of etched or wire-embedded conductor running around the perimeter of an ID-1 card body of 85,60 mm x 53,98 mm. It forms a resonant tank with a capacitance that is partly on-chip and partly distributed, tuned so that the loaded resonance sits at or slightly above 13,56 MHz.

Inside the chip, four things happen in parallel from the moment the field arrives. A rectifier converts the induced AC into DC and charges a reservoir. A shunt regulator clamps the supply and dissipates surplus energy — the closer the card, the more it must waste. A clock extractor derives the system clock by dividing the 13,56 MHz carrier, which is why the card has no independent notion of elapsed time. A power-on-reset circuit holds the logic in reset until the supply is stable.

The consequences shape the payment protocol above. There is no clock across power cycles, so the card cannot enforce "no more than one transaction per minute" by itself. There is no battery-backed counter, so any anti-replay counter must live in non-volatile memory and survive the field being pulled away mid-write, which is why torn-transaction recovery is a named feature of contactless kernels. And there is a hard energy budget for asymmetric cryptography: an RSA or ECC operation at 1,5 A/m

is genuinely difficult, which is why contactless offline data authentication was optimised into a fast variant rather than reusing the contact flow unchanged.

The EMV layer: entry point and kernels

A contactless terminal begins not by selecting an application directly but by selecting the **Proximity Payment System Environment**, whose file name is the ASCII string `2PAY.SYS.DDF01` — the contactless counterpart of `1PAY.SYS.DDF01`. The response is a directory of the card's applications, each with an ADF Name (tag `4F`) and priority.

The terminal matches those against its own configured list, and the match determines which **kernel** processes the transaction. EMVCo publishes the architecture in *EMV Contactless Specifications for Payment Systems*: Book A for architecture and general requirements, Book B for the entry point, Books C-1 through C-8 for the individual kernels, and Book D for the contactless communication protocol.

Book	Kernel	Scheme
C-2	Kernel 2	Mastercard
C-3	Kernel 3	Visa
C-4	Kernel 4	American Express
C-5	Kernel 5	JCB
C-6	Kernel 6	Discover
C-7	Kernel 7	UnionPay
C-8	Kernel 8	EMVCo (scheme-neutral)

Book C-1 defines Kernel 1, built on the EMV common core definitions; the scheme mapping quoted for it in secondary sources is inconsistent and is omitted here rather than guessed. Book C-8, *Kernel 8 Specification*, Version 1.0, was published on 5 October 2022 to reduce the number of kernels the industry maintains; it specifies elliptic curve cryptography, a secure channel, biometric verification and optional on-card data storage.

The practical significance of the split is that "the contactless limit" is not one setting. It is a different named data object in each kernel. In Kernel 2 the terminal-side amounts are carried in proprietary tags in the `DF81xx` range:

Tag	Kernel 2 data object
`DF8123`	Reader Contactless Floor Limit
`DF8124`	Reader Contactless Transaction Limit (No On-device CVM)
`DF8125`	Reader Contactless Transaction Limit (On-device CVM)
`DF8126`	Reader CVM Required Limit
`DF811B`	Kernel Configuration

Their meanings, in the vocabulary the US Payments Forum uses: above the **contactless floor limit** the transaction must be authorised online by the issuer; above the **CVM required limit** cardholder verification must be performed using the full capability of the terminal; above the **contactless transaction limit** the transaction cannot use the contactless interface at all and the customer is directed to insert or swipe. That last limit is split according to whether on-device cardholder verification is available, which is the mechanism by which a phone can pay £400 where a plastic card cannot.

Kernel 3 does the equivalent work with two well-known tags. The **Terminal Transaction Qualifiers**, tag `9F66`, four bytes, is sent to the card in the PDOL and states what the terminal can and will do.

The **Card Transaction Qualifiers**, tag `9F6C`, two bytes, comes back and states what verification the card believes should happen. Tag `9F6E`, four bytes, carries the Form Factor Indicator, which is how an issuer learns whether it was a card, a phone or a watch.

The transaction data itself is the same set of objects used on the contact interface: Application Interchange Profile (`82`), Application File Locator (`94`), Amount Authorised (`9F02`), Transaction Currency Code (`5F2A`), Terminal Country Code (`9F1A`), Unpredictable Number (`9F37`), Application Transaction Counter (`9F36`), Application Cryptogram (`9F26`), Cryptogram Information Data (`9F27`), Issuer Application Data (`9F10`) and CVM Results (`9F34`). Offline data authentication is normally **fDDA** (fast DDA) or **CDA** (Combined Data Authentication), both producing a dynamic signature under the card's own private key rather than the static signature of SDA. That distinction is essential in transit.

In the authorisation message the interface is signalled by POS entry mode: in common scheme usage, `05` for contact chip, `07` for contactless chip, `91` for the legacy contactless magnetic-stripe mode. Issuers risk-score off that field, which is why an issuer can decline contactless while approving contact for the same card.

Why the tap limit exists, and what it actually protects

The limit is a **lost-and-stolen control**. It is not a countermeasure against skimming, eavesdropping or cloning, and it has no relationship to the radio.

Its legal basis in Europe and, by onshoring, in the United Kingdom is Article 11 of Commission Delegated Regulation (EU) 2018/389, the SCA-RTS. The Article permits a payment service provider not to apply strong customer authentication for a contactless transaction at the point of sale provided the individual amount does not exceed EUR 50, and either the cumulative amount since the last application of strong customer authentication does not exceed EUR 150, or the number of consecutive such transactions since then does not exceed five.

The UK version raised the figures without changing the shape:

Date	Single transaction	Cumulative
2007	£10	—
2010	£15	—
2012	£20	—
2015	£30	—
1 April 2020	£45	set under SCA-RTS Article 11
15 October 2021	£100	£300, or five consecutive transactions
19 March 2026	set by the firm	set by the firm

The 2020 increase was accelerated as part of the industry's response to Covid-19; the 2021 figures then held for four and a half years. On 14 March 2025 the FCA published an engagement paper on contactless limits, with responses due by 9 May 2025; on 19 December 2025 it announced that it would remove the prescribed limits and allow firms with strong fraud controls to set their own, encouraging them to let customers set personal limits or switch contactless off entirely; the change took effect on 19 March 2026. As of mid-2026 the major UK issuers had publicly retained £100. Consumer protection was explicitly unchanged: unauthorised transactions on a lost or stolen card must still be reimbursed.

Enforcement happens at three independent points, and a practitioner needs to know which one is refusing. The **terminal** refuses using the kernel data objects above: over the CVM required limit it demands verification, over the transaction limit it refuses the interface entirely, producing "please

insert your card" with no network round trip. The **card** refuses using issuer-personalised counters that force a contact transaction once the number or cumulative amount of contactless transactions passes a threshold; this implements the £300 bucket at chip level, and is why the bucket resets when you insert the card and type a PIN. The **issuer host** refuses using velocity rules against its own record of the card, and is the only one of the three that sees transactions at other merchants in real time.

The fraud numbers put the control in proportion. UK Finance reported contactless fraud losses of £41,5 million in 2023, a 19 % increase on 2022, against total unauthorised fraud of £708,7 million — about 5,9 %. Its Annual Fraud Report of 15 June 2026, covering 2025, reported total fraud losses of £1,28 billion, of which contactless fraud was £46,8 million, up 8 %; lost and stolen card fraud £109,8 million, down 2 %; and remote purchase fraud — card-not-present, which no tap limit touches — £423,5 million, up 3 %. The tap limit governs a category roughly one ninth the size of the one it cannot govern at all.

Relay attacks

The design flaw, if it is one, is implicit in the standard: ISO/IEC 14443 has no concept of distance, only of *field strength sufficient to operate*, which the attacker supplies himself. Nothing in the protocol binds the card's physical location to anything.

A relay therefore requires no cryptanalysis. The attacker builds two devices — a fake card presented to a genuine reader, a fake reader presented to a genuine card — and forwards APDUs between them over any channel with adequate bandwidth. Gerhard Hancke demonstrated a practical relay against ISO 14443 proximity cards in 2005; Saar Drimer and Steven Murdoch published the case for distance bounding against smartcard relays in 2007; relays using unmodified NFC handsets followed. The problem was solved in principle much earlier, by Brands and Chaum's distance-bounding protocol in 1993 and Hancke and Kuhn's practical variant in 2005, both timing single bits at the physical layer and deriving an upper bound on distance from the speed of light.

Two things are frequently conflated. **Eavesdropping** is passive interception of an existing transaction, and it reaches much further than the operating range because it depends on radiated leakage rather than coupling. Hancke's experiments at RFIDsec'08, using a wide-range receiver and a commercial antenna kit, recovered the reader-to-card (forward) channel of ISO 14443 Type A at 5 m in an entrance hall and 4 m in a corridor, and the card-to-reader (backward) channel at 1 m; for Type B, the forward channel at 3 m and the backward channel at up to 4 m. Eavesdropping yields the transaction as it happened, not a reusable credential, because the cryptogram is bound to that amount, that unpredictable number and that ATC. **Relaying** is active and creates a new transaction. It is the dangerous one.

The Relay Resistance Protocol

The industry's answer is a timing bound. Mastercard's Relay Resistance Protocol has been in EMV Book C-2 since 2016, in that book's sections 3.10, 5.3 and 6.6. The same protocol is specified in publicly available form in the European Card Payment Association's CPACE Terminal Kernel Specification v1.0 of 12 July 2018, from which the following field-level detail is taken.

The terminal issues an **EXCHANGE RELAY RESISTANCE DATA** command immediately after application selection:

Field	Value
CLA	`80`
INS	`EA`
P1	`00`

Field	Value
P2	`00`
Lc	`04`
Data	Terminal Relay Resistance Entropy, 4 bytes, binary
Le	`00`

The card returns four data elements: **Device Relay Resistance Entropy** (4 bytes, binary), **Min Time For Processing Relay Resistance APDU** (2 bytes), **Max Time For Processing Relay Resistance APDU** (2 bytes) and **Device Estimated Transmission Time For Relay Resistance R-APDU** (2 bytes). All three timing values are expressed in units of hundreds of microseconds.

The terminal starts a timer when it sends the command, stops it when the response arrives, and subtracts its own known transmission overheads to isolate the card's contribution:

Measured Relay Resistance Time = Timer - Terminal Transmission Time For Relay Resistance Command - Expected Minimum Transmission Time For Relay Resistance Response

floored at zero, where the expected minimum is the lesser of the card's declared estimate and the terminal's own configured response transmission time. The CPACE defaults for the terminal's transmission times are `0012` and `0018` in hundreds of microseconds: 1,8 ms for the command, 2,4 ms for the response.

Three checks then run. If the measured time is below the card's declared minimum by more than the configured tolerance, the transaction fails outright — a card that answers *too fast* is as suspicious as one that answers too slowly, because it suggests a pre-computed answer. If the measured time exceeds the declared maximum plus tolerance, a Terminal Verification Results bit is set and the protocol may be retried, the relay resistance counter permitting a maximum of two attempts. And if the card's estimated transmission time and the terminal's own differ by more than a configured percentage, or the measured time exceeds the declared minimum by more than a configured difference limit, further TVR bits are set for the issuer to act on.

The terminal-side configuration lives in Kernel 2 proprietary tags:

Tag	Kernel 2 data object
`DF8132`	Minimum Relay Resistance Grace Period
`DF8133`	Maximum Relay Resistance Grace Period
`DF8134`	Terminal Expected Transmission Time For Relay Resistance C-APDU
`DF8135`	Terminal Expected Transmission Time For Relay Resistance R-APDU
`DF8136`	Relay Resistance Accuracy Threshold
`DF8137`	Relay Resistance Transmission Time Mismatch Threshold

Crucially, both entropies are subsequently included in the data over which the Application Cryptogram is computed. A relay cannot pre-compute the exchange, and cannot substitute its own timing values, because the values it saw are bound into the cryptogram the issuer will verify.

How well it works

Honestly: less well than the design suggests, because the measurement is taken at the wrong layer.

Researchers at the Universities of Birmingham and Surrey, working under the NCSC-funded TimeTrust project, measured the RRP exchange on a test card and found a mean round-trip of about 53 000 µs on the first exchange with a standard deviation near 13 170 µs, and about 40 100 µs on subsequent exchanges. Against a card-declared maximum of 79,62 ms they relayed successfully in 67,79 to 77,05 ms. The jitter inherent in an application-layer measurement forces the acceptance window so wide that a competent relay fits inside it. Their conclusion was that distance bounding is more reliable at Level 1, at the ISO/IEC 14443-A layer where timing is locked to the carrier, than at Level 3 where it is at the mercy of APDU processing; they proposed a Level 1 relay protection protocol, L1RP, and verified it formally in Tamarin.

The same group showed why relay protection matters beyond plastic: a non-standard sequence of bytes preceding the standard ISO 14443-A wake-up command would convince Apple Pay that it was speaking to a transport reader, unlocking the Express Transit path. Combined with a Visa credential and a relay, they published video of £1 000 taken from a locked iPhone. Mastercard on Apple Pay and Visa on Samsung Pay were not affected.

The defences that remain, in descending order of effectiveness: the CVM required limit, because a relay attacker still cannot produce the PIN; issuer velocity rules, because a relay is a physically slow crime; the card's own offline counters; and RRP, which raises the bar without closing the door.

Why contactless is not less secure than chip and PIN in the way people assume

The popular model is that chip and PIN has two locks and contactless has one. The accurate model is that both have exactly the same lock on the *card*; the difference is entirely in the *cardholder* lock and in offline authorisation policy.

The Application Cryptogram at tag `9F26` is a MAC computed over the transaction data, the AIP and the ATC using a session key derived from the card's issuer master key. It is the same computation on both interfaces, and it is bound to one amount, one currency, one unpredictable number and one value of a counter that never repeats. It cannot be replayed for a different amount, at a different terminal, or twice.

So the "digital pickpocket" scenario — a criminal walking through a crowd with a reader in a bag — fails structurally. What can be read without interaction is limited, and what would be needed to monetise it is absent: no CVV2, no pair usable for card-not-present, no cryptogram usable for anything but the single transaction it was generated for. To take money the criminal must run a transaction against a real acquirer, which means being a merchant, which means being findable.

The real weaknesses, when they have appeared, have been in the layer that decides *whether verification happened*, and they are worth stating precisely because they are the seam.

In 2014, Emms, Arief, Freitas, Hannon and van Moorsel of Newcastle University presented at ACM CCS an attack showing that Visa contactless cards would approve foreign-currency transactions for any amount up to €999 999,99 without the cardholder's PIN. The UK limit then in force was £20. The flaw was not in the radio or the cryptogram: the limit check was performed against the terminal's own currency and no check existed for others.

In 2021, Basin, Sasse and Toro-Pozo of ETH Zurich presented "The EMV Standard: Break, Fix, Verify" at IEEE Security and Privacy, describing a PIN bypass on Visa contactless. The Card Transaction Qualifiers, tag `9F6C`, tells the terminal what verification the card expects, and in the affected configurations

it is neither authenticated nor cryptographically protected against modification. A man-in-the-middle clears bit 8 of the first byte, removing the online-PIN requirement, and sets bit 8 of the second byte, asserting that on-device cardholder verification was performed; in their published log `8200` became `0280`. The terminal then believes the cardholder has already been verified. They demonstrated it with an Android proof of concept in a real shop for approximately $190. Mastercard was not vulnerable to the same manipulation, because there the equivalent capability is signalled in the second bit of the first byte of the Application Interchange Profile, and the AIP *is* covered by the cryptogram.

The lesson generalises. Any data element that steers cardholder verification but is not covered by the Application Cryptogram is a candidate vulnerability, whatever the interface. Contactless made the attack cheaper to mount, but it did not create the seam. The seam was in the data model.

Transit and aggregated fares

Transit breaks the standard model for one reason: latency. The US Payments Forum's technical solution for transit contactless open payments states the requirement as a go/no-go customer entry prompt within a sub-second window, typically no more than **500 milliseconds** from a valid tap. No online authorisation meets that budget reliably, at every gate, at 08:15.

So the gate runs the transaction offline and pushes everything else downstream. The architecture has four named components.

Offline data authentication at the terminal. The gate performs dynamic ODA — fDDA or CDA — using the card's public key certificate chain, verified against the scheme CA public key held in the reader. This defeats counterfeit cards, skimming and man-in-the-middle at the gate itself, with no issuer connection. It is also where the energy budget bites: the card must complete an asymmetric operation inside the tap.

A deny list. Every gate holds a locally cached list of blocked cards and checks the credential against it before opening. The forum's framing is that the ability to hot-list a card quickly, so it is declined at every point of entry, is critical to preventing recurring fraud.

Deferred authorisation. Formally, a request that occurs when a merchant captures transaction information while connectivity is interrupted and holds it until connectivity is restored. Transit uses the mechanism deliberately rather than by accident.

Aggregation. Multiple taps combined into a single authorisation for a single amount. This is what makes fare capping possible: the back office cannot know that today's fifth journey has hit the daily cap until the fifth journey exists, so it cannot price any journey at the moment of travel.

The financial consequence has a name: **first tap risk**, the exposure created when a deferred authorisation is eventually declined for insufficient funds after the passenger has travelled. **Second tap risk** is the follow-on, the same card used again before it reaches the deny list. Neither sits with the issuer or the passenger; both sit with the operator, which is why transit operators are unusually exercised about deny-list latency.

Two data-level points a practitioner must get right. The Amount Authorised in tag `9F02` as sent by the gate will not equal the amount finally cleared, because the cleared amount is the aggregate. And an issuer must validate the cryptogram using only the data carried in the authorisation message's ICC data field — in ISO 8583 terms, Field 55 — and must not cross-check it against other fields whose values legitimately differ under aggregation. Issuers that get this wrong produce mysterious declines that appear only on transit.

Regulation accommodates this explicitly. Article 12 of the SCA-RTS exempts from strong customer authentication any electronic payment transaction initiated by the payer at an unattended payment terminal for the purpose of paying a transport fare or a parking fee. Without it, the cumulative counter in Article 11 would demand a PIN from every commuter twice a week, at a gate with no keypad.

London is the reference implementation. Contactless bank cards were accepted on London buses from 13 December 2012, with 2 586 journeys on the first day, and on the Tube and rail network from 16 September 2014. By December 2022 buses alone carried about 1,7 million contactless journeys a day, with more than 2,5 billion journeys across all modes since launch and credentials from over 180 countries presented at the gates.

For the passenger the visible artefacts follow directly. Charges appear grouped rather than per journey, and a day or more late, because aggregation runs after the travel day closes. The same card or device must be used to touch in and out, because the back office identifies a journey by matching two taps from the same credential — a card and the phone holding that card are, to the fare engine, two travellers. An incomplete journey draws a maximum fare because the back office cannot infer where you went. Caps are applied by the back office, never by the gate. And a device that runs flat mid-journey costs its owner a maximum fare, which is why Express Transit modes with a power reserve exist, and why those modes are where the interesting attacks live.

The sequence, end to end

The layers a single tap traverses, and the specification governing each:

Step	Governed by
Reader field present, card powers up	ISO/IEC 14443-2; EMV Contactless Interface Specification (Level 1)
Polling, REQA/WUPA or REQB/WUPB, anticollision, SELECT	ISO/IEC 14443-3
RATS/ATS, frame size and FWT negotiation, T=CL	ISO/IEC 14443-4
APDU command and response structure	ISO/IEC 7816-4
SELECT `2PAY.SYS.DDF01`, candidate list, kernel activation	EMV Contactless Book B (Entry Point)
EXCHANGE RELAY RESISTANCE DATA, where supported	EMV Contactless Book C-2
GET PROCESSING OPTIONS with PDOL, READ RECORD, fDDA/CDA	EMV Contactless Books C-1 to C-8
GENERATE AC, cryptogram returned	EMV Contactless Books C-1 to C-8
Authorisation request carrying the cryptogram	ISO 8583 or ISO 20022, per scheme

The last four rows are identical to the contact interface. Everything above them is radio. The money never notices the difference.

18.98 Common wrong ideas

Wrong: the card is powered by radio waves it picks out of the air. **Right:** at 13,56 MHz the exchange happens deep inside the near field, where energy passes by mutual inductance between two coils rather than being radiated, which is why the range is centimetres.

Wrong: the short operating range means nobody can listen in. **Right:** the range at which a card can be powered and transacted with is not the range at which it can be overheard, and published work has recovered the reader-to-card channel at several metres.

Wrong: the £100 limit is a technical property of the chip, the radio or the kernel. **Right:** nothing in any radio standard or EMV kernel knows what a pound is; the figure came from Article 11 of the SCA-RTS and since 19 March 2026 it is the firm's own choice.

Wrong: contactless is less secure than chip and PIN because it has one lock instead of two. **Right:** the Application Cryptogram is the identical computation on both interfaces; the whole difference is in cardholder verification and offline authorisation policy.

Wrong: contactless is exactly as secure, so there is no seam. **Right:** the seam is real and documented — data elements that steer whether verification happened, in some configurations not covered by the cryptogram, which is precisely the 2021 Visa PIN bypass.

Wrong: a criminal walking through a crowd with a reader in a bag can harvest usable card details. **Right:** there is no CVV2, nothing usable card-not-present, and a cryptogram bound to one amount, one unpredictable number and one counter value; to monetise anything the criminal must be a real, findable merchant.

Wrong: two cards in a wallet confuse the reader. **Right:** they cause a collision that Part 3's anticollision loop resolves deterministically, ending either in a clean read or a "present one card only" prompt.

Wrong: the ticket gate lets you through on trust and settles up afterwards. **Right:** before it opened it verified a dynamic signature from the card's own private key and obtained a full cryptogram; what is deferred is the authorisation, and the gap is first tap risk carried by the operator.

Wrong: the transit charge on your statement is a fare. **Right:** it is an aggregate computed after the travel day closes, which is why the Amount Authorised sent by the gate never equals the amount finally cleared.

Wrong: the Relay Resistance Protocol closes the relay problem. **Right:** measured at the APDU layer the jitter forces an acceptance window wide enough for a competent relay, which is why the same researchers argued for timing at the ISO/IEC 14443-A layer instead.

18.99 Chapter summary in 20 lines

1. A contactless card has no battery: a chip the size of a grain of rice and a loop of wire around the inside of the plastic is the whole machine.
2. The reader hums a magnetic note at 13,56 MHz all day, and the card's coil draws its power out of that field by mutual inductance.
3. The card cannot transmit, so it answers by repeatedly shorting a load across its own coil and letting the reader see the dips in its own field.
4. Field strength falls away savagely with distance, so the working range is centimetres — a consequence of the power budget rather than a privacy feature.
5. ISO/IEC 14443 defines the whole radio layer: Part 2 the field and modulation, Part 3 anticollision, Part 4 the block protocol universally called T=CL.
6. Above T=CL the card speaks ordinary ISO/IEC 7816-4 APDUs, so from the payment application's point of view the radio is only a transport.
7. EMVCo layers its own Level 1 requirements over that, and above them sit Book B's entry point and the eight contactless kernel books.
8. Which kernel wakes up determines what the limits are called and where they live, which is why "the contactless limit" is never one setting.
9. The Application Cryptogram is the same computation on both interfaces, bound to one amount, one currency, one unpredictable number and one value of a counter that never repeats.

10. So the radio changes nothing about proving the card is genuine, and the tap limit exists to answer an entirely different question.
11. That limit is a lost-and-stolen control built as a leaky bucket: a single-transaction cap plus a cumulative cap since the last strong authentication.
12. It is enforced at three independent points — reader data objects, counters inside the card application, and issuer velocity rules — and knowing which one refused is the practitioner's first question.
13. Its legal basis was Article 11 of the SCA-RTS; the UK figures climbed from £10 in 2007 to £100 and £300 in 2021, and the FCA removed the mandated caps on 19 March 2026.
14. The fraud figures keep it in proportion: contactless losses are a small fraction of remote purchase fraud, which no tap limit touches at all.
15. Eavesdropping is passive, reaches further than the operating range, and yields a transaction that cannot be replayed; relaying is active and creates a new one.
16. Nothing in ISO/IEC 14443 binds a card to a place, and a maximum Frame Waiting Time of nearly five seconds leaves ample room for a very long extension lead.
17. The industry's answer is the Relay Resistance Protocol: a timed exchange whose entropies are afterwards bound into the cryptogram, so the timing cannot be forged.
18. It helps less than it looks, because application-layer jitter widens the acceptance window enough for a competent relay to fit inside it.
19. Transit breaks the model on latency alone: the gate performs offline dynamic data authentication, checks a local deny list, opens, and defers both the authorisation and the price.
20. Aggregation is what makes fare capping possible and what puts first tap risk on the operator, and it is why your statement shows a single number, a day late, matching no fare on any poster.

Sources: ISO/IEC 14443 parts 2, 3 and 4; EMVCo Level 1 EMV Contactless Interface Specification v3.1 and Specification Bulletin 245; EMVCo Book C-8 v1.0; European Card Payment Association CPACE Terminal Kernel Specification v1.0; Commission Delegated Regulation (EU) 2018/389 Articles 11 and 12; FCA contactless limits engagement paper (2025) and press releases (2025); UK Finance contactless limit FAQs and Annual Fraud Report 2026; US Payments Forum papers on contactless limits (2020) and transit contactless open payments (2018); Transport for London press material; and the published research of Hancke (2005, 2008), Emms et al. (CCS 2014), Chothia et al. (FC 2015), Basin, Sasse and Toro-Pozo (IEEE S&P 2021), and Radu, Chothia, Newton, Boureanu and Chen on practical EMV relay protection.

CHAPTER 9

The Token in Your Phone

19.0 What this chapter gives you

1. You will be able to explain why the last four digits printed on a phone receipt differ from the last four on your plastic, and prove it to yourself by buying the same coffee twice.
2. You will be able to state exactly what a Payment Token is — 13 to 19 Luhn-valid digits from a real Token BIN Range, carrying its own expiry — and why it therefore crosses the existing network with nobody changing anything.
3. You will be able to say why a stolen token is close to worthless, and name the three separate walls that make it so.
4. You will be able to correct the claim that your card number never leaves the phone, and state the narrower claim that is actually true.
5. You will be able to say who mints the token and who owns the vault, and why many issuers cannot enumerate their own outstanding tokens without calling a network API.
6. You will be able to explain what the Payment Account Reference is for, why the industry had to invent it, and why it cannot itself initiate a payment.
7. You will be able to tell a secure element wallet from a host card emulation wallet by their availability and failure modes rather than by their cryptography.
8. You will be able to explain the green, yellow and red provisioning paths, and why yellow is where the whole design lives or dies.
9. You will be able to say why a reissued card leaves the token working and a lost phone leaves every standing order untouched.
10. You will be able to read a tokenised contactless trace, including why tag `9F6E` must be parsed kernel-aware or it will produce plausible nonsense.

The previous chapter left a phone held against a reader, an antenna pulling power out of the air, and a two-byte answer travelling back at 106 kilobits per second. It deliberately did not say what number was in that answer.

Here is the thing almost everybody gets wrong, including people who work in payments: when you tap your phone, your card number is not transmitted. Not encrypted, not obscured, not hashed. It is simply not there. The number the shop receives is a different number, sixteen digits long, beginning with the same digit as your card and looking in every respect like a card number, because it is one — it is just not yours. It was minted for that phone, it works only from that phone, and if somebody steals it they will find it worth almost nothing.

That substitution is called payment tokenisation, and it is the single largest change to card security since the chip. It is also the reason your bank can kill your Apple Pay card at four in the morning and

still let you buy petrol with the plastic on the way home. The plastic and the phone stopped being the same credential some time around 2014, and most cardholders have never been told.

This chapter is about that second credential: what it is, who makes it, what stops it being used anywhere else, and why a stolen one is close to worthless.

The plain version

Think about a hotel.

You check in and they give you a key card for room 412. It opens your room, it opens the pool, and it stops working on Friday. It is not the master key. It has its own serial number printed on the back, and if you lose it in a taxi the hotel does two things: it cancels that serial number, and it cuts you a new card. Room 412 does not change. The lock does not change. Nobody has to re-key the building.

Now suppose you want a second key so your brother can come and go. The hotel does not photocopy your card. It cuts a *new* card, with a *different* serial number, which opens room 412 and the pool but not the gym, and only between nine and five. Two cards, two serial numbers, one room, different powers.

Your bank account is room 412. Your plastic card is the first key. The card in your phone is the second key: a different number, cut for a specific purpose, cancellable on its own.

The number is genuinely different

Say your card is **4659 1234 5678 9011**, expiring 11/27. That number is embossed on the plastic, printed on your statement, and known to every shop you have ever typed it into.

When you add that card to your phone, your bank and the card network create a second number for you — say **4658 7712 3456 7892**, expiring 03/29. It is stored on a small, sealed chip inside the phone. It is not printed anywhere. You cannot read it out. The phone shows you the last four digits and nothing more.

Both numbers point at the same account and the same money. But they are not interchangeable, and this is the whole idea.

A worked example: £180 of headphones

You are in a shop in Manchester buying headphones for £180. You hold your phone to the reader and look at it, and your face unlocks the payment.

Three things happen in the two seconds that follow.

First, the phone sends **4658 7712 3456 7892** and the expiry 03/29. It does not send 4659 1234 5678 9011. The shop never learns that number. Neither does the shop's till software, nor its payment provider, nor the company that stores the day's takings.

Second, the phone sends a short one-time code — eight bytes of arithmetic, computed fresh for this tap, using a secret key that never leaves the sealed chip and a counter that ticks up by one every single time. Yesterday's code will not work today. This tap's code will not work at the next shop. It is the digital equivalent of a wax seal that can only be pressed once.

Third, the phone says: *the person holding this device proved who they were, on the device, just now.* That is why £180 goes through without a PIN pad, when a plain contactless card would have stopped dead at £100. The reader does not see your face. It simply receives a statement that a face was seen, from a device the bank has decided to trust.

Your bank receives all three, notices that 4658 7712 3456 7892 stands for account 4659 1234 5678 9011, checks the one-time code against its own copy of the maths, sees that verification was done on the device, and approves.

Now look at your receipt. The last four digits printed on it will be **7892**, not 9011. Most people assume this is a misprint or a strange rounding of their card number. It is neither. It is the token's last four digits, and it is the easiest way to prove to yourself, tonight, that your phone is not sending your card number: pay for the same coffee twice, once with the plastic and once with the phone, and compare the two receipts. They will end in different digits.

What a thief gets

Six months later the shop is hacked and somebody walks off with a year of transaction records, including **4658 7712 3456 7892** and 03/29.

They cannot use it. Three separate walls stand in the way.

The number is registered as working only for contactless taps from that one phone. Type it into a website and it is refused before any money is considered, because the network can see that this number is not permitted to be typed into a website. That restriction is baked in at the moment the number is created.

The number is worthless without the one-time code, and the code cannot be produced without the sealed chip in your phone. The thief has the envelope; the seal is in Manchester in your pocket.

And it does not identify you. It is not on your statement, your bank does not use it on the phone to you, and it cannot be traced back to 4659 1234 5678 9011 by anyone outside the network.

Compare that to what a thief gets from a stolen card number: a working credential, usable in dozens of channels, that stays valid until the bank reissues the plastic and posts it to you.

Losing the phone

You leave the phone on a train. You call the bank.

The bank cancels **4658 7712 3456 7892**. That is one row in one table. It takes seconds, needs no post, and costs nothing.

Your plastic card still works. Your card number has not changed. Every standing order, every subscription, every card-on-file at every shop that has your real number is untouched. The direct debit for your gym does not fail. You buy dinner on the way home with the card in your wallet.

This is the practical payoff, and it is the sentence to carry to dinner: **the number in your phone can be destroyed without destroying your card, because it was never your card.**

Where the plain version stops being true

The hotel key analogy is sound in outline, and wrong in four ways that matter.

The token is not what makes the transaction safe

A payment token is, on its own, just a number. It has no cryptography in it. If you place a token in a channel where nobody demands a cryptogram and nobody checks the restrictions, it spends exactly like a card number — that is the point of the design, since the whole scheme was built to travel over infrastructure that already existed and could not be changed.

What makes the tap safe is the combination: a token *plus* a per-transaction cryptogram *plus* restrictions enforced by whoever is doing the checking. Take away any one of the three and you have a card number with an unusual value. Merchant card-on-file tokens, which are the same EMVCo mechanism, routinely operate with no per-transaction cryptogram at all; they rely on merchant and channel restriction alone.

So the correct statement is not "tokens are safe". It is "tokens shrink the blast radius of a breach, and cryptograms stop a stolen credential being replayed, and the two are separate mechanisms that people habitually merge into one word".

Your card number does leave the phone — once

Apple's and Google's public claims are carefully worded, and the careful wording is doing work.

When you add a card, you type the real number into the phone, or the phone reads it with the camera, or your bank's app pushes it across. That number is then encrypted and sent to the card network and your bank, because they have to know which account you are asking to tokenise. It leaves the device. It travels.

What is true is narrower and still meaningful: the wallet provider cannot decrypt the token it receives back; the token is written into the phone's sealed chip; and the wallet's servers do not store it. Apple states that the Device Account Number is "never stored on Apple Pay servers or backed up to iCloud". That is a real property. It is not the same as "Apple never sees your card details", which is what people repeat.

There is also a residue. The wallet keeps an opaque identifier derived from the funding card so that it can tell two tokens on the same card apart from two tokens on different cards. Apple exposes it to apps as `primaryAccountIdentifier`. It is not the card number, but it is a persistent handle on the card number, and it exists because the ecosystem could not function without one.

The issuer is not really in charge; the network is

The analogy implies the hotel cuts the keys. In practice, for the overwhelming majority of device tokens, the Token Service Provider is Visa or Mastercard, not your bank. They own the vault. They own the mapping table. They decide the token's BIN range, they mint the value, they hold the keys used to validate cryptograms, and they perform the substitution back to your real number before the authorisation reaches your bank's systems.

Your bank's role at provisioning time is to answer one question — approve, decline, or ask for more proof — and its role afterwards is to consume a data feed. Many issuers cannot enumerate their own outstanding tokens without calling a network API. When the network's token service is degraded, tokens stop working even though the bank is perfectly healthy and the plastic is fine. This is a real operational dependency, and it is not visible from the cardholder's side of the counter.

"Device token" describes a shrinking share of tokens

The chapter title says *phone*, and phones are where tokenisation is most visible. But EMV payment tokens are also issued in enormous numbers to merchants for stored cards, to wallets for e-commerce, and to acquirers on behalf of merchants who never asked. Those tokens live in databases, not secure elements. There is no biometric, no secure chip, no per-tap cryptogram of the kind described above.

The consequence shows up in a place nobody anticipated. Once the same person carried five different numbers at five different merchants, the systems that needed to know "these are all the same account" — fraud scoring, loyalty, anti-money-laundering, transit fare capping — broke. The industry's answer

was to invent yet another identifier, the Payment Account Reference, whose entire job is to undo just enough of the separation to make those systems work again. That is a strange thing to have to build, and its existence tells you the analogy was too clean.

The technical version

Vocabulary, exactly

The governing document is the **EMV Payment Tokenisation Specification – Technical Framework**, published by EMVCo, royalty-free and publicly available. Version 2.4 was published on 9 July 2026, superseding v2.3; v1.0 dates from 2014. A companion document, *EMV Payment Tokenisation – A Guide to Use Cases*, carries the worked scenarios.

Note the spelling and the scope. EMVCo's document says *tokenisation*, and it defines a **Payment Token** — not a DPAN. The terms this chapter's title uses are industry vocabulary that grew up alongside the specification:

Term	Expansion in common use	Status
FPAN	Funding PAN	Industry usage; not an EMVCo term
DPAN	Device PAN, or device primary account number, or (in Google's documentation) dynamic primary account number	Industry usage; expansions disagree
Payment Token	—	EMVCo term, covers device *and* non-device tokens

Apple's PassKit framework uses both abbreviations openly: `PKSecureElementPass` exposes `deviceAccountIdentifier` (documented as an opaque identifier of the DPAN), `deviceAccountNumberSuffix` (the last four digits of the DPAN), and `primaryAccountIdentifier` (an opaque identifier of the FPAN). Google's device tokenisation documentation writes "dynamic primary account number (DPAN) or device token". When you see DPAN in a scheme manual, check which expansion that manual means; they are not always the same thing.

The roles are stable across versions:

Role	Definition
Token Service Provider (TSP)	Operates the Token Vault and related processing; maps Payment Tokens and token expiry dates to PANs and PAN expiry dates; issues tokens to Token Requestors; performs de-tokenisation; sets aside licensed ISO BINs as Token BINs
Token Requestor	The entity that submits Token Requests. Apple, Google, Samsung, a merchant, a wallet, an acquirer
Token Vault	The repository holding token-to-PAN mappings and token attributes
BIN Controller	The entity that controls a BIN range, registered with EMVCo for PAR purposes

EMVCo runs two registration programmes rather than a certification programme. The **TSP Registration Programme** assigns a three-digit **TSP Code** so that every TSP issuing payment tokens is globally identifiable; as of writing the fee is USD 11,500 for initial registration and USD 2,850 per calendar year thereafter. The **BIN Controller Registration Programme** assigns a **BIN Controller ID** (**BCID**), which becomes the first four characters of every PAR that controller generates. EMVCo is explicit that it "does not evaluate, approve or otherwise endorse TSPs"; it maintains a list of codes. There is no EMVCo tokenisation test programme, and EMVCo says plainly that one would be impractical.

The token itself

Element	Specification
Payment Token	A surrogate value for a PAN: a 13 to 19 digit numeric value that must pass basic account-number validation rules including the Luhn check digit
Token BIN Range	A unique identifier consisting of the leading 6 to 12 digits of the Token BIN; carried in BIN routing tables
Token Expiry Date	The token's own expiry, mapped by the TSP to the PAN expiry date; need not equal it
Token Requestor ID (TRID)	An 11-digit numeric value identifying each unique combination of token requestor and token domain
Token Cryptogram	"A cryptogram generated using the Payment Token and additional transaction data to create a transaction-unique value." Calculation and format vary by use case
Payment Account Reference (PAR)	A fixed-length 29-character uppercase alphanumeric value: 4 characters of BCID assigned by EMVCo, plus 25 unique characters

Three properties of that last row are load-bearing and frequently misstated. PAR has a one-to-one relationship with the PAN and a one-to-many relationship with the payment tokens issued against it. It cannot be reverse-engineered to obtain either a PAN or a payment token. And it cannot itself initiate a financial transaction. It is an identifier, not a credential.

The Token BIN Range point deserves emphasis for anyone building routing: token BINs are real ISO BINs, carved out of the issuer's or the TSP's allocation and published into routing tables. That is precisely why a token traverses the existing network without any acquirer changing anything. It is also why a badly maintained BIN table will route a token to the wrong place and produce a decline that looks like a network fault.

Where the token travels in the message

In ISO 8583, the payment token occupies the primary account number field, **DE 2**, and the token expiry date occupies **DE 14**. Chip data travels in **DE 55**. Point-of-service entry mode is **DE 22**; Visa's own acquirer implementation guidance specifies DE 22 = `07` for a qVSDC contactless chip transaction and `91` for a contactless magnetic-stripe (MSD) transaction.

Payment Account Reference is carried in EMV tag `9F24` within DE 55 for chip transactions. For non-chip transactions the Secure Technology Alliance's primer places it in data element 56 of the authorisation response. Treat the tag `9F24` path as the reliable one and confirm the DE 56 behaviour against your acquirer's own manual, because it varies.

The token-specific attributes that the issuer receives — token requestor ID, token assurance data, wallet identifiers, the last four digits of the underlying PAN — travel in scheme-private data elements whose numbering differs between networks and which are specified in each scheme's own message manual, not in any EMVCo document. Do not carry a field number learned on Visa across to Mastercard. This is the single most common source of integration errors in token projects.

In the ordinary model, the network de-tokenises before the authorisation reaches the issuer's host: the issuer sees the FPAN in DE 2, plus token metadata alongside. Some issuers host their own TSP and receive the token itself. Establish which model you are in before writing a single line of authorisation logic.

The chip data a phone actually presents

A tokenised contactless tap is an ordinary EMV contactless transaction whose PAN happens to be a token. The tags below are the base EMV set, defined in the EMV ICC Specifications for Payment Systems, Book 3, Annex A.

Tag	Name	What it holds on a phone
5A	Application Primary Account Number (PAN)	The token, not the funding PAN
57	Track 2 Equivalent Data	Token, field separator, token expiry, service code, discretionary data
5F24	Application Expiration Date	The token's expiry
82	Application Interchange Profile	Card capabilities
95	Terminal Verification Results	Terminal's view of what happened
9A	Transaction Date	
9C	Transaction Type	First two digits of the ISO 8583 processing code
9F02	Amount, Authorised (Numeric)	
9F10	Issuer Application Data	Proprietary data for the issuer, including the card verification results
9F1A	Terminal Country Code	ISO 3166
9F26	Application Cryptogram	The eight bytes produced in response to GENERATE AC
9F27	Cryptogram Information Data	Cryptogram type and required terminal actions
9F34	Cardholder Verification Method Results	Three bytes of testimony, as chapter 17 explained
9F36	Application Transaction Counter (ATC)	Increments every transaction
9F37	Unpredictable Number	Terminal-supplied variability
5F2A	Transaction Currency Code	ISO 4217

Two kernel-specific objects appear in Visa qVSDC flows and are not in Book 3 Annex A:

Tag	Name	Length
9F66	Terminal Transaction Qualifiers (TTQ)	4 bytes
9F6C	Card Transaction Qualifiers (CTQ)	2 bytes

And then there is `9F6E`, which is a trap.

Scheme	Meaning of `9F6E`	Length
Visa (qVSDC)	Form Factor Indicator (FFI)	4 bytes, fixed
Mastercard (M/Chip)	Third Party Data	5 to 32 bytes, variable

The same tag number carries two entirely different structures depending on which kernel produced the trace. A Visa FFI of `20700000` decodes as version `001`, form factor "standard card", with signature panel, hologram and CVV2 capability set. Feed those four bytes to a Mastercard Third Party Data parser and you will get plausible nonsense. Feed a seven-byte Mastercard value such as `06430000303000` to a Visa FFI decoder and it will either reject the length or truncate it. If your terminal estate handles both schemes, your TLV parser must be kernel-aware, and any documentation that lists `9F6E` with a single name is incomplete.

The Form Factor Indicator is where "this was a phone, not a card" is stated in the chip data, which is why it matters to issuers doing risk scoring on tokenised taps.

The secure element

Apple's implementation is the best-documented and is worth stating precisely, because it is also the most misdescribed.

The Secure Element is, in Apple's words, "an industry-standard, certified chip running the Java Card platform, which is compliant with financial industry requirements". It is "certified in accordance with the EMVCo Security Evaluation process", for which EMVCo issues unique IC and platform certificates, and it is separately "certified based on the Common Criteria standard". The Device Account Number lives there, in a payment applet, and Apple's servers manage those applets without being able to read the number.

The NFC controller sits between the application processor, the Secure Element and the reader, routing frames. Below it is ordinary contactless plumbing: ISO/IEC 14443 Type A or Type B at the radio layer, ISO-DEP (ISO/IEC 14443-4) for the transport, and application protocol data units per ISO/IEC 7816-4 above that. Applications are addressed by Application Identifier — up to 16 bytes, consisting of a five-byte Registered Application Provider Identifier issued by the ISO/IEC 7816-5 registration authority followed by a proprietary extension. `A000000003` is Visa and `A000000004` is Mastercard, so the credit and debit applications your phone advertises are `A0000000031010` and `A0000000041010` respectively — the same identifiers a plastic card presents. From the reader's point of view, a phone is a card.

Authorisation on Apple devices is a two-chip conversation. The Secure Enclave and the Secure Element communicate over a serial interface using a shared secret established at runtime through Elliptic Curve Diffie-Hellman key agreement. When you authenticate, "the Secure Enclave then sends signed data about the type of authentication and details about the type of transaction (contactless or e-commerce) to the Secure Element, tied to an Authorization Random (AR) value". The AR is generated in the Secure Enclave when the first card is provisioned and persists while Apple Pay is enabled, protected by Secure Enclave encryption and an anti-rollback mechanism. A payment "can be made only after the Secure Element receives authorization from the Secure Enclave".

The cryptogram itself is "computed using a transaction counter and a key", with the counter incremented for each transaction, and it incorporates the terminal's Unpredictable Number for NFC transactions or an Apple Pay server anti-replay value for in-app and web transactions. That last distinction

is the answer to a question practitioners ask constantly: an in-app tokenised payment has no terminal and therefore no terminal-supplied nonce, so the freshness has to come from somewhere else.

Host card emulation

Android took a different route. From **Android 4.4** onwards the platform supports host-based card emulation, in which "the data is routed directly to the host CPU instead of being routed to a secure element". A `HostApduService` receives command APDUs through `processCommandApdu()` and returns response APDUs; Android's implementation supports a single logical channel, so the exchange is strictly single-threaded and half-duplex. AIDs are declared in groups, and Android guarantees that all AIDs in a group route to your service or none do.

The security consequence is direct. Without a secure element, the payment keys are in ordinary application memory on a device that may be rooted. The industry's answer is not to pretend otherwise but to make the keys nearly worthless: **limited-use keys**, provisioned over the network and bounded by thresholds — a time-to-live, a maximum number of transactions, a cumulative transaction amount, or a combination — and replenished from the cloud. A key that expires after a handful of taps or a few days is a poor prize.

Around this sits EMVCo's **EMV Mobile Payment: Software-Based Mobile Payment Security Requirements** and the associated **SBMP Security Evaluation Process**, introduced in 2018. It evaluates software development kits, trusted execution environments, consumer device cardholder verification methods including biometrics and authenticators, attestation mechanisms and software protection tools. EMVCo issued its hundredth SBMP security evaluation certificate on 29 November 2022, with nine accredited laboratories performing the evaluations.

The practitioner-facing difference between the two architectures is not really cryptographic. It is availability:

	Secure element	Host card emulation
Key storage	Certified hardware, Java Card applet	Application memory, protected by software and platform features
Key lifetime	Long-lived, provisioned once	Limited-use, replenished
Behaviour offline	Works indefinitely	Works until keys are exhausted, then fails
Failure mode	Hardware failure	"Unable to pay, please connect to the internet"
Certification	EMVCo Security Evaluation, Common Criteria	EMVCo SBMP Security Evaluation

A user who leaves an HCE wallet in a drawer for a month and then tries to tap in a car park with no signal will be declined, and the decline will be correct.

Provisioning: green, yellow, red

Provisioning is the security-critical moment in the whole design. A criminal who successfully provisions a stolen card into their own phone has manufactured a fresh, fully working credential with strong cryptography protecting *them*.

Apple's flow uses three server-side calls — Required Fields, Check Card, and Link and Provision — with sessions protected by TLS 1.2 or later. The device sends the card details together with information

about the account and the device, and the issuer or its network returns an encrypted Device Account Number along with the key material used to generate the per-transaction codes.

For provisioning initiated inside a bank's own app, Apple's PassKit provides `PKAddPaymentPassViewController`, configured with `PKAddPaymentPassRequestConfiguration`, which returns a `PKAddPaymentPassRequest` carrying three fields:

Field	Contents
`encryptedPassData`	Payload encrypted by the issuer host under a key derived from Apple's public certificates and a generated ephemeral private key
`ephemeralPublicKey`	The public half of a key pair generated for this provisioning attempt only
`activationData`	A cryptographic one-time value generated by the issuer host per the network's specification

`EV_ECC_V2` is the mandatory encryption scheme.

On Android, the equivalent path is Google's TSP integration with push provisioning from the issuer app. Where a challenge is required, app-to-app identification and verification hands control to the issuer's app via an Android Intent, with a base64-encoded JSON payload in `Intent.EXTRA_TEXT`; the issuer app authenticates the cardholder and returns `BANKING_APP_ACTIVATION_RESPONSE` with a value of `approved`, `declined` or `failure`, optionally accompanied by `BANKING_APP_ACTIVATION_CODE`. Google's documentation is explicit that the issuer app must verify the caller using `Activity.getCallingPackage()` and confirm it is Google Wallet, `com.google.android.gms`. Skip that check and you have built an activation oracle for anybody who can install an app.

The issuer's decision is described everywhere in the industry by colour:

Path	Meaning	Typical use
Green	"Token creation has been approved. The token is created in an active state and is ready to be used for payment."	Push provisioning from an authenticated banking session; trusted devices
Yellow	"Cardholder is subject to additional authentication requirements (called stepped-up authentication)."	Most manual card entry
Red	"Token request is declined. No token is created."	Reserved for unacceptable risk; in Google's guidance, "typically a very small fraction of attempts"

Yellow is where the design lives or dies, because a token stuck awaiting activation is a card the customer believes they have added and cannot use. The resolution methods in general use are: a one-time code by SMS; a one-time code by email; a call to the issuer's contact centre; login to the issuer's mobile banking app, including the app-to-app flow above; and, for Visa customers, web-based secure authentication. Google's published guidance is that "well thought out and high performing ID&V processes yield token activation rates of +90% on Google Pay", and that most issuers route manual provisioning attempts to yellow except on a trusted device. Issuers receive account and device risk scores from the TSP and are encouraged to override them when their own data says the user is legitimate.

The specification's own vocabulary for what yellow accomplishes is **identification and verification** (**ID&V**), with five defined cases: No ID&V Performed; Account Verification; Token Service Provider Assurance; Token Service Provider Assurance with Requestor Data; and Card Issuer Verification of the Cardholder.

One historical correction that catches people reading older material. Version 1.0 of the framework defined a two-digit **Token Assurance Level**, where "a value of 00 would indicate no ID&V and 99 would indicate the highest level of assurance". Version 2.0 replaced that concept with **Token Assurance Method**, based on the type and outcome of the ID&V performed, the entity that performed it, the domain in which the token is used, and supporting token assurance data. If a vendor document still talks about assurance *levels* as though they were current, that document has not been revised since 2016 and everything else in it should be treated with suspicion.

Mastercard's Digital Enablement Service exposes the mechanics under its own names: a `tokenUniqueReference` identifying the token for its lifetime, a `panUniqueReference` for the funding account, and operations including `GET /{id}/token/activate`, `GET /{id}/token/resendactivationcode` and `GET /{id}/token/activationmethods`. A token awaiting the yellow path carries a provisioning status of "Awaiting Activation".

Domain restriction, precisely

EMVCo defines **Token Domain Restriction Controls** as "a set of parameters that are applied during Token Processing to constrain a Payment Token to the permitted usage scenarios". Three inputs do the constraining:

Token Requestor ID. Does this token belong to this token requestor? A token minted for Apple Pay presented by a different requestor is refused.

POS Entry Mode. The specification calls for "the use of POS Entry Mode values that are carried in the POS Entry Mode Code field to limit the use of Tokens to only those POS Entry Modes agreed to during Token Requestor Registration". This is what prevents a contactless device token being keyed into a web form: the entry mode does not match the registration.

Merchant information. "Merchant-related data elements, such as the Card Acceptor ID in a combination with Acquirer-identifying data elements SHOULD be used to limit the use of a Payment Token." This is the mechanism behind merchant-scoped card-on-file tokens.

To these the framework adds the requirement for a valid token cryptogram in domains where one is expected. Validation may be performed by the TSP or delegated to the issuer; either way, a cryptogram that fails or a transaction that violates the domain controls is refused before the account is even considered.

The critical property, and the one the plain version hides, is that **domain restriction is enforced by whoever is checking, and it is only as strong as that party's implementation.**

The clearest public demonstration of this came in 2021. Andreea-Ina Radu, Tom Chothia, Christopher J. P. Newton, Ioana Boureanu and Liqun Chen, of the University of Birmingham and the University of Surrey, published *Practical EMV Relay Protection*, showing that an iPhone with a Visa card set up in Express Transit mode could be made to authorise **£1,000** while locked. Transport for London ticket gates broadcast a non-standard byte sequence before the standard ISO/IEC 14443 wake-up command — the researchers called them the "magic bytes" and deliberately did not publish their value. Replaying that sequence convinced the phone it was at a transit gate, where Express Transit deliberately suppresses the biometric prompt. The attack then set bits in the Terminal Transaction Qualifiers sent in the GET

PROCESSING OPTIONS command and set the "consumer device CVM performed" bit in the Card Transaction Qualifiers, so that the shop reader believed a cardholder verification had happened that had not. Relay success improved from roughly 40 per cent to effectively complete once READ RECORD responses were cached. The combination was specific: it did not affect Mastercard on iPhone, or Visa on Samsung Pay. Apple and Visa have since addressed it.

The lesson is not that tokenisation failed. The token worked exactly as designed. What failed was one exemption inside one domain control, on one platform, for one scheme. That is the shape of every real tokenisation failure: not broken cryptography, but a restriction that somebody decided to relax.

Lifecycle: why the token dies and the plastic lives

Tokens carry state independent of the account. Mastercard's states are the ones most commonly encountered:

State	Meaning	Reversible
Active	Provisioned, activated, usable	—
Suspended	Temporarily blocked	Yes, by unsuspend
Deactivated	Permanently ended	No

Four lifecycle events are worth committing to memory because they generate most of the support tickets.

The card is reissued. Your plastic is lost and a new card arrives with a new PAN and expiry. The token in your phone keeps working, because the TSP updates the token-to-PAN mapping in the vault. You do not re-add the card. This surprises people and is the correct behaviour: EMVCo's own material describes tokenisation as reducing "the need for card replacements" and enabling issuers "to control and replace payment tokens for specific merchants, devices or transaction types, often without any interaction from the cardholder". Note the corollary — if the card was reissued *because of* fraud, the automatic mapping update means the token survives too, unless the issuer suppresses it. Suppression is a configuration choice, and getting it wrong in either direction produces either angry customers or continued fraud.

The device is lost. The token is suspended or deactivated. The account is untouched. Apple's implementation lets Lost Mode in Find My suspend the ability to pay, or a remote erase remove it, and Apple notes that the issuer suspends the cards "even if your device is offline and not connected to a cellular or Wi-Fi network" — because suspension happens in the vault, not on the phone.

The token is compromised. A merchant breach exposes tokens. The TSP deactivates the affected tokens and the requestor provisions replacements. No plastic is reissued, no card numbers change, and nothing else the cardholder has set up is disturbed. This is the single largest operational saving in the whole design, and it is the reason issuers accepted tokenisation despite the loss of control described earlier.

The customer deletes the card from the wallet. The token is deactivated and cannot be revived; adding the card again mints a new one, with a new value, a new expiry and a new last four digits.

What it is worth

Some figures, all dated, all from the schemes themselves.

Visa announced its ten billionth token on 4 June 2024, ten years after launch, stating that tokenisation "can reduce the rate of fraud by up to 60%", that it had saved USD 650 million in fraud in the preceding

twelve months, that 29 per cent of all Visa transactions used tokens as of April 2024, and that tokens contributed a six basis-point increase in global payment approval rates.

Mastercard reported in June 2025 that nearly half of its European e-commerce transactions were tokenised, up by more than a third year on year, and has stated a goal of 100 per cent tokenisation of European e-commerce by 2030 alongside phasing out manual card entry.

The PCI Security Standards Council's position is the one to quote at auditors: payment tokens are "created by TSPs that are registered with EMVCo", are "issued to a cardholder in lieu of a PAN", and "during a Payment Token transaction, the merchant and acquirer do not receive or have access to the corresponding PAN". That last clause is the compliance argument in a single sentence, and it is why tokenisation is a scope conversation and not only a fraud conversation.

A short field guide to failures

Symptom	Usual cause
Token declines at exactly one merchant	Domain restriction on Card Acceptor ID; check the merchant data the acquirer is actually sending
Card shows in wallet but every tap declines	Yellow path never completed; token in Awaiting Activation, not Active
Receipt last four does not match the plastic	Correct behaviour; that is the token's suffix, not the PAN's
9F6E decodes as rubbish	Kernel mismatch: Visa Form Factor Indicator parsed as Mastercard Third Party Data or vice versa
PAR absent from the response	BIN controller not registered for a BCID, or PAR not populated on that route
Works contactless, fails in a browser	POS entry mode outside the registered domain; the token was never permitted there
HCE wallet declines after weeks unused	Limited-use keys exhausted or expired; device must reach the network to replenish
Issuer cannot locate a token by PAN	Search by `tokenUniqueReference`; the vault is keyed on the token, not the account
New card arrived, token stopped working	Issuer suppressed the automatic token update on reissue — usually deliberate, occasionally a misconfiguration
Token metadata field missing after a scheme switch	Scheme-private data element numbering differs; you cannot port a field number between networks

The chapter in one paragraph

Payment tokenisation replaces the primary account number with a Payment Token: a 13 to 19 digit Luhn-valid surrogate, drawn from a Token BIN Range of 6 to 12 leading digits, carrying its own expiry, minted by a Token Service Provider registered with EMVCo under a three-digit TSP Code, on behalf of a Token Requestor identified by an 11-digit Token Requestor ID. On a phone the token is written into a secure element — a Java Card platform chip evaluated under the EMVCo Security Evaluation process and Common Criteria — or, where there is no secure element, held in software under limited-use keys bounded by time, count or amount and evaluated under EMVCo's Software-Based Mobile Payment requirements. Provisioning is gated by identification and verification, whose outcome the industry calls green, yellow or red, and whose rigour the framework records as a Token Assurance Method, having abandoned the older two-digit Token Assurance Level at version 2.0. At the till the

token occupies DE 2, its expiry DE 14, and its chip data DE 55, where tag `5A` holds the token rather than the account, tag `9F26` holds a cryptogram computed from a key inside the device and a counter that only ever increases, and tag `9F24` may carry a 29-character Payment Account Reference whose first four characters are an EMVCo-assigned BIN Controller ID and which links every token on the account back together without exposing the account. Token Domain Restriction Controls constrain all of it by token requestor, by POS entry mode and by merchant identity, enforced by whichever party is checking — which is why the only serious public break of the model, the 2021 Express Transit relay, was not a break of the cryptography but the exploitation of a deliberately relaxed control. And because the token is a separate credential with a separate lifecycle, it can be suspended or deactivated in the vault in seconds, leaving the plastic, the account, the standing orders and the card-on-file arrangements entirely untouched. That is the whole point. The phone does not carry your card number. It carries a second one, cut for that phone, that the bank can throw away.

19.98 Common wrong ideas

Wrong: the phone sends your card number in encrypted or obscured form. **Right:** the card number is simply not there; a different sixteen-digit number, minted for that phone, occupies the account number field.

Wrong: tokens are safe. **Right:** a token on its own is just a number with no cryptography in it, and safety comes from the token plus a per-transaction cryptogram plus restrictions enforced by whoever is checking — merchant card-on-file tokens routinely run with no cryptogram at all.

Wrong: your card details never leave the device. **Right:** the real number is encrypted and sent to the network and your bank at provisioning; what is true and narrower is that the wallet cannot decrypt the token it receives back and its servers do not store it.

Wrong: your bank issues and controls the token in your phone. **Right:** for the overwhelming majority of device tokens the Token Service Provider is Visa or Mastercard, which owns the vault and the mapping table, mints the value, holds the validation keys and de-tokenises before the message reaches the issuer.

Wrong: tokenisation is a phone technology. **Right:** tokens are issued in enormous numbers to merchants, wallets and acquirers, living in ordinary databases with no secure element, no biometric and no per-tap cryptogram.

Wrong: the Payment Account Reference is a kind of substitute card number. **Right:** it is a 29-character identifier that cannot be reverse-engineered to a PAN or a token and cannot initiate a financial transaction.

Wrong: a receipt whose last four digits do not match the plastic is a misprint. **Right:** those are the token's last four digits, and they are the easiest available proof that the phone is not sending your card number.

Wrong: a reissued plastic card means re-adding the card to the wallet. **Right:** the TSP updates the token-to-PAN mapping in the vault and the token keeps working, which is also why a fraud reissue needs the issuer to suppress that update deliberately.

Wrong: host card emulation is a software secure element. **Right:** its keys sit in ordinary application memory and are made nearly worthless by limited-use thresholds, so an HCE wallet left unused and offline will eventually decline — correctly.

Wrong: the 2021 Express Transit attack broke tokenisation. **Right:** the token and the cryptography worked exactly as designed; what failed was one deliberately relaxed domain control, on one platform, for one scheme.

19.99 Chapter summary in 20 lines

1. When you tap your phone, your card number is not transmitted at all: a different number is.
2. That substitution is payment tokenisation, specified by EMVCo in a royalty-free, publicly available technical framework.
3. A Payment Token is a 13 to 19 digit Luhn-valid surrogate drawn from a real ISO BIN range, which is exactly why it traverses the existing network unchanged.
4. It carries its own expiry date, mapped by the vault to the account's expiry and not obliged to equal it.
5. It is minted by a Token Service Provider registered with EMVCo under a three-digit code, on behalf of a Token Requestor identified by an eleven-digit ID.
6. The vault holding the token-to-PAN mapping usually belongs to the network rather than to your bank, which is a real operational dependency invisible from the cardholder's side.
7. On a phone the token is written into a secure element, a certified Java Card platform chip, and the key that signs each tap never leaves it.
8. Every tap produces an eight-byte cryptogram from that key, a counter that only increases and the terminal's unpredictable number, so it cannot be replayed anywhere.
9. The device also asserts that the holder was verified on the device, which is why a phone pays £180 where a plain contactless card stops at £100.
10. A stolen token is close to worthless because of three separate walls: domain restriction, the missing cryptogram key, and the absence of any link back to you.
11. Domain restriction is enforced by token requestor ID, by POS entry mode and by merchant identity — and only as strongly as the checking party implements it.
12. Your real card number does leave the phone once, at provisioning, encrypted to the network and the issuer, because they must know which account you are asking to tokenise.
13. Provisioning is therefore the security-critical moment, gated by identification and verification whose outcome the industry calls green, yellow or red.
14. Yellow is where the design lives or dies, because a token stuck awaiting activation is a card the customer believes they have added and cannot use.
15. Where there is no secure element, Android's host card emulation holds keys in application memory and neutralises the risk with limited-use keys bounded by time, count or amount.
16. The practical difference between the two architectures is availability rather than cryptography: an HCE wallet that cannot reach the network eventually stops paying.
17. Because one account now wears many numbers, the industry had to invent the Payment Account Reference to tie them back together without exposing the account.
18. The token has a lifecycle of its own and can be suspended, deactivated or remapped in the vault in seconds, leaving the plastic, the account and every card-on-file arrangement untouched.
19. The only serious public break of the model, the 2021 Express Transit relay, exploited a deliberately relaxed exemption rather than any cryptography.
20. The phone does not carry your card number; it carries a second one, cut for that phone, that the bank can throw away.

Sources: EMV Payment Tokenisation Specification – Technical Framework, EMVCo, v2.4 (9 July 2026) and v2.3 glossary; EMVCo Token Service Provider and BIN Controller ID registration programme pages; EMV ICC

Specifications for Payment Systems Book 3 Annex A for base tag definitions; EMV Mobile Payment: Software-Based Mobile Payment Security Requirements and the SBMP Security Evaluation Process, EMVCo; EMVCo Payment Account Reference: A Primer, Secure Technology Alliance, April 2018; EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum, June 2019; Technologies for Payment Fraud Prevention: EMV, Encryption and Tokenization, Smart Card Alliance, October 2014; Industry Perspectives on the Evolution of EMV Payment Tokenization, Federal Reserve Bank of Boston, 2019; Apple Platform Security — card provisioning, Apple Pay component security and payment authorisation; Apple PassKit in-app provisioning documentation; Google Pay device tokenisation developer documentation, including ID&V best practices and app-to-app ID&V; Mastercard Developers, MDES pre-digitisation and customer service documentation; Visa Smart Debit/Credit and Visa payWave U.S. Acquirer Implementation Guide for DE 22 values; Android host-based card emulation developer guide; ISO/IEC 14443, ISO/IEC 7816-4 and ISO/IEC 7816-5; Radu, Chothia, Newton, Boureanu and Chen, Practical EMV Relay Protection, University of Birmingham and University of Surrey, 2021; Basin, Sasse and Toro-Pozo, The EMV Standard: Break, Fix, Verify, IEEE S&P 2021; PCI Security Standards Council FAQ on acquiring, issuer and payment tokens; Visa newsroom, 4 June 2024; Mastercard newsroom, June 2025. Tag collisions on `9F6E` *cross-checked against EFTlab's EMV and NFC tag list and emvlab.org.*

10

CHAPTER

The Terminal

20.0 What this chapter gives you

1. You will be able to explain why a card machine wipes its own keys the instant its case is opened, and why the result is a device that will never take another payment.
2. You will be able to say what tamper response protects and what it does not, and why overlays, shimmers and prompt manipulation make the enclosure the wrong place to look.
3. You will be able to name the three certification regimes, say who runs each and what each actually tests, and stop hearing "EMV certified" as a security statement.
4. You will be able to read a PCI approval listing and see that it attaches to a hardware revision, a firmware version and an expiry date rather than to a product name.
5. You will be able to explain why deployment takes months, and point at the stage of the process that repeats per scheme and per configuration.
6. You will be able to separate minor terminal settings from major ones, and say which change invalidates the description of an approved configuration.
7. You will be able to pick the right PCI approval class for a form factor, from a countertop terminal to a fuel dispenser to a PIN pad module inside somebody else's kiosk.
8. You will be able to explain why an unattended fuel transaction runs as a pre-authorisation and a completion, and why the enclosure that matters there is the dispenser's door.
9. You will be able to say what changed with softPOS: that PCI lists a whole solution rather than a phone, and that attestation and monitoring replaced the mesh.
10. You will be able to point at the four kernel data objects where the contactless limits actually live, and explain why a regulator can change a rule overnight and an estate cannot.

The previous chapters were about the card. Each of them had a second party standing just off the page, holding out a slot or a radio field and asking the questions. This chapter is about that second party.

It is the least glamorous object in payments and the most heavily regulated. A card is a piece of plastic a bank posts to you. A terminal is certified hardware built under audit, attacked in a laboratory with a scalpel and an oscilloscope, carrying an approval number valid until a stated date, and becoming an inert brick the moment somebody opens the case. Three separate certification regimes sit on top of it, run by three different sets of people, none of whom accepts the others' paperwork. That is why it takes months rather than days to put a new one into a shop, and why the machine in a corner shop in Leeds and the one at a motorway fuel pump in Perth are not the same class of object at all.

The plain version

Think of the card machine as a **locked translation booth**.

Two parties who cannot speak to each other need to do business. One is your card, which speaks a compressed technical language and will only answer certain questions in a certain order. The other is your bank, hundreds of miles away, which speaks a different language entirely. Between them sits a booth with a window on each side and a translator inside, who listens to the card, writes the question down in the bank's language, sends it off, waits, and tells everybody what happened.

Now the important part. The translator also hears your PIN. That single fact changes everything: it is why the booth is locked, why the walls are lined with sensors, and why there are rules about who may build one, who may deliver one, and what must happen if anybody ever gets inside.

The dye pack

Banks used to put dye packs into bags of cash. Force the bag open away from the counter, a small charge goes off, the notes are stained bright red, and the robbery has produced a bag of worthless paper. The money simply stops being money. The card machine works on exactly that principle, and it is the single most useful thing to understand about it.

Inside every one of these machines are secret numbers, called keys. They let the machine scramble your PIN so nobody along the wire can read it, and without them it cannot take a payment at all. The keys are worth stealing; the plastic case is not.

So the case is wired. A fine mesh, thinner than a hair, is printed around the inside carrying a small current. There are little switches under the screws, and sensors that notice if the machine gets very cold, or if somebody slows its clock down, or if light reaches a place inside where light never reaches. All of it runs day and night, even while the terminal sits unplugged in a stockroom.

Cut the mesh, undo the wrong screw, freeze it, drill it: the machine wipes its keys. Not "logs an alert". Wipes them, in the instant, permanently. Switch it back on and the screen says something like TAMPER, and it will never take a payment again. It goes back to the manufacturer to be rebuilt and re-keyed.

The thief has the bag. The money is red.

The family of machines

The booths do not all look the same, because the jobs differ.

There is the **countertop** one by the till, and the **portable** one the waiter carries to your table: the same machine with a battery and a radio, docking into a base near the bar. There is the **little reader that pairs with a phone**, which market traders and plumbers use: the phone does the amount and the internet, the reader does the card and the PIN.

There is the **unattended** one: the fuel pump, the car park machine, the vending machine, the ticket kiosk. Same job, no human anywhere near it, and that absence is the difficulty. In a shop, if somebody kneels at the counter for ten minutes with a screwdriver, the shopkeeper notices. At three in the morning at pump four, nobody notices.

There is the **integrated till**, where the card machine and the shop's computer talk so nobody keys the amount twice. That sounds like a small convenience. It is where most of the interesting failures happen, because now there are two computers and only one of them is certified.

And lately there is the machine that has no machine: **the shop's own phone is the terminal**, and you tap your card on the back of it. That required an entirely new rulebook, because every rule until then assumed a sealed case with a mesh in it.

A worked example: £34.60 in a bakery

Here is a sale in a small bakery with one countertop terminal. The baker keys **£34.60** into it; the screen shows 34.60 and the contactless symbol lights. You tap your card. The radio side wakes, powers your card through the air and asks which payment applications it carries; your card answers with a list, and the terminal picks one.

Now the terminal checks its own rulebook. It has been given four numbers by whoever runs its payments:

The rule	What it means	This bakery's setting
Contactless floor limit	Above this, always ask the bank	£0 — always ask
Contactless transaction limit, no phone check	Refuse a plain tap above this	£100
Contactless transaction limit, phone checked you	Refuse a phone tap above this	£5,000
CVM required limit	Above this, demand a PIN	£100

£34.60 is under £100, so no PIN. The floor limit is zero, so the bank gets asked anyway.

The terminal asks your card to sign the transaction. The card produces an eight-byte code — computed from the amount, the date, the currency, a random number the terminal invented on the spot and a counter inside the card — and hands it over. The terminal packs that code, the card number, the amount and about forty other small facts into a message and sends it down the line. Two seconds later the bank approves, the terminal beeps and prints, and the baker gives you your bread.

Now change one thing. Make it **£134.60**, because you bought a cake. Same tap, same card, but 134.60 is above the CVM required limit of £100, so the terminal asks for a PIN. You put the card in the slot and type four digits. Those digits never leave the sealed part of the machine as digits: they go straight into the secure processor, which scrambles them with one of its keys into a block of gibberish. The gibberish travels; the PIN does not.

If somebody has glued a fake keypad over the real one — an *overlay* — they get your PIN and the machine never knows, which is the first hint that the dye pack is not the whole story.

Why a new machine takes months

The baker is switching payment provider, and assumes a new terminal is like a new kettle: order Tuesday, plug in Thursday. It is not, because three different bodies must each be satisfied by their own tests.

Test one: does the plug fit? Can this reader physically talk to a card — the right voltages in the right order in the slot, a radio field of the right strength so that a card held at a sensible distance actually answers? Electrical and mechanical, done once, by the manufacturer, for the model.

Test two: does the software know the rules? Inside the terminal is a program that knows the whole question-and-answer sequence a chip card expects: what to ask, in what order, what to do with each answer, when to go online, when to decline. It is tested against a very large book of scenarios, again by the manufacturer.

Test three: does *this* setup, talking to *this* bank, work end to end? This is the one that takes months, because it must be redone for the bakery's particular combination of terminal, software version, payment provider and card schemes. Every scheme has its own test list; every test is a scripted transaction with a test card, checked field by field. One wrong value and that section starts again. Then someone must load the secret keys, two people present, in a room built for it, before the machine can take its first real payment.

That is the sentence to carry to dinner: **the card machine is a locked booth that destroys its own secrets if you open it, and it takes months to deploy not because the hardware is hard but because three separate examiners must each pass it, and only the last of them cares about your particular shop.**

Where the plain version stops being true

The booth is a good picture, and it is wrong in four ways that matter to anyone who buys, builds or certifies one.

There is no such thing as "the terminal"

The plain version speaks of one object with one approval. In the paperwork there is no such object. There is a contact interface module, separately approved; a proximity coupling device — the radio front end — separately approved; a contact kernel and up to eight contactless kernels, each separately approved; a PIN entry device with its own security approval, which may be a physically separate module bought from another vendor; a payment application on top of the kernels, which is nobody's type approval and everybody's problem; and the till, certified by no one.

Approvals attach not to a product name but to a product name **plus a hardware revision plus a firmware version plus a checksum**. Change the firmware and you may or may not still be approved, depending on which requirements the change touches; the process for answering that has a name, *delta evaluation*, and a laboratory must run it. Where a device embeds other approved devices, the renewal date of the whole is the earliest among all of them: one expiring component expires the terminal. So "is this terminal certified?" is not a yes or no question. It is a query against several lists, each with its own dates.

Tamper response protects the keys, not the transaction

Tamper response guarantees one thing: an attacker who penetrates the enclosure gets no usable cryptographic material. It guarantees nothing about attacks that never penetrate it, and those are the ones that take money. Overlays: a moulded plastic keypad sitting on top of the real keypad, recording digits, defeated by nothing inside the box. Shimmers: a paper-thin device pushed into the card slot to sit between card and reader. Prompt manipulation: persuading the terminal to display "ENTER PIN" when the software behind it is not doing what the cardholder thinks. And at unattended sites the enclosure that matters is the *dispenser's* door, not the payment module's.

PCI's requirements do address prompt control, and its technical FAQ spends considerable space on touchscreens precisely because a bezel can conceal an overlay. But the honest statement is narrower than the analogy: opening the box is one of the harder ways in, which is why intelligent attackers stopped trying.

The three certifications test three unrelated things

"EMV certified" is used in the trade as though it were a security statement. It is not. EMVCo's functional approvals assess performance and compatibility: Level 1 asks whether the electrical and radio interface conforms, Level 2 whether the kernel implements the specified processing logic. Neither asks whether the device resists attack.

The security standard for the terminal is PCI PTS POI, from a different organisation, tested in different laboratories, against a different document. Level 3 is a third thing again: not run by EMVCo, not a security test, and not a conformance test in the usual sense, but an integration test whose content each payment system writes for its own acquirers. Merchants routinely assume that possession of any one implies the others. Nothing implies anything.

With softPOS, the phone is not the terminal

The last form factor breaks the analogy rather than stretching it. When the shopkeeper's own phone accepts the tap there is no sealed enclosure, no tamper mesh, no key store you control and no PCI PTS approval, because PCI PTS approves hardware and there is no hardware to approve. What replaced it is a different architecture: cryptographic work pushed into whatever secure element the phone already has, plus continuous *attestation* — the phone proving to a back-end service, repeatedly, that it is the device it claims to be and has not been rooted, hooked or emulated — plus *monitoring*, a server-side system that can refuse to let a suspect device transact. The certified thing is the whole solution including its servers, not the handset.

The security boundary therefore moved off the counter and into somebody's data centre, and a softPOS solution can be switched off remotely in a way a countertop terminal never could: a real gain and a real new dependency, and the booth analogy has nothing to say about either.

The technical version

What is physically inside

A conventional attended terminal contains a contact card reader, a contactless front end, a secure processor with a protected key store, a keypad, a display, communications and usually a magnetic stripe head and a printer.

The **contact interface** is governed by ISO/IEC 7816: part 2 defines the eight contacts and their positions, part 3 the electrical characteristics and the transmission protocols, of which EMV uses the character-oriented T=0 and the block-oriented T=1. The reader must handle the class A (5 V), class B (3 V) and class C (1.8 V) supply classes cards may demand, negotiate the answer-to-reset, and cope with cards inserted crookedly or withdrawn mid-transaction. In EMVCo's vocabulary this component is the **Interface Module**, or IFM.

The **contactless front end** is governed by ISO/IEC 14443, at 13.56 MHz with an initial data rate of 106 kbit/s, supporting Type A and Type B modulation and framing, and supplies the card's power through the field. In EMVCo's vocabulary it is the **Proximity Coupling Device**, or PCD, specified in the EMV Contactless Interface Specification with the protocol layer in Book D of the EMV Contactless Specifications for Payment Systems.

The **secure processor** holds the keys, performs PIN block encryption, executes firmware whose authenticity it checks at boot and drives the tamper circuitry. It is permanently powered from a coin cell or supercapacitor, so the tamper sensors run while the terminal sits unplugged.

The catalogue of form factors

Form factor	Typical PCI PTS approval class	Notes
Countertop	PED	Attended, mains, fixed line or Ethernet
Portable, with base	PED	Same silicon, battery, Bluetooth or DECT to a base
mPOS reader plus phone	SCRP, or PED with SRED	Reader takes card and PIN; phone does UI and comms
Unattended kiosk, fuel, parking, vending	UPT, containing an EPP	Cardholder-operated, no attendant
PIN pad for an ATM or kiosk	EPP	A module, not a complete terminal
Card reader with no PIN	SCR, or Non PED	PIN handled elsewhere or not at all
softPOS on a merchant's own phone	none, outside PTS	Covered by PCI MPoC instead

PCI SSC's approval classes, exactly as the approved-devices listing filters them, are PED, EPP, HSM, Multi-tenant HSM, UPT, Non PED, SCR, SCRP, KLD (key loading device) and RAP (remote administration platform). The PED/EPP distinction matters commercially: an EPP is a module supplied to an ATM or kiosk manufacturer, and a UPT built around an approved EPP still requires its own laboratory evaluation, because the final enclosure introduces cardholder interfaces the EPP vendor never saw. The OEM's module cannot itself receive a UPT approval; the final form factor vendor's product does, and the module is listed separately as an approved component.

EMV classifies the environment in a single byte, tag `9F35`, Terminal Type, defined in Annex A of EMV Book 4. The first digit is who controls the terminal: 1 a financial institution, 2 a merchant, 3 the cardholder. The second digit carries environment and capability together, 1 to 3 attended and 4 to 6 unattended, and within each triple the order is online only, offline with online capability, offline only.

Operated by	Attended	Unattended
Financial institution	11 12 13	14 15 16
Merchant	21 22 23	24 25 26
Cardholder	not defined	34 35 36

A supermarket lane is typically 21 or 22, a fuel dispenser 24 or 25, a bank-operated ATM 14. Terminal Type is a *major* setting: change it and the Level 2 approval of the configuration no longer describes the device.

The PIN entry device as a certified object

The security standard is the **PCI PIN Transaction Security Point of Interaction Modular Security Requirements**, published by the PCI Security Standards Council. Version 7.0 was published on 29 May 2025, superseding v6.2, and PCI SSC describes it as containing 59 requirement changes and 23 pieces of additional guidance arising from two requests for comment during 2024. Among them: a new requirement for the physical and logical security of biometric interfaces; a new requirement permitting third-party applications, that is, app stores, on a POI device; an allowance for keys not to be zeroised on tamper where forward secrecy is used and extraction would require destroying the

processing element; and a rule that terminal security keys, such as firmware authentication and tamper or storage keys, must use cryptography with an effective key strength of 128 bits or stronger.

The standard is *modular*, which is the point of its name. The evaluation modules are: POI Device Core Requirements, covering core logical and physical requirements; Device Integration Requirements, which apply whenever previously approved components are combined; Open Protocols, covering the interface to open networks; Secure Reading and Exchange of Data, or SRED, covering encryption of account data inside the POI; and Device Management Security Requirements, covering how the device is produced, controlled, transported, stored and used across its life. Products need not support Open Protocols or SRED, but a product that implements either must be evaluated against that module.

Within those modules the requirements are lettered rather than numbered straight through: the core physical requirements run from A1, the core logical requirements from B1, and the SRED requirements are the K series, which is why practitioners talk about "requirement K20" rather than a section number.

Security is measured, not asserted. The standard uses an attack-costing framework expressed in points, combining the effort to identify an attack with the effort to exploit it. The headline physical requirement, A1, requires that defeating the device's tamper protections to disclose sensitive material demands an attack potential of at least **26 points for identification and initial exploitation, with a minimum of 13 for initial exploitation**. The same figures recur through the physical requirements, including side-channel monitoring during PIN entry.

Some specifics from PCI's technical FAQ that decide whether a device passes:

Open Protocols. New POI evaluations using the Internet Protocol suite must support TLS 1.2 or higher, with cipher suites providing at least 112 bits of security. Suites using triple DES are no longer permitted, on the grounds that a 64-bit block size does not protect large volumes encrypted under one key. Where a device runs Android, the version must be one officially supported with security patches; reports submitted with an unsupported version are rejected, and where Google no longer patches, the vendor must evidence its own monthly patching, validated by the laboratory.

Key loading. The initial terminal master key must be loaded by asymmetric techniques or manual ones such as the keypad, an IC card or a key loading device; plaintext keys and their components are never permitted over a network connection. Key blocks are mandatory: ASC X9 TR-31 for triple DES keys, TR-31 or ISO 20038 for AES, or a demonstrably equivalent scheme that has passed independent expert review. Remote key distribution using asymmetric techniques follows ASC X9 TR-34, in either its two-pass form using nonces or its one-pass form using timestamps, and binding the device to its key distribution host is a prerequisite for both. PIN block formats are constrained too: ISO 9564 format 4, the AES format, is treated explicitly, with its use distinguished under DUKPT, fixed key and master/session key schemes. Where a device is compromised, keys are not reloaded by any methodology; the device is withdrawn.

Approval identity and expiry. A PCI listing identifies a device by model name and number, a hardware number that may contain wildcard positions, a firmware number, an approval number and an expiry date, alongside a vendor-published security policy setting out the roles the device supports and the services each may use. The device must perform only its designed functions; no hidden functionality is permitted. Approvals expire, and the vendor must have the model re-evaluated against the current version before the renewal date. PCI SSC's listing records that version 3 POI approvals expired on 30 April 2021 and version 4 on 30 April 2024. Version 5 approvals were due to expire on 30 April 2026, and a Council bulletin of September 2025 extended them by one year to **30 April 2027**, which is now the earliest expiry date on the approved-devices list. These dates govern new deployments rather

than the instant removal of installed units, but they are why estates are replaced on a cycle unrelated to hardware wearing out, and why the version of an approval matters as much as its existence.

Tamper: what actually happens

PCI's technical FAQ leaves no room for a cheaper design. A card reader or non-PIN device cannot meet the physical requirements through tamper resistance and tamper evidence alone; it must have permanently active tamper detection that monitors for intrusion and responds with the immediate erasure of sensitive information, rendering the device inoperative. More broadly, in PCI's words: "A device cannot meet PTS POI requirements without having an active tamper response mechanism to zeroize secret and private keys during a penetration attack regardless of which modules of the PTS POI standard the device is designed to comply with."

The mechanisms in a real device are layered. A flexible printed circuit or serpentine trace forms a mesh wrapping the secure area, monitored for open circuit, short circuit and resistance change, so that cutting it, bridging it or drilling through it all register. Switches detect case separation, light sensors an opened enclosure, and further sensors out-of-range temperature, supply voltage and clock frequency, defeating the classic laboratory tricks of freezing memory so that it retains its contents or under-clocking a processor to make its behaviour observable. Where the device offers a service hatch, requirement K20 of the SRED module obliges the design either to make sensitive material physically unreachable from that hatch or to erase it when the hatch is opened.

The response is defined as tightly as the detection. On tamper the device must become **immediately inoperable** and must automatically erase secret information such that recovery becomes infeasible. The FAQ closes the obvious loopholes: keys that could be used to download further keys must be zeroised or rendered unusable for that purpose, covering both symmetric key-encrypting keys and the private keys used in asymmetric key loading; and where a device encrypts other keys under an internally generated key held in a register, erasing that one register suffices, provided the encrypted keys are protected under compliant algorithms and key sizes.

A tampered PIN-acceptance device must immediately stop processing all PIN-based transactions. If a reset is implemented at all, only one is permitted unless the device is removed for inspection and repair, and any intervention that re-enables transactions requires on-site presence, dual control, logged user identities with timestamps and actions, and authentication values that cannot be replayed on the same or another device. Decommissioning is a security event too: PCI PIN Security requires that a device removed from service has all keys destroyed that were, or could have been, used for any cryptographic purpose — firmware validation keys, display prompt control keys and keys used during loading included — and if that cannot be assured, the device is physically destroyed.

The practical consequences are unglamorous and expensive. Terminals tamper for reasons unconnected with attack: dropped on a tiled floor, left in a van overnight in February, battery flat for months in a drawer, or a well-meaning engineer removing a case screw. Each produces a device that cannot be repaired in the field, must be returned and must be re-keyed under dual control. It cannot be swapped for a spare unless the spare was keyed in advance, so estates carry keyed spares, and keyed spares are themselves controlled inventory.

EMV certification, level by level

Level 1 tests the electrical, mechanical and radio-frequency interfaces. For contact the object of the test is the IFM; for contactless, the PCD. EMVCo defines the processes and recognises the laboratories but performs no testing itself: laboratories test with qualified tools and submit reports, and EMVCo issues a Letter of Approval when a report demonstrates sufficient conformance. Contactless Level 1 includes

range and positioning tests, measuring how close a card must be held before the exchange succeeds — a functional question with a large effect on how the terminal feels to use.

Because a phone's antenna is not designed to be a payment reader's antenna, EMVCo launched a **Reduced Range PCD Level 1 approval process** on 10 September 2024, defining two reduced-range compliance levels with different read-range and positioning requirements, specifically so TapToMobile devices could be measured at all.

Level 2 tests the kernel. For contact, the governing document is the EMV Integrated Circuit Card Specifications for Payment Systems, version 4.4 of October 2022, in four books: Book 1 on the application-independent ICC-to-terminal interface, Book 2 on security and key management, Book 3 on the application specification, Book 4 on cardholder, attendant and acquirer interface requirements. EMVCo now calls this the Contact Kernel Approval Process; the vendor registers once, declares the product in an Implementation Conformance Statement on a versioned form — 4.4c and 4.4d are the current ones — has a recognised laboratory run the Terminal Level 2 Test Cases, and receives a **four-year** Letter of Approval.

For contactless the architecture is different and routinely misdescribed. The EMV Contactless Specifications for Payment Systems comprise Book A (Architecture and General Requirements), Book B (Entry Point), Books C-1 to C-8 (the kernel specifications) and Book D (the contactless communication protocol). Books A and B and the C-x kernel books stood at v2.12, published 6 July 2026. Books C-1 to C-7 are the individual payment systems' kernels, published by EMVCo on their behalf, with test plans managed by the relevant scheme rather than by EMVCo. Book C-8 is EMVCo's own kernel, separately versioned at v1.2 and licensable on the same terms as the contact specification; it introduces a split architecture allowing processing in different locations including the cloud, Secure Channel, Blinded Diffie-Hellman and elliptic-curve cryptography, and it may be tested either inside a full acceptance device or standalone.

A contactless product approval requires the successful validation of the PCD, of Entry Point (Books A and B) and of at least one kernel, and ends with a **three-year** Letter of Approval — one year shorter than the contact kernel's. Where a manufacturer's architecture complies with EMVCo's Modular Architecture Requirements, an EMVCo-qualified compliance auditor must validate that architecture and report to EMVCo before approval, and separate lighter processes exist for a product change, a derivative product and a renewal.

The granularity of an actual approval surprises people. A contact Level 2 Letter of Approval names a kernel version, tested on a named terminal model, with a named PIN pad running named firmware, on a named operating system. It carries approval numbers of the form `2-05820-1-1S-0426-4.4c`, in which the trailing element is the specification version tested against and the four digits before it the month and year of approval, plus a renewal date four years out, a laboratory report identifier, an eight-hex-digit kernel checksum and a separate configuration checksum. EMVCo states in it that the samples sufficiently conform, and that the letter is valid while the approval number is posted on the EMVCo website — which is to say, EMVCo can end it.

Attached is the Implementation Conformance Statement, and it is the ICS, not the letter, that tells you what the terminal can do. It enumerates, item by item: manual key entry, magnetic stripe and contact IC input; plaintext PIN, enciphered online PIN, paper signature, offline enciphered PIN, no-CVM and biometric CVMs; SDA, DDA, CDA and Extended Data Authentication; whether the terminal performs floor limit checking, random transaction selection and velocity checking; whether it holds a transaction log or exception file; and whether it supports PIN bypass.

That granularity is why re-certification is a live risk rather than a formality. Terminal settings divide into *minor* ones — Terminal Country Code, tag `9F1A`, or Transaction Currency Exponent, tag `5F36` — changeable freely, and *major* ones, such as Terminal Type, tag `9F35`, whose change invalidates the description of the approved configuration.

Level 3 is the integration test, and EMVCo describes its own role narrowly: L3 validates the integration of an approved acceptance device with its acceptance infrastructure, and the test plans are provided not by EMVCo but by the **Participant Systems** — the payment systems and other entities that elect to use the framework. What EMVCo supplies is the **EMV L3 Testing Framework**: standardised test-tool components, a qualification process for L3 test tools, and a Participant System Identifier service that lets domestic schemes place their own requirements inside a qualified tool alongside the international schemes', so the tool selects and runs the right tests for the right system. In practice your acquirer or processor obtains a qualified tool, loads the relevant test plans and runs scripted transactions against your terminal and your host, scheme by scheme and configuration by configuration, each result inspected field by field.

Why deployment takes months

Stage	Who does it	Repeats for
Select hardware; confirm PTS approval class, version and expiry	Merchant or acquirer	Each model
EMV L1 for IFM and PCD	Manufacturer, before you buy	Each hardware revision
EMV L2 for contact and each contactless kernel	Manufacturer, before you buy	Each kernel and configuration
Payment application build and configuration	Terminal estate owner	Each acquirer and each terminal type
EMV L3 integration testing	Acquirer or processor with a qualified tool	Each scheme, each configuration
Key injection under dual control	Key injection facility or acquirer	Each device
PCI DSS or P2PE scoping of the surrounding estate	Merchant	Each deployment model
Estate rollout, terminal management system, remote updates	Estate owner	Continuously

Two features of that table explain the months. First, the repeat column: L3 is per scheme and per configuration, so an estate supporting four schemes across attended, unattended and mobile configurations is not running one certification but a matrix of them. Second, the delta problem: a change to the payment application, the kernel version, the terminal type or the CVM configuration can reopen the L3 matrix and, if it touches security, the PTS evaluation as well. A laboratory test cycle cannot be paused and resumed either: if it ends with discrepancies, it ends.

Unattended terminals

The unattended payment terminal is not a countertop terminal in a metal box. PCI treats the UPT as its own approval class, covering cardholder-operated devices that read, capture and transmit card information in conjunction with an unattended self-service device, with fuel dispensers, ticketing machines and vending machines named explicitly, and notes that UPTs may have a compound architecture combining payment with the delivery of goods or services. That compound architecture is the difficulty: the payment module and the thing being sold share an enclosure, a controller and often a

maintenance regime, and the people who service the dispenser are not the people who understand the payment module.

Three consequences follow. The first is **prompt control**: in an attended environment the cardholder can see who is asking for the PIN, while in an unattended one the display and keypad may be driven by a device controller outside the secure boundary, which is why PCI constrains what may be displayed during PIN entry, and why an EPP with no display of its own must be integrated with a display it can trust.

The second is **inspection**. An overlay on a countertop keypad has a shopkeeper looking at it all day; an overlay inside a fuel dispenser has nobody. Estate owners answer with tamper-evident seals, dispenser door alarms wired into the site controller and scheduled physical inspection — none of it in the terminal standard, all of it in the operational regime around it.

The third is **the transaction shape**. An unattended fuel transaction cannot know its final amount before authorisation, because the fuel has not been dispensed; it runs as a pre-authorisation for an estimated amount followed by a completion for the actual amount, so the terminal must support offline data capture and later completion messaging, and the terminal type byte reflects that offline capability. Vending and parking behave differently again, often as small online transactions with no CVM at all. Fuel also has its own integration standard, maintained by IFSF, between forecourt controller, dispenser and payment application.

softPOS: when the phone is the terminal

The standards arrived in three steps. PCI SSC published **Software-based PIN Entry on COTS** (**SPoC**) in 2018 for PIN entry on an ordinary phone paired with a secure card reader, and **Contactless Payments on COTS** (**CPoC**) in 2019 for contactless acceptance on the phone's own NFC with no PIN. It then combined them into **Mobile Payments on COTS** (**MPoC**), first published in November 2022, with version 1.1 in November 2024, whose principal addition is allowing both PIN entry and contactless card data on the same commercial off-the-shelf device.

"Objective-based", PCI's own description of MPoC, is doing real work. PCI PTS specifies a physical enclosure and tests it with tools; MPoC cannot, because the enclosure is a phone the solution provider did not design and cannot control. It therefore states security objectives and requires evidence that the solution meets them, across the software, the back-end, the attestation mechanism and the monitoring system. Solutions, not devices, are listed.

Apple's implementation is the best-documented public example. In Tap to Pay on iPhone the Secure Element hosts the payment acceptance applets and the kernel configurations, downloaded from Apple's servers and validated for integrity and authenticity before installation. The applet performs the card read and then encrypts and signs the card data, so the card number is, in Apple's words, "secured by the Secure Element and isn't visible to the acceptance device", and is released only to the merchant's payment service provider. The Secure Element also captures the PIN digits and builds the encrypted PIN block; screenshots and screen recording are blocked during PIN entry; and because the phone has been facing the payer, the merchant must re-authenticate with Face ID, Touch ID or a passcode to get control of it back. Apple states that on iOS 18.4 or later Tap to Pay on iPhone is a PCI MPoC validated solution listed by PCI SSC, and that its servers monitor device security compatibly with MPoC. Store-and-forward transactions do not support PIN entry.

On the EMVCo side nothing is waived. A TapToMobile solution still needs Level 1 for its reader, which is why the Reduced Range PCD process exists, Level 2 for its kernels, which is why kernel configurations are provisioned onto the Secure Element at all, and Level 3 with each acquirer. So

softPOS is not "a phone with PCI approval" — PCI PTS does not approve phones — but a listed solution of which the phone is one component, and it does not remove certification effort so much as move it: the hardware burden goes and a continuous attestation, monitoring and lifecycle burden takes its place.

The integrated till, and the interface nobody certifies

An integrated deployment connects the electronic cash register to the payment terminal so the basket total flows across automatically and the result flows back. In a *fully integrated* design the ECR drives the payment device directly and is therefore in scope for card data. In a *semi-integrated* design — now the common architecture — the ECR sends only an amount and receives only a result, card data never traverses it, and the till leaves PCI DSS scope for that data, which is very often the entire commercial reason the architecture exists.

The interface between them is covered by no EMVCo or PCI approval. In practice it is either proprietary to the terminal vendor or acquirer, or an industry protocol: **nexo standards**' Retailer protocol, built on ISO 20022 messages, specifying the interface between payment application and ECR, with the companion nexo FAST specification defining the payment application itself; or IFSF's equivalents in fuel. Choosing an open protocol does not shorten Level 3, which tests the messages that reach the acquirer, not those that cross the counter.

The terminal's own configuration

Everything a terminal decides is driven by data loaded into it, per acquirer and per application identifier. These are the terminal-side data objects a practitioner meets most often, as defined in EMV Book 3 Annex A.

Tag	Name	Length	Note
9F1A	Terminal Country Code	2	ISO 3166-1 numeric
5F2A	Transaction Currency Code	2	ISO 4217 numeric
5F36	Transaction Currency Exponent	1	Minor unit position
9F35	Terminal Type	1	Table above
9F33	Terminal Capabilities	3	Card input, CVM, security capability
9F40	Additional Terminal Capabilities	5	Transaction types, data input and output
9F1B	Terminal Floor Limit	4	Per application
9F1C	Terminal Identification	8	Unique within a merchant
9F1E	Interface Device Serial Number	8	Unique to the physical device
9F15	Merchant Category Code	2	ISO 18245
9F16	Merchant Identifier	15	Assigned by the acquirer
9F4E	Merchant Name and Location	variable	
9F09	Application Version Number (terminal)	2	Per AID
9F06	Application Identifier (terminal)	5 to 16	The AID the terminal supports

Tag	Name	Length	Note
9F1D	Terminal Risk Management Data	1 to 8	Optional; Kernel 2 uses 8
9F02	Amount, Authorised	6	n12, minor units
9F03	Amount, Other	6	Cashback
9F37	Unpredictable Number	4	Generated per transaction
95	Terminal Verification Results	5	Output: what the terminal found
9B	Transaction Status Information	2	Output: what the terminal did
9F34	Cardholder Verification Method Results	3	Output: how, and whether it worked

Alongside these sit the **Terminal Action Codes** — Default, Denial and Online, five bytes each, held per application identifier. They are terminal-resident acquirer policy, bit-masked against the Terminal Verification Results together with the card's Issuer Action Codes to decide whether to decline offline, go online or approve offline. Getting them wrong is the classic cause of a terminal that approves offline what it should have sent online, and it is exactly what Level 3 exists to catch.

The contactless reader carries a second set of limits, held by the kernel rather than the contact application. In the Kernel 2 configuration these are the Reader Contactless Floor Limit at tag DF8123, the Reader Contactless Transaction Limit without on-device CVM at DF8124, the Reader Contactless Transaction Limit with on-device CVM at DF8125, and the Reader CVM Required Limit at DF8126. Those four values, per application and per currency, decide whether a tap goes through, whether it demands a PIN and whether it is refused outright.

Which is where this chapter ends, because those four numbers are also where policy lands. From 19 March 2026, following an FCA announcement in December 2025, United Kingdom firms with strong fraud controls have been permitted to set their own contactless limits in place of the old fixed pair of £100 per transaction and £300 in the background. UK Finance's guidance observes that most banks were likely to keep existing limits at first, and that "separate changes would also need to be made to card acceptance terminals and the industry rules that govern them to process contactless card payments above £100". A regulator can change a rule overnight. Changing DF8124 across a national estate is a deployment programme, and mobile wallets were never inside the cap in the first place, because they are governed by DF8125.

20.98 Common wrong ideas

Wrong: a terminal is one object with one approval. **Right:** it is a contact interface module, a proximity coupling device, a contact kernel and up to eight contactless kernels, a PIN entry device and an uncertified payment application, each with its own paperwork and its own dates.

Wrong: the approval belongs to the product. **Right:** it belongs to a product name plus a hardware revision plus a firmware version plus a checksum, and where approved devices are embedded in others the earliest renewal date among them expires the whole terminal.

Wrong: tamper response protects the transaction. **Right:** it guarantees only that an attacker who penetrates the enclosure gets no usable cryptographic material, and the attacks that take money — overlays, shimmers, prompt manipulation — never go inside the box.

Wrong: "EMV certified" means the device is secure. **Right:** EMVCo Level 1 tests the electrical and radio interface and Level 2 the kernel logic; neither asks whether the device resists attack, and the security standard is PCI PTS POI from a different organisation entirely.

Wrong: Level 3 is another conformance test. **Right:** it is an integration test whose plans each payment system writes for its own acquirers, run in a qualified tool, repeated per scheme and per configuration.

Wrong: a tampered terminal can be reset on site. **Right:** it cannot be repaired in the field; it goes back to be re-keyed under dual control, which is why estates carry pre-keyed spares as controlled inventory.

Wrong: terminals tamper because somebody attacked them. **Right:** most tamper because they were dropped, left in a van overnight in February, allowed to go flat in a drawer, or opened by a well-meaning engineer — and the response is identical.

Wrong: an approved encrypting PIN pad inside a kiosk makes the kiosk approved. **Right:** the final form factor needs its own evaluation, because the finished enclosure introduces cardholder interfaces the module vendor never saw.

Wrong: softPOS is a phone with a PCI approval. **Right:** PCI PTS approves hardware and there is no hardware to approve; what is listed is a whole solution, including its attestation and monitoring servers, of which the phone is one component.

Wrong: the FCA removing the contactless caps raised the limit. **Right:** the limits live in reader data objects across a national estate, so changing them is a deployment programme accompanied by changes to the industry rules — and wallets were never inside the cap anyway.

20.99 Chapter summary in 20 lines

1. The terminal is the second party standing just off the page in every chapter about the card: the thing holding out a slot or a radio field and asking the questions.
2. It is best pictured as a locked translation booth, listening to the card in one language and to the bank in another.
3. What makes it a certified object rather than a peripheral is that the translator also hears your PIN.
4. Inside sits a secure processor holding the keys, checking its own firmware at boot and driving a tamper circuit powered from a coin cell so it runs even while the terminal is unplugged in a stockroom.
5. A fine mesh, switches under the screws and sensors for light, temperature, supply voltage and clock frequency all report to that circuit.
6. On tamper the device zeroises its keys immediately and becomes inoperable, like a dye pack turning a bag of notes into worthless paper.
7. That protects the keys and nothing else: overlays, shimmers and prompt manipulation take money without ever entering the enclosure, which is why intelligent attackers stopped trying to open it.
8. The form factors differ by job — countertop, portable, mPOS reader, unattended kiosk, PIN pad module, softPOS — and each maps to its own PCI approval class.
9. EMV records the environment in a single byte, Terminal Type at tag `9F35`, encoding who operates the device and whether it is attended and online-capable.
10. The security standard is the PCI PTS POI Modular Security Requirements, now version 7.0, modular by design and measuring attack potential in points rather than asserting it.
11. Approvals name a hardware revision, a firmware version and an expiry date, so "is this terminal certified?" is a query against several lists rather than a yes or no.

12. EMVCo's Level 1 and Level 2 approvals assess conformance and processing logic; neither is a security test.
13. Level 3 is not EMVCo's test at all but the Participant Systems' integration test plans, run by an acquirer or processor in a qualified tool.
14. A Level 2 letter names a kernel version on a named terminal with a named PIN pad and firmware, and the attached Implementation Conformance Statement, not the letter, is what tells you what the terminal can do.
15. Settings divide into minor ones that may be changed freely and major ones, such as Terminal Type, whose change means the approval no longer describes the device.
16. Deployment therefore takes months because Level 3 repeats per scheme and per configuration, because a delta can reopen the matrix and the security evaluation with it, and because keys must then be injected under dual control.
17. Unattended terminals are a class of their own: nobody to notice an overlay, a display and keypad possibly driven from outside the secure boundary, and an enclosure shared with the goods being sold.
18. A fuel transaction cannot know its own amount before dispensing, so it runs as a pre-authorisation followed by a completion, and the terminal type byte must reflect that offline capability.
19. softPOS removes the enclosure and replaces it with continuous attestation and server-side monitoring, moving the security boundary off the counter and into a data centre without removing any EMVCo obligation.
20. Everything the terminal decides comes from data loaded into it, and the four contactless reader limits are where policy finally lands — which is why a rule can change overnight and a national estate cannot.

Sources: EMVCo specification, approval-process and knowledge-hub pages, published Letters of Approval, the EMV Contactless Specifications for Payment Systems and the EMV Integrated Circuit Card Specifications v4.4; PCI Security Standards Council publications, including the PTS POI Modular Security Requirements v7.0 announcement, the PTS Device Testing and Approval Program Guide, the PTS POI technical FAQs, the approved PTS devices listing and the MPoC standard pages; Apple's Tap to Pay on iPhone security documentation; UK Finance guidance on contactless card limits; ISO/IEC 7816 and ISO/IEC 14443; nexo standards and IFSF documentation.

CHAPTER

11

Point-to-Point Encryption

21.0 What this chapter gives you

1. You will be able to explain why a shop can take card payments for a decade and never once hold a readable card number, and why that is a design decision rather than an achievement.
2. You will be able to tell a PCI-listed P2PE solution from a non-listed encryption solution, and say which of the two entitles a merchant to the short questionnaire.
3. You will be able to read a key serial number, name its fields in both the TDEA and the AES layouts, and explain why real-world KSNs so often begin `FFFF`.
4. You will be able to work out why a DUKPT device gets 1,048,575 transactions under TDEA and 2,448,023,842 under AES, rather than the powers of two those counters would suggest.
5. You will be able to explain why extracting a terminal's key state exposes every future transaction and no past one, and why the remedy is re-injection rather than investigation.
6. You will be able to say why the card number and the PIN in the same transaction travel under different keys, and point at the derivation data that makes it so.
7. You will be able to explain to a merchant why encryption cannot support a refund six weeks later, and name the mechanism that can.
8. You will be able to list the four conditions that silently destroy a merchant's scope reduction while the shop carries on working perfectly.
9. You will be able to diagnose the standard failures — key or KSN mismatch, counter exhaustion, a format 4 PIN block meeting a format 0 host, key block version mismatch — from their symptoms.
10. You will be able to explain why perfectly functional card readers were replaced in 2024, and what the supplier meant when it said "compliance".

There is a garden centre outside Norwich that takes about eleven hundred card payments a week and has never held a card number. Not in the till. Not in the back-office PC that runs the stock system. Not in the cloud service that sends the loyalty emails. Not in the nightly export that goes to the accountant. Card numbers have passed through that building at a rate of one every four minutes since 2016, and not one of them has been legible to anybody on the premises, including the owner.

This is not because the garden centre is unusually sophisticated. It is because the card reader on the counter arrived from the supplier already loaded with a secret, and turns every card it reads into an unreadable block before the till software is allowed near it. The shop is not trusted with card numbers. It is not asked to be trustworthy. The design assumes that sooner or later somebody will get into the shop's network, and arranges for that to be uninteresting.

The previous chapter was about the terminal as a certified object. This chapter is about what the terminal does with what it reads, and about the single commercial fact that pays for all of it. Encryption

in the terminal is not bought for elegance. It is bought because a merchant who cannot read card numbers has a much smaller compliance problem than a merchant who can, and the difference is measured in weeks of audit and thousands of pounds a year.

The plain version

Imagine you run a shop, and imagine that the most dangerous thing in the shop is not the cash in the till but a notebook by the door in which every customer's card number is written down. You do not want the notebook. You never wanted the notebook. But every card machine you have ever seen had to read the number in order to ask the bank for the money, and if the machine read it, something in your shop knew it.

Now imagine a different machine. This one has a slot for the card and a little sealed box inside it. When a customer inserts a card, the number goes straight into the sealed box and is scrambled there. What comes out of the machine is not a card number. It is a block of nonsense, thirty-two characters long, that looks like somebody sat on a keyboard. Your till receives the nonsense. Your network carries the nonsense. Your broadband router, your stock system, your accountant's export, the hacker who eventually gets into your Wi-Fi: all of them get nonsense.

Somewhere else entirely, in a building you will never visit, there is a matching sealed box. It is the only thing in the world that can turn that nonsense back into a card number, and it does so for a fraction of a second, inside its own metal case, in order to ask the bank for the money. There are exactly two points in this story where a readable card number exists: inside the sealed box in your card reader, and inside the sealed box in that distant building. Between them there is a pipe full of nonsense. That is why it is called point-to-point encryption. The two points are the only places worth attacking, and neither of them is in your shop.

A worked example

A customer buys £46.80 of compost and bark chippings. She taps her card. Inside the reader, the chip hands over a card number, sixteen digits long, and the reader immediately does two things with it.

First, it keeps a small, deliberately useless summary: the first six digits and the last four, so the till can print `---- ---- ---- 0016` on the receipt and so the software knows the card was a Visa debit rather than an American Express. Second, it takes the full number and scrambles it with a key.

Here is the part that is cleverer than people expect. The reader does not have one key. It has a book of keys, and it uses a different page for every single transaction. The compost sale uses page 1. The next customer, buying a rose bush at 10:14, uses page 2. The man who buys two bags of gravel at 10:19 uses page 3. And each page, once used, is torn out and burnt: the reader physically overwrites the key it just used so that it no longer exists inside the machine. The book has room for roughly a million pages in older readers and around two thousand million in newer ones, which at eleven hundred transactions a week is more than the reader will live to spend.

So the block of nonsense that arrives at the till carries one extra thing with it: a page number. Not a key, a page number. Something like "device 90123456, transaction 1". The distant sealed box holds the master recipe from which every page of that particular reader's book can be worked out, so when it is told "device 90123456, page 1", it can reconstruct page 1, use it once to unscramble the compost sale, and throw it away again.

Count what the thief gets if he owns your shop completely. He gets the nonsense blocks. He gets the page numbers, which are not secret and are useless without the master recipe. He gets `---- 0016`

and the fact that a Visa debit card spent £46.80 on compost. He cannot manufacture a card, buy a television, or sell a list of numbers, because the shop never had a list.

And the boring reason anybody bothers

Everyone who takes card payments has to answer to a rulebook called PCI DSS. If your systems handle readable card numbers, the version of that rulebook you have to satisfy is enormous: hundreds of individual requirements about firewalls, patching, anti-virus, logging, password rules, penetration tests and network segmentation, applied to every computer that the card number could conceivably touch, which in a small business means every computer.

If you use one of these encrypting readers, and if the encrypting reader is part of a scheme that has been formally inspected and published on a list, you are allowed to answer a much shorter questionnaire instead. It asks whether your card readers are the ones you think they are, whether anybody has glued a fake one on top, whether you followed the manual the supplier gave you, and whether you have written down who is responsible. That is very nearly the whole of it, because there is no computer in the shop holding card numbers for the long questionnaire to protect.

That reduction is the product. The cryptography is the mechanism.

Where the plain version stops being true

The reader does not encrypt everything, and it does not use one key

The plain version implies that nothing readable leaves the sealed box. In practice the box deliberately emits readable things, because the shop cannot function otherwise. Almost every deployment returns the first six or eight digits and the last four in clear, so that the till can route by card range, apply the right surcharge rules and print a receipt a human can match to a card. Many return the expiry date and the cardholder name in clear too. That is a choice made by the solution designer, and it is a choice with consequences: a BIN plus last four is not a card number, but it is not nothing either.

More importantly, the PIN is not protected by the same key as the card number, and never was. Card data and PIN travel under separate keys derived from the same hierarchy, so that a compromise of the account-data key does not hand over PINs, and so that PIN handling stays inside the older, stricter regime that governs PINs specifically. There is not one encrypted blob in the message. There are two, with different keys, different formats and different rulebooks.

Encrypting is not the same as being out of scope

This is the correction that costs merchants the most money. A vendor who says “our terminals encrypt end to end” has told you something about their engineering and nothing about your paperwork. The short questionnaire is available only where the whole arrangement — device, firmware, key loading, key management, decryption environment, and the manual the merchant is handed — has been assessed against the P2PE Standard by a qualified assessor and published on the Council’s list of validated solutions.

Solutions that encrypt in hardware but have never been through that process are called non-listed encryption solutions, and there is a separate, formal, and distinctly less generous route for them: an assessor documents what the solution actually achieves, and the merchant’s acquirer decides how much scope reduction, if any, to grant. It may be a great deal. It is not automatic, and it is not the merchant’s decision.

One-time keys are not one-time pads, and the protection runs backwards in time only

The plain version's book of pages suggests that each transaction is sealed off from every other. That is half true, and the half that is false matters.

The keys are derived from one another in a chain that cannot be run in reverse. An attacker who extracts the working state from a reader cannot compute the keys it used yesterday, so yesterday's captured traffic stays safe. But the state he extracts is precisely the material from which every future key of that device is derived. From the moment of compromise until the device is taken out of service and re-keyed, he can decrypt everything it sends. The scheme protects the past, not the future.

And behind every device's book of pages sits one master secret, held by the solution provider, from which the initial key of every reader in the family is derived. Lose one reader and you lose one shop's future traffic. Lose that master secret and you lose every reader derived from it, retrospectively and prospectively, at which point the only remedy is to re-inject the entire estate. This is why the master secret never exists in one piece, in one person's hands, outside a tamper-responsive box.

The second point is not your bank, and encryption gives you nothing to hold on to

"Point to point" invites the assumption that the far point is the merchant's bank. Usually it is not. The decrypting box normally belongs to whoever runs the encryption scheme — a gateway, a processor, a terminal estate manager — and the card number is decrypted there before an authorisation message is built and sent onward. The merchant's acquirer may be a further hop away, and may never see the encrypted form at all.

There is also something encryption structurally cannot do. Because the ciphertext changes on every transaction, it is useless as an identifier. It cannot tell you that the woman buying compost today is the same woman who bought a hosepipe in May, cannot be stored for a refund six weeks later, and cannot be used to bill a subscription. For those a merchant needs a stable substitute for the card number, valid for months, safe to store, and that is a different mechanism with a different name and a different scope story. Encryption protects data in flight. Tokenisation replaces it at rest. Most real estates use both, and confusing them is the commonest error in this field.

The technical version

What "P2PE" means in the Council's own words

Point-to-point encryption in the ordinary engineering sense is any arrangement that encrypts account data at the point of capture and decrypts it somewhere trusted. P2PE with a capital P is narrower: it is a PCI Security Standards Council programme, with a standard, an assessor qualification, a reporting template and a public list.

The Council's own summary is worth quoting because the wording is load-bearing. A PCI-listed P2PE solution "cryptographically protects account data from the point where a merchant accepts the payment card to the secure point of decryption", with the effect that account data — "cardholder data and sensitive authentication data" — "is unreadable until it reaches the secure decryption environment". The current standard is the *PCI Point-to-Point Encryption Standard* v3.2, published in June 2025, superseding v3.1 of September 2021; the accompanying Program Guide reached v3.2 in July 2026.

Four defined objects do most of the work in that programme, and practitioners who mix them up write incoherent contracts:

A **P2PE solution** is the complete package — devices, applications, key management, decryption — offered to merchants and listed as a unit. A **P2PE application** is "software with access to clear-text

account data, intended to be loaded onto a PCI-approved point of interaction (POI) device and used as part of a P2PE solution". A **P2PE component** is a service that is assessed and listed on a stand-alone basis and can be incorporated by reference into somebody else's solution. A **P2PE solution provider** is "an entity, usually a third-party such as a processor, acquirer (merchant bank), or payment gateway, that designs, implements, and manages the P2PE solution", and — the sentence that decides liability arguments — "may outsource certain responsibilities, but will always retain overall responsibility for the P2PE solution".

The merchant is not on that list, with one exception: a merchant that builds and runs its own encryption estate is assessed as a merchant-managed solution, against its own reporting template, and cannot then use the short questionnaire as if it were somebody else's customer.

Five domains and the paperwork that maps onto them

The v3 standard organises its requirements into five domains, numbered 1 to 5, and assessments are reported as a set of Reports on Validation, abbreviated P-ROV. The mapping is set out explicitly in the summary table of the Solution P-ROV template:

Assessed function	Domains
P2PE Solution Management	Domain 3
Encryption Management Services (EMS)	Domains 1 and 5
Decryption Management Services (DMS)	Domains 4 and 5
Key Management Services (KMS)	Domain 5, where not already covered by EMS or DMS
P2PE Application	Domain 2

Read that table as an architecture diagram rather than a filing system. Domain 1 is the encryption side: the devices, their deployment, their firmware, their chain of custody. Domain 2 is any software running on the device with sight of clear-text account data. Domain 3 is the governance of the solution as a whole, including the instructions given to merchants. Domain 4 is the decryption environment. Domain 5 is cryptographic key operations, and it is shared, which is why it appears twice: the same key-management requirements bind the party that injects keys into terminals and the party that decrypts with them.

Requirements are identified in a compound style — Domain 1's requirement `1B-1.1.1` and Domain 2's `2A-3.1.2`, for instance, both concern devices and applications that facilitate merchant-managed decryption — and anyone drafting a gap analysis should cite them in that form rather than by paraphrase.

The programme also lists individual components, referred to throughout the templates by abbreviation: EMCP, PDCP, PMCP, KMCP and KLCP, along with key injection facilities (KIF) and certification and registration authorities (CA/RA). A solution provider assembles validated components and inherits their assessed scope. Where remote key distribution is used, its requirements are additional to, not instead of, the rest.

Two more artefacts matter operationally. The **P2PE Instruction Manual** (PIM) is the document the solution provider must produce and the merchant must follow; deviating from it is the most common way a merchant silently loses eligibility for the short questionnaire. The **Attestation of Validation** (AOV) is the signed summary that accompanies the P-ROV and precedes listing.

The device: PTS POI approval and the SRED module

P2PE encryption happens in a PCI PTS approved point-of-interaction device. The relevant standard is the *PCI PIN Transaction Security (PTS) Point of Interaction (POI) Modular Security Requirements*, v7.0 as of May 2025, with v6.2 (January 2023) and v6 (June 2020) still supporting large fielded populations. Its structure is modular, and the module that makes P2PE possible is Evaluation Module 4, *Secure Reading and Exchange of Data* — SRED — whose requirements are lettered K under the heading "Account Data Protection".

SRED is where the sealed box of the plain version is specified. Requirement numbering has been revised between POI versions, so cite the version you are certifying against, but the substance of the K series has been stable since it was introduced:

All account data must be "either encrypted immediately upon entry or entered in clear-text into a secure device and processed within the secure controller of the device", with no method of accessing the clear-text data. Encryption must use "only ANSI X9 or ISO-approved encryption algorithms (e.g., AES, TDES)" and should use approved modes of operation. Secret and private keys held in the device to support account data encryption must be "unique per device" — a single requirement that rules out the entire class of shared-key deployments that used to be normal. Encrypting or decrypting arbitrary data with an account-data key or key-encrypting key is forbidden, which closes the obvious decryption-oracle attack. When operating in encrypting mode there must be "no mechanism in the device that would allow the outputting of clear-text account data", and the secure controller may release clear-text account data only to authenticated applications executing within the device. Where the device generates surrogate PAN values for output, it must not be possible to determine the original PAN from the surrogate, and where a hash is used the input must include a salt of at least 64 bits. Devices must also have characteristics that "prevent or significantly deter" their use for exhaustive PAN determination — that is, for BIN-range brute forcing by a criminal with a stolen terminal.

Determining the account-data encryption keys by penetrating the device or by monitoring its emanations, including power fluctuations, must require an attack potential above a defined threshold: SRED is tested against side channels, not only screwdrivers.

Note what SRED does not cover. PINs are governed by the PIN-related modules of the same document and by the PCI PIN Security Requirements. A device can be PTS approved without SRED, which is why "PCI approved terminal" is not a synonym for "encrypting terminal", and why terminal estates contain both.

What counts as account data

The scope of encryption is account data, not merely the PAN: the primary account number, the full track data or track-equivalent data from chip or stripe, the expiry date, the service code, and the card verification values where present. Sensitive authentication data — full track data, card verification codes, PINs and PIN blocks — must not be retained after authorisation at all, encrypted or otherwise, a prohibition that survives every scope reduction in this chapter and is stated in PCI DSS v4.0.1 at Requirement 3.3.1.

Where a PAN is displayed, the maximum that may be shown is the BIN and the last four digits (Requirement 3.4.1); where it is stored, it must be rendered unreadable (Requirement 3.5.1). Those two rules, not P2PE, are the source of the `---- 0016` convention on receipts.

DUKPT, part one: the TDEA scheme

Derived Unique Key Per Transaction is the key-management scheme that makes per-transaction keys practical without a network round trip. It is specified in ANSI X9.24 — Part 1 for the original triple-DEA scheme, Part 3 for the AES scheme. Every practitioner meets the TDEA version first because it is what most fielded devices still use.

The structure is a three-level hierarchy. At the top is the **base derivation key** (BDK), a double-length TDEA key held only inside the solution provider's HSMs. From the BDK and a device-specific serial value, the provider derives one **initial PIN encryption key** (IPEK) per device, injects it, and does not keep it. The device then never uses the IPEK for a transaction: it uses it to populate a set of twenty-one future-key registers, and from those it derives the key for each transaction in turn, erasing each key after use through a transformation that cannot be inverted. That non-reversibility is the whole security argument: the device's present state does not reveal its past keys.

Every message carries a **key serial number** (KSN) so the receiving HSM knows which key to reconstruct. The TDEA KSN is ten bytes. In the conventional layout it is read as a 24-bit key set identifier, a 19-bit device identifier and a 21-bit transaction counter, left-padded with `F` nibbles to fill the ten bytes — which is why real-world KSNs so often begin `FFFF`. Field engineers also describe the same thing as "6-5-5" after the hexadecimal digit groups; on that reading a scheme permits roughly sixteen million BDKs, five hundred thousand devices per BDK and a million transactions per device.

The million is not `2^21`. The algorithm increments the counter so that no more than ten of its twenty-one bits are ever set at once, because the count of set bits determines how many derivation steps are needed. The usable counter values therefore number 1,048,575, and a device that reaches the end of its counter must be re-injected rather than rolled over.

DUKPT, part two: the AES scheme, with real numbers

ANSI X9.24-3-2017 replaces the TDEA construction with an AES one, and changes the vocabulary: the per-device key is now the **initial key** (IK) rather than the IPEK, since it derives data and MAC keys as well as PIN keys. Key types may be two-key TDEA, three-key TDEA, AES-128, AES-192 or AES-256, and the same tree serves all purposes.

The AES KSN is twelve bytes: a 32-bit BDK identifier, a 32-bit derivation identifier and a 32-bit transaction counter. The first eight of those bytes together are the **initial key ID**, which is what the IK is derived from.

Derivation is a keyed function over a sixteen-byte block of derivation data whose layout is fixed:

Bytes	Field	Example
0	Version	`01`
1	Key block counter	`01`
2–3	Key usage indicator	`1000`
4–5	Algorithm indicator	`0002`
6–7	Key length in bits	`0080`
8–15	Initial key ID, or derivation ID plus transaction counter	`90123456 00000001`

The key block counter exists so that keys longer than one AES block can be produced from more than one derivation block. The algorithm indicator distinguishes key types: `0000` for two-key TDEA and

`0002` for AES-128 in the standard's own test vectors. The key usage indicator is the field that separates purposes, and its values are worth memorising because they appear in HSM command traces:

Usage	Meaning
`0002`	Key encryption key
`1000`	PIN encryption
`2000`	Message authentication, generation
`2001`	Message authentication, verification
`2002`	Message authentication, both ways
`3000`	Data encryption, encrypt
`3001`	Data encryption, decrypt
`3002`	Data encryption, both ways
`8000`	Key derivation
`8001`	Key derivation, initial key

The published test vectors accompanying X9.24-3-2017 make the whole mechanism concrete, and they are the fastest way to prove a new implementation correct. Take the base derivation key `FEDCBA98 76543210 F1F1F1F1 F1F1F1F1` and the initial key ID `12345678 90123456`. The derivation data for the initial key is `01018001 00020080 12345678 90123456` — version 1, block counter 1, usage `8001` for an initial key, AES-128, 128 bits, then the full initial key ID — and it yields the initial key `1273671E A26AC29A FA4D1084 127652A1`.

Now take transaction 1. The device's derivation key for that transaction is `4F21B565 BAD9835E 112B6465 635EAE44`, and from it three working keys fall out, differing only in two bytes of input:

Purpose	Derivation data	Resulting key
PIN encryption	`01011000 00020080 90123456 00000001`	`AF8CB133 A78F8DC2 D1359F18 527593FB`
MAC generation	`01012000 00020080 90123456 00000001`	`A2DC23DE 6FDE0824 A2BC321E 08E4B8B7`
Data encryption	`01013000 00020080 90123456 00000001`	`A35C412E FD41FDB9 8B69797C 02DCD08F`

Note the last eight bytes: not the whole initial key ID but its low four bytes, `90123456`, followed by the transaction counter `00000001`. And note that the PIN key and the data key for the very same transaction are different keys. When this chapter said earlier that the card number and the PIN travel under separate keys, that is the arithmetic that makes it so.

The AES counter is 32 bits and, as in the TDEA scheme, only values with a limited number of set bits are used — up to sixteen one-bits rather than ten. That yields 2,448,023,842 usable transactions per device, which is the practical meaning of "the reader will not live to spend the book".

	TDEA DUKPT	AES DUKPT
Standard	ANSI X9.24-1	ANSI X9.24-3-2017
Per-device key	IPEK	IK (initial key)
KSN length	10 bytes	12 bytes
KSN layout	24-bit key set ID, 19-bit device ID, 21-bit counter	32-bit BDK ID, 32-bit derivation ID, 32-bit counter

	TDEA DUKPT	AES DUKPT
Counter constraint	at most 10 bits set	at most 16 bits set
Transactions per device	1,048,575	2,448,023,842
PIN block	ISO 9564-1 formats 0, 1, 3	ISO 9564-1 format 4

The PIN block row is a migration trap in its own right. Format 4 is the AES format and is required with AES keys; a host expecting format 0 will decrypt an AES-era PIN block into rubbish and reject perfectly good PINs.

Why a different key every time

State the threat model precisely, because "more keys is more secure" is not an argument.

An attacker with a year of a shop's traffic holds a year of ciphertexts, each under a distinct key. No volume of traffic helps him: he cannot accumulate material under one key, and breaking one transaction gains him one transaction. With a single static terminal key — the arrangement DUKPT replaced — the same capture is one key away from a year of card numbers.

An attacker who steals a terminal and successfully extracts its live key state gains every future transaction of that terminal and no past ones, for the reason given earlier: the derivation runs one way. In practice extraction is what SRED's tamper and side-channel requirements are there to prevent, and the commercial consequence of a suspected extraction is a re-injection, not an investigation.

An attacker inside the merchant's network gains ciphertext plus KSNs. The KSN is not secret; it is an index. Publishing it is not a weakness, and in fact its counter is useful defensively: a host can detect duplicate KSNs, replayed transactions and impossible counter jumps, because the counter of a healthy device only ever increases.

The party with real exposure is the solution provider holding the BDK, which can regenerate any key of any device in its family from the KSN alone. Hence the requirements in Domain 5, and hence HSMs.

Key blocks, and why integrations broke in 2023 and 2025

A key in transit needs its attributes bound to it. A bare 16-byte value carries no statement of what it is for, and a host that will use whatever it is sent as whatever it is told can be manipulated into using a PIN key as a data key, or a data key as a key-encrypting key. Key blocks solve this by encrypting the key together with a header that states its usage, algorithm, mode of use and exportability, and authenticating the whole structure.

The format that the payments industry standardised on was ANSI X9 TR-31, since superseded by **ANSI X9.143-2022**; the two are generally interoperable and the names are used interchangeably in vendor documentation. The key that protects a key block is a key block protection key (KBPK), distinct in principle from a plain key-encrypting key because it protects the header as well as the key.

Key blocks stopped being optional on a published timetable. PCI PIN Security Requirement 18-3 states that encrypted symmetric keys must be managed in key blocks, and the Council's revision bulletin of 17 July 2020 fixed the phases as follows:

Phase	Scope	Effective date
1	Internal connections and key storage within service provider environments, including all applications and databases connected to HSMs	1 June 2019
2	External connections to associations and networks	1 January 2023
3	All merchant hosts, point-of-sale devices and ATMs	1 January 2025

Those dates explain a decade of integration pain. Phase 2 broke host-to-network links that had exchanged variant-format keys for twenty years. Phase 3 pushed the same change out to terminal estates and merchant hosts, which is why perfectly functional readers were replaced in 2024 by suppliers who described the reason as "compliance" rather than explaining it.

Underneath sits the algorithm question. PCI's requirements are expressed in terms of effective key strength, and double-length TDEA at 112 bits still satisfies them; NIST, in SP 800-131A Rev. 2, treats three-key TDEA encryption as deprecated through 2023 and disallowed thereafter. The practical result is a long overlap in which AES DUKPT and TDEA DUKPT coexist in the same estate, sometimes in the same shop, and every host must support both.

Key injection

An initial key has to get into a device without ever existing in the clear where a person could read it. There are two accepted routes.

The first is a **key injection facility**: a physically secured room, assessed against the relevant domain requirements, where devices are connected to a key loading device driven by an HSM. Access is under dual control — no single person can perform a key operation — and the BDK components are held under split knowledge, so that no individual has enough material to reconstruct the key. Components are usually delivered as printed component sheets held by named custodians in tamper-evident envelopes, combined only inside the HSM during a witnessed and logged key ceremony.

The second is **remote key distribution**, in which the device authenticates itself cryptographically to a distribution host and receives its initial key over an untrusted network. The mechanism is asymmetric: ANSI X9.24-2 specifies the approach, and TR-34 is the protocol implementation used in practice, with a key distribution host and a key receiving device exchanging signed and enveloped material. The POI requirements insist on mutual authentication between host and device for remote key distribution, and the P2PE templates treat remote key distribution requirements as additional to the baseline. Remote injection is what makes it economically possible to ship a reader to a merchant by courier rather than through a depot.

Chain of custody is assessed as seriously as cryptography. Devices must be shipped with tamper-evident packaging, tracked by serial number, inspected on receipt against a manifest, and stored securely before deployment. A device whose whereabouts cannot be accounted for between the injection facility and the shop counter is, formally, an unknown device.

The decryption environment

At the far point, decryption occurs inside a hardware security module. This is not an ordinary server with a cryptographic library: it is a tamper-responsive device that erases its secrets when opened, holds a local master key that never leaves it, and refuses to output the keys it protects. Payment HSMs are certified against the *PCI PTS Hardware Security Module (HSM) Modular Security Requirements* — v4.0 of December 2021 for most currently fielded units, with v5.0 published in May 2026 — and typically also against FIPS 140-3 (formerly 140-2) at Level 3 under the NIST validation programme. ISO 13491 covers secure cryptographic device requirements generally, and ISO 11568 covers retail key management, both of which the P2PE and PIN standards reference.

The properties that matter architecturally are simple to state and expensive to achieve. Clear-text account data exists only inside the HSM's boundary. Keys are used, not read: the application asks the HSM to decrypt a block under the key derived from a given KSN, and receives plaintext, never the key. Key operations require dual control and are logged. The BDK exists in the clear nowhere.

In P2PE terms this is the Decryption Management Services function, assessed against Domain 4 with the shared key requirements of Domain 5. It is also the point at which the merchant's scope reduction is paid for by somebody else's scope: everything the merchant no longer has to protect, the solution provider protects instead, under audit, and prices accordingly.

Scope reduction, stated honestly

The commercial case is real, and it is narrower than sales decks imply.

A merchant using a PCI-listed P2PE solution, and following the PIM, may validate using SAQ P2PE instead of the long self-assessment questionnaires or a full Report on Compliance. What remains is drawn chiefly from the physical-security and policy requirements: knowing which devices you have and where, checking them for tampering and substitution, controlling physical access to them, handling any paper records that carry card numbers, training staff, maintaining an incident response plan, and managing the third parties involved. What disappears is the mass of network, patching, malware, logging and segmentation requirements — not because they stopped being good ideas, but because there is no in-scope system in the shop for them to apply to.

Four conditions routinely destroy that position, and all four are common:

The solution must be listed, and listed for the specific device types and applications in use. A validated solution running an unlisted application on a listed device is not a validated solution.

The merchant must not have another route by which clear-text account data enters its systems. A shop with a beautiful P2PE estate on the counter and a member of staff typing telephone orders into a browser has a cardholder data environment again, and the browser is in it.

The PIM must be followed. It specifies device inspection frequency, permitted device configurations, what to do about a suspected tamper, and how devices must be shipped and stored. Ignoring it is not a technical failure; it is loss of eligibility.

Encrypted account data that the merchant stores is out of scope only if the merchant genuinely has no means of decrypting it. A merchant who holds ciphertext and also holds, anywhere, the key or the decryption service is back in scope for that data.

For non-listed solutions, the Council's *Assessment Guidance for Non-listed Encryption Solutions* (June 2020) sets out how a QSA documents the solution's effect so that the acquirer can decide what scope

reduction to accept. This route is widely used and perfectly respectable. It is simply not the same product, and it does not entitle a merchant to SAQ P2PE.

Tokenisation at the acquirer versus encryption in the terminal

These are complementary mechanisms that solve different problems, and the confusion between them causes real architectural mistakes.

Encryption in the terminal protects a card number in flight, from capture to decryption, and produces a value that changes on every transaction. Tokenisation replaces a card number with a surrogate that persists, so it can be stored, quoted, refunded against and billed again.

There are three distinct kinds of token in payments, and only one of them belongs in this chapter. **Acquirer or gateway tokens** are issued by the merchant's acquirer or gateway after decryption: the vault stores the PAN, returns a surrogate, and the merchant stores the surrogate. **EMVCo payment tokens** are network tokens, issued by a token service provider, with their own device and domain controls, covered in the chapter on the token in your phone. **Device-generated surrogate PANs** are produced inside the POI itself, under the SRED requirements quoted earlier, and are useful for local matching where the merchant needs no server round trip.

	Terminal encryption (P2PE)	Acquirer tokenisation
What it produces	Ciphertext, different every transaction	A stable surrogate value
Where the secret lives	Solution provider's HSMs (BDK)	Vault operator's database and HSMs
Useful for refunds and recurring billing	No	Yes
Protects data in flight	Yes	No, on its own
Formal scope effect for the merchant	SAQ P2PE, where the solution is listed	Reduces stored-data obligations; no equivalent listed programme
Fails how	Key compromise, device compromise	Vault compromise, or a token usable by anybody who steals it

The right architecture for a merchant that needs both is a listed P2PE solution for capture and an acquirer token returned in the authorisation response for storage. The terminal sends ciphertext plus KSN; the provider decrypts, authorises, and returns a token; the merchant stores the token and never the number. Refunds, subscriptions and analytics all run on the token.

Two cautions. A token a thief can present for payment without further authentication has merely moved the value rather than removed it, which is why tokens are scoped to one merchant and one channel. And format-preserving tokens that pass the Luhn check are convenient for legacy software and treacherous for governance, because nothing in a schema distinguishes them from real card numbers.

What actually goes on the wire

The encrypted blob does not travel on the card networks. It travels from the terminal to the decrypting party over that party's own protocol, and it dies there.

A typical terminal-to-host request carries the ciphertext as a hexadecimal or base64 string, the KSN in the clear, an indication of the encryption scheme and format in use, a masked PAN, the expiry if the

solution exposes it, and the ordinary transaction fields — amount, currency, the EMV data from the chip. The host passes the ciphertext and KSN to its HSM, which derives the transaction key from the BDK and the KSN, decrypts inside its boundary, and returns the account data.

Only then does the message the rest of this book is about get built. The clear PAN goes into the ISO 8583 primary account number field, DE 2; track data, where present, into DE 35 or DE 45; the PIN block, encrypted under a different key and translated by the HSM from the terminal's key to the acquirer's working key, into DE 52, with key management data in DE 53. The scheme never sees the P2PE ciphertext, has no knowledge of the BDK, and cannot tell from the authorisation message whether the merchant's terminal was encrypting or not. That handoff is the subject of the next chapter.

A short field guide to failures

Key or KSN mismatch. Decryption returns plausible-length rubbish. Almost always the host has been configured with the wrong BDK for that key set ID, or the KSN is being parsed with the wrong field boundaries. Check the key check value of the BDK before suspecting anything subtle.

Counter exhaustion. A device that has quietly reached the end of its counter stops encrypting. It is not broken; it needs re-injection. On TDEA devices this is reachable inside a decade at high volume.

Format assumptions. An AES-era device sending an ISO 9564-1 format 4 PIN block to a host expecting format 0 produces PIN verification failures that look like customer error. The same class of fault arises when a host assumes the encrypted field includes the expiry and it does not.

Key block version mismatch. A host that only accepts one key block format will reject an otherwise valid key. This is the ordinary consequence of the 2023 and 2025 phases arriving at different times in different estates.

Tamper trips in transit. Tamper-responsive devices sometimes respond to being dropped, frozen or X-rayed. A device that arrives dead has done its job.

Silent loss of eligibility. The most expensive failure has no error message. Somebody installs an unlisted application, or starts keying card numbers into the back office, or stops doing the device inspections the PIM requires, and the shop stays fully functional while its right to the short questionnaire lapses. This is discovered at the next assessment, or after a breach.

The through-line of this chapter is that none of the cryptography here exists to protect the merchant from the merchant's enemies. It exists to remove the merchant from the problem entirely. A shop that cannot read card numbers cannot leak them, cannot be socially engineered out of them, and cannot be sued for them, and that is a stronger security property than any amount of diligence in a shop that can.

21.98 Common wrong ideas

Wrong: an encrypting reader means nothing readable leaves the sealed box. **Right:** almost every deployment deliberately emits the first six or eight digits and the last four in clear, and many emit the expiry date and cardholder name too, because the till cannot route, surcharge or print a receipt otherwise.

Wrong: "our terminals encrypt end to end" means the merchant qualifies for SAQ P2PE. **Right:** the short questionnaire is available only where the whole arrangement has been assessed against the P2PE Standard and published on the Council's list; anything else is a non-listed encryption solution, and how much scope reduction it earns is the acquirer's decision, not the merchant's.

Wrong: the card number and the PIN are protected by the same key. **Right:** they are two separate encrypted blocks under keys derived for different usages, with different formats and different rulebooks, which is why compromising the account-data key does not hand over PINs.

Wrong: per-transaction keys mean every transaction is sealed off from every other. **Right:** derivation runs one way only, so an attacker who extracts a device's live state cannot compute yesterday's keys but can compute every future one until the device is re-injected.

Wrong: the far point of point-to-point encryption is the merchant's bank. **Right:** the decrypting box normally belongs to whoever runs the scheme — a gateway, a processor, a terminal estate manager — and the acquirer may be a further hop away and never see the encrypted form at all.

Wrong: encryption and tokenisation are two names for the same protection. **Right:** encryption protects data in flight and produces a value that changes on every transaction; tokenisation replaces the number at rest with a stable surrogate that can be stored, refunded against and billed again.

Wrong: a PCI-approved terminal is an encrypting terminal. **Right:** the encryption requirements live in Evaluation Module 4, SRED, which is optional; a device can be PTS approved without it, which is why terminal estates contain both kinds.

Wrong: the key serial number has to be kept secret. **Right:** it is an index, not a key, and its counter is defensively useful, because a healthy device's counter only ever increases and duplicates or impossible jumps are detectable.

Wrong: storing only encrypted account data always keeps a merchant out of scope. **Right:** it does so only if the merchant genuinely has no means of decrypting it; a merchant holding ciphertext and also holding, anywhere, the key or the decryption service is back in scope for that data.

Wrong: a card reader that arrives dead was badly made. **Right:** tamper-responsive devices respond to being dropped, frozen or X-rayed, and a device that erased itself in transit has done exactly its job.

21.99 Chapter summary in 20 lines

1. A card reader that arrives from the supplier already loaded with a secret turns every card number into unreadable ciphertext before the shop's own software is allowed near it.
2. There are exactly two places in the story where readable account data exists — inside the reader and inside the decrypting party's hardware security module — and neither of them is in the shop.
3. The product being bought is not the cryptography but the scope reduction: a merchant that cannot read card numbers answers a short questionnaire instead of the full PCI DSS regime.
4. The reader uses a different key for every transaction, destroys each key after use, and sends a non-secret key serial number so the far end can reconstruct the key it needs.
5. The plain version overstates the protection, because most solutions return the BIN and last four digits in clear, and often the expiry and the cardholder name as well.
6. Card data and PIN travel as two encrypted blocks under keys of different usages, so a compromise of the account-data key does not surrender PINs.
7. P2PE with a capital P is a PCI Security Standards Council programme with a standard, an assessor qualification, reporting templates and a public list, not a generic description of engineering.
8. The programme defines a solution, an application, a component and a solution provider, and the provider retains overall responsibility however much of the work it outsources.
9. Five domains carry the requirements — encryption, application, solution management, decryption and key management — with key management shared between the encrypting and decrypting sides.

10. The encryption itself is specified in SRED, which demands unique keys per device, approved algorithms, no path by which clear-text account data can leave, and resistance to side channels as well as screwdrivers.
11. What must be protected is account data, not merely the PAN, and sensitive authentication data must not be retained after authorisation at all, encrypted or otherwise.
12. DUKPT makes per-transaction keys practical without a network round trip, deriving every key of every device from a base derivation key that exists only inside the provider's HSMs.
13. The TDEA scheme uses a ten-byte KSN and yields 1,048,575 usable transactions; the AES scheme uses twelve bytes and yields 2,448,023,842, because both counters are limited by how many bits may be set rather than by their width.
14. Derivation is irreversible, so a compromised device leaks its future traffic and not its past, and the commercial consequence of a suspected extraction is a re-injection rather than an investigation.
15. Key blocks bind a key's usage, algorithm and mode to the key itself, and the phased mandate of PIN Security Requirement 18-3 in 2019, 2023 and 2025 explains a decade of integration pain.
16. Initial keys reach devices either through a physically secured injection facility under dual control and split knowledge, or by remote key distribution with mutual authentication over an untrusted network.
17. Decryption happens inside a tamper-responsive HSM that uses keys without ever revealing them, and everything the merchant no longer has to protect the solution provider protects instead, under audit and for a price.
18. Four things routinely destroy the scope reduction: an unlisted application, another route by which clear-text card numbers enter the merchant's systems, deviation from the P2PE Instruction Manual, and holding both ciphertext and the means to decrypt it.
19. Encryption and tokenisation are complementary rather than competing, and the right architecture is a listed solution for capture with an acquirer token returned for storage, so that refunds, subscriptions and analytics all run on the token.
20. None of this cryptography protects the merchant from the merchant's enemies; it removes the merchant from the problem, and a shop that cannot read card numbers cannot leak them, cannot be socially engineered out of them, and cannot be sued for them.

Sources: PCI Security Standards Council document library, including the PCI Point-to-Point Encryption Standard v3.2 (June 2025) and its summary of changes, the P2PE Program Guide v3.2, the P2PE v3.x Technical FAQs, the P2PE Solution, EMS, DMS, KMS, Application and MMS P-ROV templates, the P2PE Instruction Manual and Attestation of Validation templates, "Securing Account Data with the PCI Point-to-Point Encryption Standard v3" (July 2020), and the Assessment Guidance for Non-listed Encryption Solutions (June 2020); PCI PIN Transaction Security Point of Interaction Modular Security Requirements, Evaluation Module 4 (Secure Reading and Exchange of Data) and its section K, with version history to v7.0 (May 2025); PCI PTS Hardware Security Module Modular Security Requirements v4.0 (December 2021) and v5.0 (May 2026); PCI DSS v4.0.1 Requirements 3.3.1, 3.4.1 and 3.5.1; PCI SSC "Revisions to the Implementation Date for PCI PIN Security Requirement 18-3" (17 July 2020) and the PIN Security Requirement 18-3 Key Blocks information supplement; ANSI X9.24-1 and ANSI X9.24-3-2017, with the Supplement to ANSI X9.24-3-2017 Test Vectors published by the Accredited Standards Committee X9 for the derivation data, key usage indicators and worked key values; ANSI X9.143-2022 and ANSI X9 TR-31 for key blocks; ANSI X9.24-2 and TR-34 for asymmetric key distribution; ISO 9564-1, ISO 11568 and ISO 13491; NIST SP 800-131A Rev. 2 and the FIPS 140-3 validation programme; AWS Payment Cryptography industry terminology documentation for KSN field layouts and key type naming; IBM z/OS and Linux on Z cryptographic services documentation for the X9.24 transaction counter behaviour.

12

CHAPTER

What the Terminal Actually Sends

22.0 What this chapter gives you

1. You will be able to take any piece of a payment message and say which of the four writers — the card, the terminal, the acquirer or the issuer — put it there.
2. You will be able to explain why the terminal almost never sends the message the banks actually read, and what the acquirer's host builds from scratch on its behalf.
3. You will be able to read a field 55 tag by tag and recover the transaction from it, including the single bit that explains why it went online at all.
4. You will be able to do the byte arithmetic on field 55 and predict which tags a 255-byte or 255-character ceiling will amputate, and which of those losses are fatal.
5. You will be able to explain why a disagreement between DE 4 and tag `9F02` presents as a cryptographic failure when it is really a mapping bug in somebody's message translator.
6. You will be able to say what DE 22 claims about a transaction, and why claiming a chip read while sending no field 55 is the commonest certification failure in the industry.
7. You will be able to state what the terminal must and must not do when tag `91` is present in the response, and what it must do when it is absent.
8. You will be able to explain why a reversal has to replay the exact bytes that were sent rather than regenerate them from a database.
9. You will be able to distinguish the authorisation from the completion advice and from the clearing record, and say which of the three actually claims the money.
10. You will be able to explain why the twelve-character reference the acquirer stamps on a message today is the identifier a dispute will be argued from next summer.

At 09:45:12 on Monday 17 August 2026, in a coffee shop in Leeds, somebody buys a flat white for £4.20 and puts a card in a slot. Eleven chapters of this volume have been about what happens inside that moment: the plastic, the number, the chip, the eight bytes of mathematics it produces, the reader limits, the keypad, the encryption in the head of the terminal. This chapter is about the sentence at the end of it. The terminal has finished thinking. Something now has to leave the shop.

What leaves is about three hundred and fifty bytes. Less than a photograph of the receipt. Less than this paragraph, encoded as text. Inside those bytes are two things of quite different natures: a set of facts asserted by the machine on the counter, and a short, sealed block written by the chip, which the machine cannot read, cannot alter and does not understand. The whole of the rest of this book is the story of where those bytes go. This chapter is the story of how they are assembled, and of who wrote each one.

It is also the last chapter of Volume II, so it has a second job. Everything up to here has been about one end of the wire. From the next chapter the subject changes: four parties, a message format, a network, a settlement cycle, a dispute process. The join between the two is a single data element with a two-digit number, and getting the join wrong is the most common way for an otherwise correct chip implementation to fail certification.

The plain version

Imagine a hospital referral. Your GP writes a letter to a consultant: your name, your age, the date, what she observed, what she thinks. Stapled to that letter is a sealed envelope from the laboratory containing a test result. The GP cannot open it. She does not need to. Her job is to describe the room she was standing in; the laboratory's job is to say something the consultant can verify independently. The consultant reads both, and the value of the letter depends heavily on the fact that the envelope was sealed before it left the laboratory.

The terminal is the GP. The chip is the laboratory. The letter is the transaction: how much, when, where, which machine, how the card was read, whether anybody typed a PIN. The sealed envelope is eight bytes of cryptogram plus the exact facts the chip signed over when it made them. The bank at the other end reads the letter, opens the envelope, recomputes the mathematics with its own copy of the key, and sees whether the answer matches.

There is one more character: the courier. The card machine does not know how to reach your bank and has never heard of it. It knows exactly one address, the company the shop signed a contract with, which the trade calls the acquirer. The acquirer takes the letter, puts it in a standard envelope of a format the whole industry agrees on, writes its own reference number on the outside, adds its own return address, works out from the first few digits of the card number which bank the letter is for, and sends it.

So there are three writers, and one of the most useful habits you can build is asking, of any piece of a payment message, which of the three wrote it.

A worked example

Take the flat white. The basket is £4.20. The shop is in the United Kingdom, so the currency is sterling. The card is a chip card, inserted rather than tapped. The amount is small, so no PIN is asked for. The terminal decides, for reasons Chapter 16 set out, that it will not approve this one on its own authority, and asks the chip for the kind of cryptogram that means "ask the bank".

The chip produces its sealed block. In this transaction it is one hundred and thirty-four bytes long, and it contains twenty-one separate items. Eight of them the chip wrote. Thirteen the terminal wrote and then handed to the chip to be signed over, so that nobody in the middle can change them without the mathematics breaking. Among the thirteen are the amount, £4.20; the currency, sterling; the country, the United Kingdom; the date; the four random bytes the terminal generated to make this transaction unlike any other; and the terminal's own account of what the customer did or did not have to do to prove who they were.

Around that block, the terminal writes the letter. The parts that matter to a human read roughly as follows.

What it says	The value	Who wrote it
Amount	£4.20	The till
Currency	Pounds sterling	Terminal configuration

What it says	The value	Who wrote it
Local time and date	09:45:12 on 17 August	The terminal's clock
How the card was read	Chip, inserted, PIN pad present	The terminal
Which machine	Terminal 20194831	The acquirer, at installation
Which shop	Merchant 602948300012345	The acquirer, at contract
What kind of shop	Eating places and restaurants	The acquirer
The card number	Sixteen digits	The card
The sealed block	134 bytes	The chip, and the terminal

Then the acquirer adds its own three lines: a reference number that will follow this transaction for the next eighteen months, its own institution number so the reply knows where to come back to, and a timestamp in universal time so that two computers in different countries can agree on the order of events.

The reply arrives in under a second and is much shorter. It contains a two-character verdict, a six-character approval code that the receipt will print, and, if the bank has bothered, a second sealed envelope addressed to the chip. That second envelope is the interesting one. It lets the chip check that the approval really came from its own bank and not from someone in the middle of the wire with a convincing manner. The terminal cannot read that envelope either. It simply passes it to the chip, and the chip says yes or no.

Count the writers once more. The chip wrote eight items and sealed twenty-one. The terminal wrote thirteen of the sealed ones and about a dozen unsealed ones. The acquirer wrote the outside of the envelope. The bank wrote three lines back. Nobody wrote the whole message, and no single party in the chain could have.

Where the plain version stops being true

The terminal does not send the message the banks read

The letter analogy quietly implies that the GP's letter is the document the consultant receives. It is not. In almost every real estate, the terminal speaks a protocol of its own to the acquirer — sometimes a national standard, sometimes a vendor's own format, increasingly a web API carrying JSON — and the acquirer's host builds the industry-standard message from scratch. The terminal frequently has no idea that data elements have numbers. It sends fields with names like `amount` and `emvData`, and something in a data centre translates.

This matters more than it sounds. It means that when a specification says "the terminal sends field 55", what it actually means is that the terminal sends a blob of chip data through whatever pipe it has, and the acquirer puts that blob into field 55 without touching it. It means the person debugging a chip problem may be looking at three different representations of the same bytes. And it means that the two halves of the industry have different vocabularies for the same object: Visa's documentation calls it Field 55, Mastercard's calls it DE 55, and both are talking about the same one hundred and thirty-four bytes.

There is a stronger version of the correction. Where point-to-point encryption is in use, as in the previous chapter, the card number is not in the terminal's message at all. What the terminal sends is a ciphertext and a key serial number. The card number appears for the first time inside a hardware security module in the acquirer's data centre, and is written into the message there. The field that

everybody thinks of as the most important one in a payment message is, in those estates, authored by the acquirer.

Field 55 is not "the chip data"

It is a selection from the chip data, made by the acquirer, and about half of it was written by the terminal rather than the card.

The card gives the terminal far more than travels. Public key certificates, the issuer's public key remainder, the card's own list of acceptable ways of verifying a cardholder, its offline counters and limits, its PIN try counter, its transaction log, the templates telling the terminal how to format commands: none of that leaves the shop. What leaves is a list of tags that the acquirer's specification names, and if a tag is not on the list it does not travel however interesting it is.

The other half of the correction is the part that surprises people who have only read about EMV. Of the twenty-one objects in our one hundred and thirty-four bytes, thirteen were written by the terminal: the amount, the currency, the country, the date, the transaction type, the random number, the terminal's type and capabilities, its serial number, its account of cardholder verification. They are in the chip's sealed block not because the chip knows them but because the chip was asked to sign over them. That is the entire mechanism by which an issuer can detect that the amount was altered after the card agreed to it.

The same fact appears twice, and only one copy is protected

The amount is in the message twice: once in the ordinary amount field, and once inside the sealed block. So is the currency. So is the date, in two different formats. So is the way the card was read. So is the card's sequence number. This is not redundancy for safety; it is two systems that were designed twenty years apart and glued together.

The consequence is a rule worth memorising. Only the copy inside the sealed block is protected by the cryptogram. If the two copies disagree, the issuer's cryptogram check will validate against the inner copy and fail against the outer one, and the failure will look like a cryptographic problem when it is really a mapping bug in somebody's message translator. A great deal of expensive debugging in this industry is somebody discovering that their host was populating the outer amount from the basket total and the inner one from the pre-authorisation amount.

One message is not a transaction

The plain version tells the story as request and answer, which is how it looks from the counter. It is not how the money moves. The authorisation is a question about whether funds exist and whether the issuer consents. It moves no money and, in most schemes, is not even the record that will be settled. A second message follows, telling the acquirer that the sale actually completed and carrying the chip's final cryptogram; a third, in a batch overnight, is the one that claims the money. If the answer to the first message never arrives, a fourth kind of message has to be sent to unwind it, and that message must repeat enough of the first to name it unambiguously.

Approval is not payment. Chapter 30 is about the difference, and Chapter 49 is about what happens when somebody disputes the whole thing eleven months later and the only surviving evidence is the twelve-character reference the acquirer stamped on this message today.

The technical version

The three hops, and the protocol on each

The path from the counter to the issuer has at least three links, and each has a different standard.

The first, where the terminal is not a standalone device, is between the electronic cash register and the point of interaction. This is the link nobody certifies as a payment interface, and it carries the basket total in one direction and the receipt data in the other. In Europe it is increasingly the nexo Retailer Protocol, an ISO 20022 message set defining the exchanges between a Sale System and a POI System.

The second link is terminal to acquirer. In the United Kingdom this was standardised by APACS: Standard 60, *Message Interchange between Card Acceptor and Acquirer*, dated 2000, and its successor Standard 70, issued from 2007 in numbered books, in which Books 2 and 3 carry the message formats while the remaining books apply whichever message standard the terminal uses. Elsewhere in Europe the modern answer is the nexo Acquirer Protocol, described by nexo as a standard that "replaces ISO 8583 and its national derivatives", built from the ISO 20022 Acceptor-to-Acquirer Card Transactions catalogue. Its authorisation pair is `caaa.001.001.15`, `AcceptorAuthorisationRequestV15`, and `caaa.002`, `AcceptorAuthorisationResponse`; its completion advice is `caaa.003`. Many acquirers offer none of this and publish their own format, which in a modern gateway means a JSON document with an `emvData` string in it.

The third link is acquirer to scheme, and this is where ISO 8583 lives. The fourth, scheme to issuer, is ISO 8583 again in the scheme's own dialect, or increasingly the ISO 20022 Acquirer-to-Issuer set, whose authorisation initiation is `cain.001.001.05`.

ISO 8583, in exactly enough detail to hand over

Three facts will carry you to the next chapter.

A message begins with a four-digit **message type indicator**. `0100` is an authorisation request, `0110` its response, `0200` a financial request that both authorises and claims, `0210` its response, `0400` a reversal.

After it comes a **bitmap**, sixty-four bits, one per possible data element. A bit set to one means that element is present later in the message, in numerical order. This is why the format is compact and why it is unreadable without a specification.

Then come the **data elements** themselves, identified only by their position in the bitmap. They are numbered, not named, and the numbering is the industry's shared vocabulary: everybody in acquiring knows what DE 4 is without looking. Volume III takes this apart properly.

For now, one number matters: **55**. It is the element that carries chip data. Later editions of the standard formalise it as the integrated-circuit-card-related data element, with a three-digit length prefix and a theoretical maximum of 999 bytes; every scheme caps it far lower, and 255 bytes is the common ceiling. Crucially, the standard is explicit that its own rules for structured elements do not apply inside DE 55: the linking of sub-elements there follows ISO/IEC 7816-6. In plain terms, field 55 is a bag of EMV tag-length-value objects, parsed by EMV's rules, sitting inside a message parsed by ISO 8583's rules. It is a format inside a format, and every practitioner error in this area comes from forgetting which set of rules applies to which bytes.

What the kernel hands the terminal

Before anything is assembled, the chip's part of the transaction ends and a boundary is crossed inside the terminal itself.

In the contact world the boundary is the second GENERATE AC, or, for an online transaction, the first: the card returns the cryptogram, the counter, the cryptogram information data and its issuer

application data, in one of the two response templates of EMV Book 3. In the contactless world the boundary is explicit and named. The kernel completes and hands the terminal an outcome carrying a **Data Record** and **Discretionary Data**; EMV Contactless Book A's Annex A lists which data elements populate each. The Data Record is the set the terminal is expected to forward to the acquirer. The Discretionary Data is for the terminal's own logs and receipts and is not meant to travel.

Either way, what the terminal now holds is a pool of tags — some from the card, some of its own — and a list, supplied by the acquirer, of which ones to send.

Field 55, tag by tag, for one flat white

Here is the actual field for the £4.20 transaction. The values follow the worked cryptogram of Chapter 15 and the terminal risk management of Chapter 16, so the arithmetic joins up with what those chapters computed.

Tag	Object	Bytes	Value	Written by
9F26	Application Cryptogram	8	733084432BA24DAE	Card
9F27	Cryptogram Information Data	1	80	Card
9F10	Issuer Application Data	7	06000C03A00000	Card
9F37	Unpredictable Number	4	9A1BC3D4	Terminal
9F36	Application Transaction Counter	2	003D	Card
95	Terminal Verification Results	5	0000008000	Terminal
9A	Transaction Date	3	260817	Terminal
9C	Transaction Type	1	00	Terminal
9F02	Amount, Authorised	6	000000000420	Terminal
9F03	Amount, Other	6	000000000000	Terminal
5F2A	Transaction Currency Code	2	0826	Terminal
82	Application Interchange Profile	2	1D00	Card
9F1A	Terminal Country Code	2	0826	Terminal
9F33	Terminal Capabilities	3	E0F8C8	Terminal
9F34	CVM Results	3	1F0002	Terminal
9F35	Terminal Type	1	22	Terminal
9F1E	Interface Device Serial Number	8	3132333435363738	Terminal
84	Dedicated File Name	7	A0000000031010	Card
9F09	Application Version Number	2	008C	Terminal

Tag	Object	Bytes	Value	Written by
9F41	Transaction Sequence Counter	2	0071	Terminal
5F34	Application PAN Sequence Number	1	01	Card

Read down the values and the transaction is legible. 9F27 of 80 says the card produced an authorisation request cryptogram rather than an approval or a decline, because the top two bits of that byte carry the cryptogram type. 95 has one bit set, byte 4 bit 8, "transaction exceeds floor limit", and that single bit is the reason this transaction is online at all. 9F34 of 1F 00 02 says the terminal applied the rule "no cardholder verification required", under the condition "always", and considers it successful — a statement, as Chapter 17 laboured, not a proof. 9F10 of 06 00 0C 03A00000 tells the issuer's host which master key to load and which cryptogram version to run, and carries the card's own four-byte account of the transaction, which the schemes append to the terminal's data before recomputing the MAC. 82 has byte 1 bit 3 set, so the card is willing to be told, cryptographically, that the reply really came from its issuer. And 9F41 of 0071 is the terminal's own transaction counter: the one hundred and thirteenth sale on that machine since it was last reset.

Encoded, the whole field is:

```
9F2608733084432BA24DAE 9F270180 9F100706000C03A00000
9F37049A1BC3D4 9F3602003D 95050000008000 9A03260817 9C0100
9F0206000000000420 9F0306000000000000 5F2A020826 82021D00
9F1A020826 9F3303E0F8C8 9F34031F0002 9F350122
9F1E083132333435363738 8407A0000000031010 9F0902008C
9F41020071 5F340101
```

One hundred and thirty-four bytes. Add the tags and lengths and there is no slack in it: 9F26 costs eleven bytes to carry eight, because a two-byte tag and a one-byte length ride with every object.

The byte budget, and what truncation does

That is the arithmetic nobody does until it bites. Twenty-one objects, one hundred and thirty-four bytes, against a scheme ceiling of 255. Comfortable. Now add the tags that a contactless transaction wants — the terminal transaction qualifiers, the card transaction qualifiers, the form factor indicator — and the tags a tokenised transaction wants, and the issuer script results after a script has run, and a kernel's proprietary data objects, and the budget tightens quickly.

Two encodings make it tighter still. If the interface between the terminal and the acquirer is character-based and carries the field as hexadecimal text, one hundred and thirty-four bytes become two hundred and sixty-eight characters. Any component in that chain with a 255-character limit will silently cut the field in the middle of a tag. And a field 55 truncated at the end loses its trailing objects first, which in the list above means 5F34, 9F41 and 9F09 — the least interesting three. Reorder the same list, as BER-TLV entitles you to, and truncation eats 9F10 instead, at which point the issuer cannot select a key, cannot derive a session key, cannot verify the cryptogram, and declines with a code that says nothing about why.

Which tags an acquirer requires is published, and reading that list is the first task of any integration. One large processor's public specification for encrypted device integration, for instance, marks as mandatory 4F, 5F2A, 5F34, 82, 84, 95, 9A, 9C, 9F02, 9F03, 9F09, 9F10, 9F1A, 9F26, 9F27, 9F33, 9F34,

9F35, 9F36 and 9F37, with 9F1E, 9F41, 50, 9F5B and 9F6E optional, and in the response 8A, 91, 71 and 72. That shape — the card's cryptogram set, the terminal's signed context, the identity of the application — is common to every scheme, and the differences between acquirers are at the edges.

Read those lists sceptically. The same document describes tag 9F6E as the application transaction counter, which it is not, and tag 9F1A as a two-letter alphabetic country code, which it is not either: 9F1A is three numeric digits in two bytes, 0826 for the United Kingdom. Vendor documentation copies vendor documentation. The primary sources are EMV Book 3's Annex A for what a tag is, and your acquirer's own message specification for whether they want it.

What the terminal adds outside field 55

The rest of the terminal's contribution is the ordinary transaction, and its home in a scheme's authorisation request looks like this.

DE	Element	Value here	Where the value comes from
2	Primary account number	4111111111111111	Card, or the acquirer's decryption HSM
3	Processing code	000000	Transaction type at the till
4	Amount, transaction	000000000420	The basket
11	System trace audit number	004871	Message originator
12	Local transaction time	094512	Terminal clock, local
13	Local transaction date	0817	Terminal clock, local
14	Expiration date	2903	Card
18	Merchant type	5812	Acquirer's record of the merchant
22	Point of service entry mode	051	Terminal
23	Application PAN sequence number	001	Card, from tag 5F34
25	Point of service condition code	00	Transaction context
35	Track 2 data	From tag 57	Card
41	Card acceptor terminal identification	20194831	Acquirer, at installation
42	Card acceptor identification code	602948300012345	Acquirer, at contract
43	Card acceptor name and location	40 characters	Acquirer
49	Currency code, transaction	826	Terminal configuration
55	Integrated circuit card data	134 bytes	Card and terminal

Several rows deserve a note.

DE 22 is three digits in the 1987-format messages: two for how the account number was captured and one for the terminal's PIN entry capability. 05 is contact chip, 07 contactless chip, 91 the legacy contactless stripe mode, 80 a fallback to magnetic stripe after the chip failed. Ours is 051: chip read, PIN pad present. This field is where the commercial consequences live. Claim 05 and send no field

55 and you have failed a scheme certification test in the most common way there is. Claim `07` after performing a stripe read and the issuer looks for a cryptogram that is not there. And the third digit must be true: a terminal that claims PIN capability and never sends a PIN block invites an issuer to distrust everything else it says.

DE 23 is three digits carrying what the chip holds in one byte, right-justified and zero-filled. `5F34` of `01` becomes `001`. This is the field most often left out, and the failure it causes is specific: an issuer with two cards live on one account cannot tell which one it is being asked about, so it cannot select the right master key.

DE 35 carries the track 2 equivalent data the chip returned in tag `57` — not a stripe read, but the same layout, including a service code whose first digit of `2` or `6` announces that this is a chip card. The separator the chip encodes as hexadecimal `D` appears as `=` on character-based interfaces, and a translator that fails to swap it produces a field the issuer rejects for format.

DE 12 and DE 13 are local. **DE 7**, which the acquirer writes, is universal time. Our transaction at 09:45:12 British Summer Time is `094512` in DE 12 and `0817084512` in DE 7. Neither DE 12 nor DE 13 carries a year, which is a real problem exactly once a year.

DE 41 and DE 42 are not the terminal's opinion. They are identifiers assigned by the acquirer and loaded into the terminal at installation, which is why a terminal moved between shops without reconfiguration produces transactions that reconcile to the wrong merchant and are extremely tedious to unpick.

The PIN, when there is one

Our flat white needed no cardholder verification. Had it been a £47.50 purchase requiring online PIN, one more field would have appeared and it would not have come from the chip at all.

The PIN, captured on the approved keypad, is formatted into an ISO 9564-1 PIN block and encrypted under a key shared with the acquirer. In the 1987-format messages it occupies **DE 52**, defined as sixty-four bits — exactly one Triple DES block, which is why formats 0 to 3 are eight bytes long. It travels under a different key from the card data, with the key management information alongside it in **DE 53**, whose layout differs in every scheme's dialect. At each hop between the terminal and the issuer's own hardware security module the block is translated: decrypted and re-encrypted under the next zone's key, inside a secure cryptographic device, never in the clear on any bus. That regime is the PCI PIN Security Requirements, and it is older and stricter than anything governing the card number.

One seam is worth flagging now because it is a live migration problem. ISO 9564-1's format 4 block, introduced for AES, is sixteen bytes. A sixty-four-bit field cannot hold it. Carrying AES PIN blocks therefore requires arrangements outside the original definition of DE 52, and they are specified by each scheme rather than by ISO 8583. If you are planning an AES migration, that is a question for your acquirer's message specification and not for the standard.

Note also what the PIN is not. If the cardholder verification was offline PIN, nothing about the PIN appears in the authorisation message at all. The only trace is `9F34` saying it happened, and the card's own account inside `9F10` saying whether it agrees.

What the acquirer adds

The acquirer's host receives the terminal's message and builds the scheme message. Its additions fall into four groups.

Identity and routing. The acquiring institution identification code in **DE 32**, and a forwarding institution code in **DE 33** where an intermediary is in the path. The acquirer also does the routing itself: the leading digits of the account number are looked up in a table to decide which scheme, and therefore which network connection, this message belongs to.

Timekeeping and identifiers. DE 7, transmission date and time, in universal time, rewritten at every hop by whoever is sending. **DE 37**, the retrieval reference number, twelve alphanumeric characters, assigned by the acquirer and echoed unchanged by everybody downstream. Its internal construction is not fixed by the standard; a common convention is year, day of year, hour and sequence, which is how `622908004871` was built for this transaction: year 6, day 229 — 17 August 2026 — hour 08 universal time, sequence 004871. That string is the identifier with the longest life in the entire message. It will still be the key in a chargeback file next summer.

Merchant description. DE 18, the merchant category code — `5812`, eating places and restaurants, from ISO 18245 — and **DE 43**, forty characters of name and location, conventionally twenty-three of trading name, thirteen of town, two of region and two of country. Both are the acquirer's records rather than the terminal's, and the second is what appears on the cardholder's statement, which makes it the single most effective fraud-prevention field in the message and the one most likely to be wrong.

Cryptography and scheme dialect. The account data, if it arrived encrypted, is decrypted in a hardware security module and written into **DE 2** and **DE 35**. The PIN block, if present, is translated. A message authentication code over the whole message may be computed and placed in **DE 64**. And the scheme's own private data elements are populated: Mastercard's DE 61 point-of-service data, DE 48 additional data, DE 63 network data, and their equivalents elsewhere. Those are the fields where scheme dialects diverge most, and a field number learned on one network must never be carried across to another.

Two things the acquirer does that are not fields at all. It applies its own duplicate detection and velocity rules before the message leaves, which is why an acquirer can decline a transaction the issuer never saw. And it keeps a record of what it sent, because if the response does not arrive it is the acquirer, not the terminal, that must reverse.

The assembled message

Put it together and the `0100` for our flat white is a four-digit message type indicator, an eight-byte bitmap with seventeen bits set, and the elements in numerical order. In character encoding with a binary field 55, it is a little over three hundred and fifty bytes. Rendering field 55 as hexadecimal text pushes it near five hundred. Packing the numerics as binary-coded decimal brings it well under three hundred.

American Express's *Global Credit Authorization Guide*, written to the 1993 edition of the standard, states the chip requirement plainly: ICC system related data in data field 55 must be present, alongside track 1 in data field 45 and/or track 2 in data field 35, and a point of service data code in data field 22. Visa's *Transaction Acceptance Device Guide* puts the division of labour equally plainly from the other side: the acquirer formats the authorisation message with the chip data in Field 55. Both sentences say the same thing. The terminal produces chip data; somebody else builds the message; the two must agree about what the transaction was.

The response, and what the terminal does with it

The `0110` comes back with three things that matter and one that matters to the chip.

DE 39, the response code, two alphanumeric characters in the 1987 edition and three in the 1993 edition: `00` approved. **DE 38**, the authorisation identification response, six characters, which the receipt prints and the clearing record carries. And **DE 55** again, now populated by the issuer rather than the terminal.

In the response, field 55 carries up to three objects.

Tag	Object	Purpose
`91`	Issuer Authentication Data	Proves to the chip that the issuer answered
`71`	Issuer Script Template 1	Commands for the chip, before the final cryptogram
`72`	Issuer Script Template 2	Commands for the chip, after it

For our transaction, under ARPC Method 1 and with the two ASCII characters `0` and `0` as the authorisation response code, Chapter 15 computed the issuer's cryptogram as `735C24FD87F1BF0E`. The issuer authentication data is that cryptogram followed by the response code, so the object on the wire is `91 0A 735C24FD87F1BF0E3030`: ten bytes of value, twelve on the wire.

What the terminal does next is not optional and is where implementations diverge from the specification most often. If `91` is present, the terminal delivers it to the card, either through EXTERNAL AUTHENTICATE or inside the second GENERATE AC, and sets the issuer-authentication bit in the transaction status information. If `91` is absent, the terminal must not attempt issuer authentication, must leave that status bit at zero, must leave the corresponding failure bit in the terminal verification results clear, and must proceed on the response code alone. A downgraded authorisation is not a failed one.

Then the terminal asks the card once more, using the second data object list, for a completion cryptogram, and passes it the two-byte authorisation response code in tag `8A` — which it must pass through exactly as received and must never invent when it came from a response message. The card runs its own second round of risk management and answers with a transaction certificate if it is content or a decline cryptogram if it is not. A card that declines after an approved authorisation has almost always either failed to verify the issuer's cryptogram or verified it and found the approval bit clear.

Where scripts were sent, the terminal delivers them, records what happened in tag `9F5B`, the issuer script results, and sets the script-processing bit in the transaction status information. Kernel 3 is relaxed about how much of `9F5B` travels: acquirer documentation quoting it notes that it is acceptable to transmit only byte 1, though preferable to send all five.

For our transaction the transaction status information ends as `B8 00`: offline data authentication performed, card risk management performed, issuer authentication performed, terminal risk management performed, no scripts. That two-byte object, and the final cryptogram, belong to the next message rather than this one.

The second and third messages

The authorisation is finished and nothing has been paid.

What follows depends on the scheme and the acquirer. In a dual-message model the terminal or its host sends a completion advice — `caaa.003` in the nexo set, a scheme-specific advice or an entry in the overnight batch elsewhere — carrying the transaction certificate in the same tag `9F26`, the cryptogram information data now showing a transaction certificate rather than a request, the transaction status information, and the script results. That message is what tells the acquirer the sale actually happened. The clearing record built from it is what claims the money, and it is the record a dispute will be argued from.

In a single-message model the `0200` did both jobs at once, and the chip data requirements are the same.

And if the response never arrived, the acquirer must reverse. A reversal names its victim by repeating the original message type indicator, system trace audit number, transmission date and time and institution codes, and acquirer specifications commonly require that the original field 55 be repeated in the reversal too. This is the practical reason for a rule that sounds pedantic: keep the exact bytes you sent, not a re-derivation of them. A reversal built by regenerating field 55 from a database will differ from the original in some small way and will fail to match.

A short field guide to failures

DE 22 says chip and field 55 is absent. The commonest certification failure in the industry. The message claims a chip was used and contains no evidence of one.

Field 55 truncated. Trailing tags vanish. If `9F10` was among them the issuer cannot select a key and declines uninformatively. Check the length before suspecting the cryptogram.

Hexadecimal where binary was expected, or the reverse. Every length doubles or halves and parsing fails at the first tag. This is the single most common integration fault on a new acquirer interface.

Flat parser meets a constructed tag. `71` and `72` contain further tag-length-value objects. A parser that treats them as opaque will deliver an issuer script the card cannot read.

Amount mismatch between DE 4 and `9F02`. The cryptogram validates against `9F02`. If the two disagree the issuer sees evidence of tampering, which is exactly what the design intends, and the tampering is your own message translator.

Card sequence number omitted. Two cards on one account; the issuer loads the wrong master key; cryptogram verification fails for a card that is entirely genuine.

Response code altered before the second GENERATE AC. A host that normalises `00` into something friendlier for its own logs, and passes the friendly version to the card, breaks issuer authentication and gets a decline cryptogram after an approval.

Reversal rebuilt rather than replayed. The original and the reversal differ in one byte and never match. This is how transactions become stuck: authorised, never completed, never reversed, and visible to the cardholder as a held balance for a week.

What you now know, and what Volume III answers

Volume II has been a single argument, and it is worth stating in one place before the subject changes.

A card is a physical object carrying an account number whose structure is public and whose last digit is arithmetic. The number alone is a bearer credential and always has been, which is why the printed codes, the stripe codes and the chip codes are four different things with four different threat models.

The chip is a computer running a specified application, and its value is not that it stores a secret but that it computes one: eight bytes that depend on a key never transmitted, a counter that only rises, and the exact terms of this transaction. Whether it goes online is decided by bit arithmetic between the terminal's observations and the issuer's declared policies, both held as five-byte lists. Whether the cardholder was verified is a claim the terminal writes down, not a fact the network can check. Contactless is the same machinery with different limits and a different kernel. A phone holds not your card number but a separate credential minted for that device. The terminal is a certified object whose configuration is auditable after the fact, and in a well-built estate it can encrypt the account number so thoroughly that the shop itself never sees it. And all of it converges on the message this chapter assembled: three hundred and fifty-odd bytes, one hundred and thirty-four of them a sealed statement the shop cannot read.

Volume III begins the moment that message leaves the acquirer's building, and answers the questions this chapter has deliberately left open.

Question	Chapter
Who are the parties, and who pays whom?	23, The Four-Party Model
What is ISO 8583, precisely, and why does it survive?	24
How does a bitmap work, and why not text?	25
What does each field mean, field by field?	26
What does one whole transaction look like, end to end?	27
What is the six-character approval code actually worth?	28
What does a decline code mean, and what may you retry?	29
Where does authorisation end and payment begin?	30
Where does the £4.20 go, and how much of it arrives?	31

The terminal's job ends with a message. The rest of this book is about what a message is worth.

22.98 Common wrong ideas

Wrong: the terminal builds the message the banks read. **Right:** in almost every real estate the terminal speaks a protocol of its own — a national standard, a vendor format, increasingly a web API carrying JSON — and the acquirer's host builds the industry-standard message from scratch.

Wrong: field 55 is the chip data. **Right:** it is a selection from the chip data, named by the acquirer's specification, and about half of it was written by the terminal rather than the card.

Wrong: the card number is always the card's contribution to the message. **Right:** where point-to-point encryption is in use the terminal sends ciphertext and a key serial number, and DE 2 is authored inside a hardware security module in the acquirer's data centre.

Wrong: the amount appearing twice is harmless redundancy. **Right:** only the copy inside the sealed block is protected by the cryptogram, so if the two disagree the issuer sees evidence of tampering, and the tampering is your own message translator.

Wrong: an approval means the money is on its way. **Right:** the authorisation moves nothing and in most schemes is not even the record that will be settled; a completion advice follows, and a clearing record is what claims the money.

Wrong: a missing tag `91` means issuer authentication failed. **Right:** if `91` is absent the terminal must not attempt issuer authentication, must leave the status and failure bits clear, and must proceed on the response code alone, because a downgraded authorisation is not a failed one.

Wrong: the card sequence number is an optional nicety. **Right:** it is the field most often left out, and without it an issuer with two cards live on one account cannot select the right master key, so cryptogram verification fails for a card that is entirely genuine.

Wrong: a host may tidy the authorisation response code before handing it to the card. **Right:** tag `8A` must be passed through exactly as received and never invented, because a host that normalises `00` into something friendlier breaks issuer authentication and gets a decline cryptogram after an approval.

Wrong: tags may be reordered freely inside field 55 because BER-TLV permits it. **Right:** it does permit it, and the order decides what truncation eats first; lose `9F10` and the issuer cannot select a key, cannot derive a session key, and declines uninformatively.

Wrong: vendor tag documentation is a primary source. **Right:** vendor documentation copies vendor documentation, and published specifications in circulation describe `9F6E` as the application transaction counter and `9F1A` as a two-letter alphabetic country code, neither of which is true.

22.99 Chapter summary in 20 lines

1. About three hundred and fifty bytes leave the shop, and they contain two things of quite different natures: facts asserted by the machine on the counter, and a sealed block written by the chip.
2. Three parties write the outbound message — the chip, the terminal and the acquirer — and a fourth, the issuer, writes the reply.
3. For a £4.20 flat white the chip's sealed block is one hundred and thirty-four bytes holding twenty-one objects, eight of them written by the card.
4. Thirteen of those objects were written by the terminal and merely signed over by the chip, which is the entire mechanism by which an issuer detects that an amount was altered after the card agreed to it.
5. The terminal usually knows nothing about numbered data elements; it sends named fields down whatever pipe it has, and something in a data centre translates.
6. Field 55 is a bag of EMV tag-length-value objects parsed by EMV's rules, sitting inside a message parsed by ISO 8583's rules, and most practitioner errors here come from applying the wrong rule-book to the wrong bytes.
7. Reading the tags down the list makes the transaction legible: `9F27` of `80` says the card asked the bank, and one bit of `95` says the amount exceeded the floor limit, which is the only reason this sale went online.
8. Several facts appear twice, inside and outside the sealed block, because two systems designed twenty years apart were glued together, and only the inner copy is cryptographically protected.
9. The byte budget is real: one hundred and thirty-four bytes against a 255-byte ceiling is comfortable until contactless, tokenisation and script results are added, and hexadecimal encoding doubles everything at a stroke.
10. Truncation removes the trailing objects first, so the order in which tags are written decides whether the loss is trivial or fatal.
11. Outside field 55 the terminal contributes the ordinary transaction fields, while the terminal and merchant identifiers were assigned by the acquirer at installation and at contract.

12. DE 22 declares how the card was read and whether a PIN pad is present, and every digit of it must be true, because claiming a chip read with no field 55 is the commonest certification failure there is.
13. Where an online PIN exists it travels in DE 52 under a different key from the card data, translated inside a secure cryptographic device at every hop under the PCI PIN Security Requirements.
14. Offline PIN leaves no trace in the authorisation at all beyond the terminal's claim in `9F34` and the card's own account of it inside `9F10`.
15. The acquirer adds identity and routing, universal time, the retrieval reference number, the merchant category code and statement descriptor, and the scheme's own private data elements.
16. That twelve-character retrieval reference number has the longest life of anything in the message and will still be the key in a chargeback file next summer.
17. The response carries a two-character verdict, a six-character approval code, and optionally a second sealed object addressed to the chip that the terminal cannot read and simply passes on.
18. If tag `91` is present the terminal must deliver it to the card and set the issuer-authentication bit; if it is absent it must do neither and rely on the response code.
19. The authorisation then ends with nothing paid: a completion advice carries the final cryptogram, a clearing record claims the money, and a reversal — replayed byte for byte, never rebuilt — unwinds a request whose answer never came.
20. Volume II closes here because everything before it concerned one end of the wire, and everything after it concerns what a message is worth once it leaves the acquirer's building.

Sources: EMV Integrated Circuit Card Specifications for Payment Systems, Books 2, 3 and 4, in particular Book 2 sections 8.1 and 8.2 for cryptogram and ARPC generation, Book 3 Annex A for the data element dictionary and Annex C for the terminal verification results and cardholder verification coding, and Book 4 section 12 for online processing and issuer authentication; EMV Contactless Specifications for Payment Systems, Book A Annex A for the Data Record and Discretionary Data, and Book C-3 for Kernel 3's issuer script results requirement; EMV Payment Tokenisation Specification – Technical Framework for token carriage; ISO 8583 and its 1987, 1993, 2003 and 2023 editions for data element definitions and for the statement that DE 55 sub-element linking follows ISO/IEC 7816-6; ISO/IEC 7816-4 and 7816-6; ISO/IEC 7813 for track 2 and the service code; ISO 4217 and ISO 18245; ISO 9564-1:2017 for PIN block formats; ISO 20022 message catalogue, Acceptor to Acquirer Card Transactions (caaa.001.001.15 and its siblings) and Acquirer to Issuer Card Transactions (cain.001.001.05), with nexo standards' own descriptions of the nexo Acquirer and Retailer Protocols; APACS Standard 60:2000, Message Interchange between Card Acceptor and Acquirer, and Standard 70 Books 1 to 7 (2007); Visa Transaction Acceptance Device Guide v3.3; American Express Global Credit Authorization Guide (ISO 8583:1993); Mastercard Transaction Processing Rules; Worldpay ISO 8583 Reference Guide v2.51 and 610 Interface Reference Guide v2.56 for a published acquirer dialect; Fiserv encrypted device integration documentation for a published mandatory and optional tag list, cited also as an example of vendor documentation misdescribing tags 9F6E and 9F1A; PCI PIN Security Requirements v3.1 and PCI PTS POI v7.0 for PIN capture and translation. Tag names, formats and lengths cross-checked against EFTlab's EMV and NFC tag list, neapay's tag reference and emvlab.org.

ABOUT THE AUTHOR

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

Shikhar Singh is the Founder & Chairman of KedByte Technologies Private Limited.

He wrote this book from the position of someone building payment software rather than someone commentating on it, and the difference shows in what it chooses to explain. The chapters that go deepest are the ones where he has watched capable engineers lose days: the gap between authorisation and settlement, the several identifiers all called "the transaction number", and reconciliation, which is unglamorous, unavoidable, and where more young financial technology companies fail than for any more interesting reason.

His conviction is that payments has no genuinely hard idea in it, only a very long chain of simple ones, most of which are documented for people who already understand them. The barrier is not difficulty. It is that the industry writes for itself. That is a solvable problem, and this book is an attempt at solving it.

Under his direction, KedByte Technologies Private Limited treats technical education as infrastructure rather than as a product feature: knowledge that should be transferable, verifiable, and free of the vocabulary barriers that keep capable people out of the field.

ABOUT THE PUBLISHER

KedByte Technologies Private Limited

KedByte Technologies Private Limited publishes technical work under the KedByte Press imprint.

The editorial standard applied to this volume is simple to state and difficult to meet. Every explanation must work twice: once for a reader who has never opened a terminal, and once for an engineer who needs the exact figure. Every claim that can be checked must be checked. Every figure that will go stale must carry a date. Where the field genuinely disagrees, both positions are given rather than one being quietly chosen.

Where this book uses real evidence, it uses it unaltered. The network trace in Part F, the failed pushes in Part G and the error messages quoted throughout are reproduced exactly as they were observed. Diagnosis is taught against ambiguous real data, because that is the only kind there is.

KEDBYTE

TECHNOLOGIES PRIVATE LIMITED

COLOPHON

This first edition of *How Money Moves* runs to five volumes and fifty-four chapters, and was set entirely from plain text.

The body text is set in TeX Gyre Pagella, a digital revival of Hermann Zapf's Palatino, chosen for long-form reading. Headings, tables and the KedByte identity are set in Poppins. Code, command output and diagrams are set in DejaVu Sans Mono, and every diagram in this book is drawn in plain ASCII so that it survives being copied anywhere.

The book was composed with pandoc and typeset with XeLaTeX on A4 stock. The single accent colour used on rules, chapter numbers and section headings is KedByte navy.

Shikhar Singh

Founder & Chairman, KedByte Technologies Private Limited

First Edition • KedByte Press