

THE VAULT

*How a neighbourhood can run its own platform — with
no one in the middle*

By

MARTIN CARR

INDEPENDENT PUBLISHER

Copyright © 2026 by Martin Carr.

All rights reserved.

No part of this book may be reproduced, digitalized, or transmitted in any form without written permission from the publisher, except in the case of brief quotations embodied in critical reviews.

Publisher: Independent Publisher

ISBN Placeholder

TABLE OF CONTENTS

Who this book is for	4
What this book is about	6
The road ahead	9
How to read it	11
Part One — The Idea	12
The Landlord	13
Part Two — The Node and the Chip	22
A Box That Can't Rewrite the Past	23
The Five-Pound Conscience	29
Part Three — The Neighbourhood	37
Nobody in Charge	38
The Front Door	46
Part Four — What Runs On Top	59
The Two Percent Platform	60
Where It Goes	67
What's Built, What Isn't, and What You Could Build	73

CHAPTER 1

WHO THIS BOOK IS FOR

You have noticed that almost everything you do online now passes through somebody's platform, and that the platform takes a cut, keeps a record, and could shut you out tomorrow without explaining why. You suspect it doesn't have to be this way. You can read a little code — the examples are Python, and they run on an ordinary computer or a £80 Raspberry Pi — but you don't need to be an engineer. Where a chapter leans on an idea, it builds it first.

What you need to bring is what every builder brings: a willingness to run the thing, poke it, break it, and see for yourself. The core of this book is real. It

ships

alongside as a working prototype you can run, tamper with, and take apart. When I show

you a node proving a transaction, or two nodes catching a forger, that is not a diagram

of something that might work — it is a program that runs, and I'll tell you its name so

you can run it too.

CHAPTER 2

WHAT THIS BOOK IS ABOUT

Here is an uncomfortable fact underneath modern life: you cannot pay, sell, ride, or publish without a middleman. When you take a taxi, the ride is between you and a driver two feet away — but the money, the record, and the trust all route through a company on another continent that takes a fifth of the fare and owns the account it lives in. When you sell something, when you tip a creator, when you order food, when you install an app: a platform sits in the middle of a thing that was fundamentally between two local people, and it charges rent, keeps the ledger, and holds the off switch.

We've grown so used to this that it looks like the weather. It isn't. It's a structure, and it was chosen. Vault OS is a bet that a different structure is possible: that a neighbourhood can run its own node — a small computer in the corner of a shop — that hosts its own transactions, proves they really happened with a cheap chip, and keeps the value and the record local. No central operator. No landlord in the middle. The taxi fare stays in the town. The record can't be quietly rewritten. And the node answers to the person who runs it, not to a company you've never met.

That's the idea. This book takes it from the smallest possible demonstration — one node writing down a transaction that can't be secretly altered — all the way to a mesh of nodes across a village, checking each other's honesty over radio when the internet is down, and a marketplace running on top where the platform's cut is 2% instead of 30% because there's no landlord to feed. Along the way you'll see that the hard part was never the app. It's trust without a boss — and that a five-pound chip,

not a
corporation, turns out to be enough.

I build this kind of thing on my own, on normal hardware, because
I've become convinced
the interesting frontier isn't a bigger platform — it's how much of
one person's digital
life can be handed back to them, running on hardware they own, with
nobody renting them
the middle. This book hands you a working piece of that, and shows
you where it goes.

CHAPTER 3

THE ROAD AHEAD

1. The idea. Why needing a platform is a cage, and how a community can host itself

instead. The four bars, and the door Bitcoin cracked open — then left half-shut.

2. The node and the chip. What a node actually is (a Pi in the corner of a shop),

how it writes down transactions that can't be quietly rewritten, and how a five-pound

TPM lets it prove that to a stranger. You'll run the ledger and watch it catch a

tamper.

3. The neighbours. The heart of the book. One node you have to trust; many nodes

that check each other so you don't. Co-attestation, quorum, and a mesh that keeps

working over radio when the internet dies. You'll watch nodes
catch a forger with no
one in charge.

4. The front door. The last power standing: who lets a new node in?
You'll watch a

total stranger earn membership from the neighbourhood itself —
admitted by any k

members, on probation until proven, and evicted by pure arithmetic
if it cheats,

founders included. No doorman anywhere.

5. What runs on top, and where it goes. The marketplace, the 2% app
store, the phone

that shows no ads — and the honest map of what's built, what's a
drawing, and what you
could build next.

HOW TO READ IT

At the keyboard, with the prototype open next to you. When you reach a program, run it before you read my explanation. Guess what it'll do. Change a number — inflate a fare, forge a certificate — and watch what breaks. The reader who fiddles understands twice as fast as the reader who nods along.

Let's begin with the landlord, and how to get out from under him.

PART ONE — THE IDEA

• • •

CHAPTER 4

THE LANDLORD

Picture a taxi ride. You stand on a corner; a driver pulls up; you get in, you go three miles, you get out. Two people, one car, one short journey between them. Now look at what actually happens to make that ride count — to book it, price it, pay for it, and record that it happened:

Your request goes up to a cell tower, across to a data centre in another country, into a company's servers, which match you to the driver, set the price, take the payment from a card issued by a bank, route a cut to themselves, hold the record of the trip in an account they own, and finally send the driver a number. The driver,

sitting three feet
from you, learns you exist via a round trip through machinery on
another continent owned
by a company neither of you has ever met — and that company
keeps roughly a fifth of the
fare for standing in the middle of it.

We've got so used to this that it looks normal. It is not normal. It is a
cage, and
it's worth being precise about the bars, because Vault OS is an
attempt to file through
every one of them.

THE FOUR BARS

The platform. The ride only happens if the company's systems are up
and willing. The
app, the matching, the pricing, the payment rails — all of it lives on
infrastructure
that has nothing to do with the two people in the car. If the platform
is down, or has
decided your town isn't worth serving, two people ready to do
business simply can't. The
ability of a driver and a rider to transact depends entirely on a third
party who isn't
there.

The cut. Somebody takes a slice of every transaction for occupying the middle — a fifth, sometimes a third. It comes out of the driver's pay and the rider's pocket both, and it leaves the town entirely. A local ride, local food, a local sale: the money makes a trip to a corporate account far away and comes back smaller. Multiply that across every fare, every meal, every app install in a place, and you're watching a community's economy quietly drained through a pipe it doesn't own.

The record. For any of this to work, someone has to keep the books — who rode, who paid, who earned, what happened. That someone is always the platform, and the ledger is always theirs. You can't see the whole of it, you can't take it with you, and you can't prove your side of a dispute without their permission. Your history — the thing that makes you trustworthy to the next customer — is held hostage in a database you'll never touch.

The off switch. Because they own the platform, the cut, and the record, they own you.

They can suspend the driver by algorithm at 2am with no appeal.

They can de-list the shop.

They can change the rules, the price, the terms, and your only options are accept or

vanish. Coordination between two willing local people has become a permission granted

by a distant company, and permission can always be revoked.

Here's the thing that made me start pulling at this. The first time I felt the shape of a

different world was watching early Bitcoin — not the money, the structure. The

astonishing part wasn't the coin; it was that there was no central operator. No

company you had to ask. No server that, switched off, took the whole thing with it. A

system that just... ran, between its participants, with nobody in charge of the middle.

It felt like a door opening.

But Bitcoin only decentralised money, and only the ledger of it. It left the entire

platform — the app you book the ride in, the marketplace you sell in, the store you

install from, the record of your reputation — exactly as caged as before. You can now

send value with no bank in the middle, and then you go and spend your whole digital life inside somebody's walled platform anyway. Which raises the obvious question, the one this book is chasing: can you decentralise the platform itself? Can a driver and a rider, a shop and a customer, transact directly — booked, priced, paid, and recorded — with no company standing in the middle to take a cut, keep the book, or hold the off switch?

WHAT'S ACTUALLY NEEDED

Go back to the two people and the taxi. Strip away everything we've been told is necessary. What does that transaction genuinely require?

It needs a way to agree — you want the ride, the driver will give it, at a price. It needs a way to pay. And it needs a way to remember honestly — a record both sides trust, so tomorrow's customer knows this driver is real and this rider paid. That's it. Agreement, payment, a trustworthy memory.

None of those three things inherently requires a company on another

continent. Humans ran markets for ten thousand years with a handshake and a witness. The handshake is the agreement. The coin is the payment. And the witness — the neighbour who saw the deal and would vouch it happened — is the trustworthy memory. Villages didn't need a head office to know who owed whom; they had each other, and a shared, stubborn memory of what had passed.

That is the entire idea of Vault OS. Put the witness back in the village. Give a neighbourhood a node — a small, cheap computer someone local runs — that hosts the transactions, keeps the record, and can prove that record is honest to anyone who asks.

Let many such nodes check each other, so no single one has to be trusted. Strip the platform back to what it actually is — agreement, payment, and an honest memory — and put all three where the transaction really happens: locally, between the people doing it.

WHY A NODE

Of all the ways to do this, why a little box in the corner of a shop?

Because it is the most stubbornly ownable thing there is. A

Raspberry Pi costs less than

a night out. It draws less power than a lightbulb. It sits on a shelf in a café, a corner

shop, a community centre — somewhere with a name and a face, run by someone in the town.

It needs no permission from anyone to switch on. And crucially, because it is there,

physically, owned by a local person, the value it hosts and the record it keeps stay

where the transaction happened, instead of being sucked down a pipe to somewhere else.

A node is honest about its limits, too, which turns out to be a feature.

One box on a

shelf could lie, or break, or be tampered with by the very person who runs it. We are not

going to hide from that — in fact it's exactly where the interesting engineering lives,

and it's the first hard problem we'll meet in code. The answer is not to trust the box.

The answer is a chip that makes the box unable to lie without being caught, and a

neighbourhood of boxes that catch each other. How you stay

trustworthy without a boss
is most of the story.

WHAT WE'RE GOING TO BUILD

We'll start as small as it gets: one node, writing down transactions in a way that can't be quietly rewritten, and the question of whether a tamper gets noticed. From there we'll add the chip that lets one node prove its honesty to a stranger, and discover why that single five-pound part is what replaces the corporation in the middle.

Then we'll do the thing that matters most. We'll put several nodes in a neighbourhood, let them check each other's records, and watch them agree that a transaction is real without any one of them being in charge — and catch a forger who tries to sneak in. We'll take away the internet and watch them keep working over radio. And only once that honest memory is standing will we put a marketplace on top of it, and see why the economics come out so differently when there's no landlord to feed.

The cage has four bars: the platform, the cut, the record, and the off

switch. By the end
of this book you'll have watched all four fall away, in code you can
run yourself. Turn
the page, and let's put a node on the shelf.

PART TWO — THE NODE AND THE CHIP

. . . .

CHAPTER 5

A BOX THAT CAN'T REWRITE THE PAST

Let's build the witness. Not a metaphor — an actual program, running on an actual small computer, that hosts transactions for a neighbourhood and keeps a record. The prototype that ships with this book is a folder called `vault-ledger`, pure Python, and the first program to run is `demo_two_nodes.py`. But before you run it, let's be clear about the one job this box has, because everything rests on it.

The job is an honest memory. When a taxi fare, a shop sale, or a tip to a creator happens, the node writes it down. The record has to be two things at once: complete (you can add new events) and unchangeable (you cannot quietly alter or

delete an old one).

Those pull against each other — an ordinary database lets you edit any row — and resolving them is the first real trick.

THE CHAIN

Here's the move, and it's the same one Bitcoin uses, minus the mining. Each transaction the node records is not just stored; it's linked to the one before it by a fingerprint.

A fingerprint here is a hash — a short string of characters computed from some data, with a magic property: change the data by even one penny, and the fingerprint changes completely and unpredictably. You can't work backwards from a fingerprint to the data, and you can't find two different pieces of data with the same fingerprint. It's a seal that's easy to check and practically impossible to fake.

Now the trick. Every entry in the node's ledger includes the fingerprint of the previous entry. Entry five carries entry four's fingerprint; entry four carries entry three's; all

the way back to the first. They're a chain, each link holding the one behind it.

Watch what that does to a cheat. Suppose the node's owner wants to quietly change entry

two — turn an £8 fare into £80 after the fact. They edit entry two.

But entry two's

fingerprint is computed from its contents, so changing the contents changes its

fingerprint. And entry three was carrying the old fingerprint of entry two. The link no

longer matches. The chain is visibly broken, exactly at the tampered entry, and anyone who

recomputes the fingerprints — a check that takes milliseconds — sees precisely where the

lie is. To hide it, the cheat would have to re-fingerprint entry two, and then entry three

to match, and then four, and five, all the way to the end. In a moment we'll make even

that impossible; but already, the past has grown teeth.

RUN IT

Open the prototype and run:

```
python demo_two_nodes.py
```

You'll see two nodes — call them Manchester and Bristol — each host a handful of transactions: a taxi fare, a tip to a creator, a marketplace sale. Each transaction gets appended to that node's chain. The program prints a Merkle root for each node — a single fingerprint that summarises the entire chain at once, the way one wax seal can stand for a whole stack of pages. If any entry anywhere in the chain changes, the Merkle root changes. It's the node's whole history compressed to one honest number.

Then the demo does the thing worth watching. It takes Manchester's records and tampers with one — inflating a fare from £8.40 to £84.00, exactly the quiet edit a dishonest operator would want. It re-runs the check. And the output is not "looks fine." It's:

content tampered (entry hash mismatch) at entry 0

The lie is caught, and located, automatically. Not because a company was watching — no company is in this picture at all — but because the record is built so that lying to it

leaves a mark. Change the number in the demo yourself. Make it £8.41. The check still catches it. Make it back exactly £8.40 and it passes again. The past will only accept the truth.

WHY THIS IS ALREADY DIFFERENT

Stop and notice what we have, because it's more than it looks.

A node is a small computer someone local owns. It records the neighbourhood's transactions. And its record cannot be quietly rewritten — not by an outsider, and *not by the person who runs the node*. That last part matters more than anything. The whole reason we tolerate a distant corporation holding the ledger is a fear that whoever holds it might cheat. The hash chain shrinks that fear: the holder of the ledger can add to it but cannot secretly rewrite it. You've moved the record from a company's database you can't see to a box on a shelf you can — and made the box unable to lie about the past.

But — and this is the honest gap that drives the next chapter — you

can only see the lie
if you have the original chain to compare against. And a determined
cheat, remember, could
re-fingerprint the whole chain from the tampered entry forward,
producing a smooth,
unbroken fake that its own recompute would accept. Right now the
node's honesty still
rests on the node's goodwill, or on someone keeping a copy to check
against. That's not
good enough. A witness you have to take on trust is just a landlord
with a friendlier face.

We need the node to be able to prove — to a stranger, who holds no
prior copy and trusts
the operator not at all — that its record is genuine and unaltered. We
need it to be unable
to produce a convincing fake even if it wants to. That's not a
software problem anymore.
For that, we have to reach for a piece of hardware — a chip that costs
about five pounds,
and that turns "please trust my box" into "here is proof you can
check yourself."

Turn the page. We're going to give the node a conscience it can't
override.

CHAPTER 6

THE FIVE-POUND CONSCIENCE

We left the node with a problem. Its chain catches a clumsy tamper, but a determined operator could re-fingerprint the whole thing and hand you a smooth fake. As long as the node could lie convincingly, "trust my box" is still trust in a person — and this whole book is about not having to do that. We need the node to be unable to produce a believable fake, and to be able to prove that to a stranger who holds no copy and trusts no one. That is the job of the chip.

WHAT A TPM IS

Inside a lot of modern hardware — and available as a component the

size of a fingernail
that costs about five pounds — is a TPM: a Trusted Platform
Module. The one this
book's prototype is written around is a real, off-the-shelf part, the
Infineon SLB9670.
You can buy it as a tiny board that plugs onto a Raspberry Pi's pins.
No custom silicon,
no fabrication run — a five-pound chip and a solder-free connector.

A TPM does one thing that matters here: it holds a secret key that
never leaves the
chip. The key is generated inside the module, lives inside it, and the
chip will use it
to sign things on request — but it will never hand the key out, not to
the operating
system, not to the person who owns the board, not to anyone. You
can ask the chip "sign
this," and it will give you back a signature; you cannot ask it "give
me your key," ever.

A signature is the other half of the fingerprint idea. Sign a piece of
data with a
private key, and anyone holding the matching public key can check
that the signature is
genuine and the data wasn't changed — but nobody can produce that
signature without the

private key. So a signature says two things at once: this exact data, and *signed by the holder of this specific key*. It's unforgeable by anyone who doesn't have the key — and with a TPM, nobody has the key, not even the chip's owner. They can only make the chip use it.

SIGNING THE RECORD

Now wire that into the ledger. Every time the node appends a transaction, it asks its TPM to sign that entry's fingerprint. The signature goes into the record alongside the entry.

Follow what this does to our determined cheat. Before, they could re-fingerprint the whole chain and produce a smooth fake. Now, each entry must also carry a valid signature from this node's TPM, over that entry's fingerprint. To fake a chain, the cheat needs to produce new signatures for the altered entries — and they can't, because the key that makes those signatures is locked inside a chip that will only sign what it's given, and will never reveal itself. The operator owns the board, owns the

software, owns the whole
node — and still cannot forge its own history, because the one part
that signs is a chip
that won't be told what to do about the past. The conscience is in
hardware, and even the
owner can't override it.

BUT WHO SAYS THE CHIP IS REAL?

There's a gap left, and it's a subtle, important one. A signature proves
"signed by the
holder of this key." But how does a stranger know that key belongs
to a genuine Vault
node, and not to some impostor who generated their own key and is
signing their own
made-up transactions? A liar with a chip is still a liar.

This is the oldest problem in trust, and it has a clean answer:
certification. When a
node is set up, a trusted party — call it the manufacturer, the foundry
that makes the
nodes — signs a little document that says "the key ending in these
digits belongs to node
Manchester-01, and I vouch for it." That document is a certificate.
Now a stranger who
trusts the manufacturer's single public key can check any node's

certificate, and through it, trust that node's signatures — without ever having met the node before. One root of trust, checked once, and every node underneath it becomes verifiable.

Note what the manufacturer is and isn't. It vouches that a node is a genuine node. It does not see the transactions, hold the records, take a cut, or run anything. It's a notary who certified an identity once and then went home — not a landlord. And on the brand's own terms it's a small thing: the certificates in this prototype are issued using an RSA signature scheme taken straight from an expired patent in the Vault code library — public-domain cryptography, no license, no dependency on anyone.

WATCH IT HOLD, AND WATCH A FORGER FAIL

Run `demo_two_nodes.py` again and read the rest of its output with new eyes. Bristol receives Manchester's chain — arriving as plain data over a wire, the way it would cross a real network — and checks four things in order: is Manchester's certificate signed by the

manufacturer (a genuine node)? does the certificate match the key that signed the entries?

is every entry's signature valid and the chain unbroken? does the summary fingerprint

match? All four pass, and Bristol issues a signed receipt — its own TPM vouching, "I

checked Manchester's record at this point, and it's genuine." A receipt anyone can verify with Bristol's public key alone.

Then the demo sends in a forger: a node whose certificate was not issued by the real

manufacturer — an impostor from a rival foundry. It presents a perfectly well-formed chain, signed with its own key. And Bristol's verdict:

EK cert not signed by manufacturer (impostor node)

Refused at the root. The forger can sign all it likes; it can't sign its way into being a

genuine node, because it doesn't hold a certificate from the one authority everyone

checks against. Tamper detection catches a changed record; certification catches a fake

identity. Between them, the node can now prove its honesty to a stranger — and a liar

can prove nothing.

WHAT WE BOUGHT FOR FIVE POUNDS

Step back and see the size of what just happened. We wanted to remove the corporation from the middle of a transaction. The corporation's one genuine job was to be trusted — to hold the record and vouch that it was real. We've now done that with a five-pound chip and a notary who leaves after saying hello. The node keeps the record; the chip makes the record unforgeable even by its owner; the certificate lets a stranger trust it cold. There is no company in this picture, and yet the record is more trustworthy than a company's, because you can check it yourself instead of being told to have faith.

And there's a quiet bonus that turns out to matter enormously for the whole economic argument later. Because each hosted transaction carries an unforgeable signature from the node's chip, you can count them honestly. A node can't inflate how much work it did — every claimed transaction must carry a real signature a stranger will check. "Paid per

transaction you actually hosted" stops being a promise and becomes arithmetic. Hold onto that; it's the seed of how a node earns its keep without a platform skimming the top.

One node can now prove itself to one stranger. But a single box on a shelf can still break, or vanish, or be seized. To make the memory of a neighbourhood truly robust — and truly ownerless — one honest node isn't enough. We need many, checking each other, so that no single node has to be trusted or even has to survive. That's the neighbourhood, and it's the heart of this book.

PART THREE — THE NEIGHBOURHOOD

. . . .

CHAPTER 7

NOBODY IN CHARGE

Here is the centre of the book. Everything so far has been about one node earning trust.

But one node is a single point of failure and a single point of power — it can break, be seized, or, however well we've chained and signed it, simply go dark and take the neighbourhood's memory with it. A village that depends on one box on one shelf hasn't escaped the landlord; it's just become its own small landlord. The real prize is a memory that no single node has to be trusted for, and no single node can destroy. For that, the nodes have to check each other. And crucially, they have to do it with **nobody in charge* of the checking* — because the moment there's a central referee,

you've reinvented the
company in the middle.

GOSSIP

The mechanism is old and humble, and it's literally called gossip.

Each node

periodically tells the others "here is my chain, at this point,
summarised by this

fingerprint." Any node that hears it runs the four-step check from the
last chapter — is

this a genuine node, does its certificate match, is every entry signed
and the chain

unbroken, does the summary hold — and if it all passes, the listener
signs a receipt:

its own chip vouching, "I checked node Manchester at this point, and
it's real."

Those receipts then travel too. They're built to stand on their own —
each receipt carries

the certificate of the node that issued it, so anyone can check a
receipt they were merely

handed, without having witnessed the original chain themselves.

Receipts flood outward

through the neighbourhood the way a rumour does, and every node
collects them. Nobody

gathers them centrally. Each node just listens to the chatter and keeps its own tally of who has vouched for whom.

QUORUM: TRUST AS A COUNT

Now the beautiful part. Once receipts are flowing, a node's chain isn't trusted because one neighbour said so — it's trusted because k of them independently did. Set k to three, and a record only counts as solid once three separate nodes, three separate chips, have each checked it and signed off. This is called reaching quorum, and it changes the nature of trust completely.

Think about what a forger now faces. To make the neighbourhood believe a lie, it's not enough to fool one node, or to run one dishonest node yourself. You'd need three genuine, manufacturer-certified chips to each independently verify and sign a record that is, in fact, false — and they won't, because each one runs the same check and the check catches the lie. Trust has become a count of independent witnesses, exactly like the village

that knew who owed whom because several neighbours saw the deal.
No head office. No
referee. Just enough honest chips agreeing, and a threshold you can
set as high as the
stakes deserve.

RUN IT

```
python demo_gossip.py
```

Five nodes come up, host some transactions, and gossip. Watch the
output: each node,
independently, reports which of its neighbours it has seen reach
quorum — and they all
converge on the same answer, having compared notes with no central
tally. Then the demo
sends in trouble. A node advertises a tampered chain: zero receipts
— every neighbour's
check rejects it, so it never reaches quorum and never counts. An
impostor from a rogue
foundry joins and advertises: zero receipts — its certificate fails at
the root. Neither
can be believed, and neither can be billed, because billing runs on
quorum-verified
transactions and a lie collects no signatures.

There's a detail in that demo worth pausing on, because it's the sort of honesty a real system needs. When a node hosts a new transaction, its summary fingerprint changes — and for a moment the neighbourhood still only vouches for the old summary. The new transaction isn't yet co-signed; it hasn't been witnessed. The demo shows the certified count sitting at the old height until the next round of gossip lifts it. You can never bill a transaction the neighbourhood hasn't actually seen and agreed on. Trust lags reality by exactly one honest conversation, and that's the correct behaviour, not a bug.

WHEN THE INTERNET DIES

So far the nodes have been gossiping over an ordinary local network — a cheap switch in a cupboard, which is fine for a cluster in one building. But the whole point was to not depend on infrastructure that can fail or be owned. So the prototype carries the same gossip over two more channels, and this is where it stops being a computer-science demo and starts being a thing a real town could lean on.

The first is a private encrypted mesh (the Vault code calls it B8M): nodes talk to each other in end-to-end-encrypted messages through a relay that is deliberately blind — it forwards sealed envelopes and can't read a thing inside them. Run `demo_gossip_b8m.py` and the exact same five nodes reach the exact same quorum, except now every chain and every receipt crosses as ciphertext a relay operator could never snoop. The middle, even where a relay exists, is made deaf.

The second is the one that matters when things go wrong: radio. LoRa is a cheap, long-range, license-free radio — the kind that carries a trickle of data across a town for milliwatts, no cell tower, no contract. Run `demo_gossip_lora.py` and watch the same gossip fragmented into tiny radio packets and reassembled on the far side. A transaction record is far bigger than a single LoRa packet, so it's chopped up, sent as a burst, and rebuilt — and because radio drops packets, the nodes notice the gaps and ask for just the missing pieces back. The demo turns the loss up to 30% and shows the difference starkly:

without that recovery, the neighbourhood's agreement collapses to nothing; with it, every node claws its way back to full quorum, and — this is the point — the forger and the tamper still collect zero receipts through all of it. Robustness never once loosens a security check.

Sit with what that means. A market town's nodes could keep hosting transactions, checking each other, and reaching agreement over cheap radio with no internet at all — through an outage, a disaster, or simply in a place the networks never bothered to reach. The honest memory of a community becomes something that runs on milliwatts and air, owned by the community, answerable to no one, and standing when the infrastructure everyone assumed was load-bearing has fallen over.

THE SHAPE OF IT

Look back at what the neighbourhood has become. No node is in charge. No node has to be trusted — trust is a count of independent chips, set as high as you like. No node is

load-bearing — lose one, or several, and the memory survives in the others. There's no central tally, because each node keeps its own and they agree by gossip. And the whole thing keeps working when the internet doesn't, over radio a hobbyist can buy for the price of a takeaway.

That is an honest, ownerless, resilient memory for a community — the witness put back in the village, made robust enough to lean a real economy against. But before we build on it, there's one door left with someone standing in it. Every node in this chapter was let in by a certificate from one signer — and whoever signs the identities decides who exists. "Nobody in charge" isn't true until nobody is in charge of the membership either: until a stranger can join on the neighbourhood's say-so, and a cheat can be thrown out with no judge. That's the next chapter, and it's the one that finishes the argument.

CHAPTER 8

THE FRONT DOOR

There is a crack in everything we've built so far, and before we put a marketplace on top of it I want to show it to you, because fixing it is the best chapter in the whole story.

Think back to the chip. A stranger trusts a node because its certificate was signed by the manufacturer — one trusted party who vouched for each node's identity and went home.

For a handful of Raspberry Pis you set up yourself, that's fine: you were the manufacturer, you were only vouching to yourself. But now imagine the mesh growing. A shop two streets over wants to run a node. A stranger in another town wants in. Who signs their

certificate?

If the answer is "the same one party, forever," then look at what we've actually built:

a network where one person decides who exists. They don't hold the records, they don't

take a cut — but they hold the door, and holding the door is the oldest power there is.

We'd have thrown the landlord out of the living room and left him standing in the porch.

The big "decentralised" networks mostly never fixed this. The famous ones are validated

by a couple of dozen computers effectively chosen by a founder; membership is bought with

wealth, or granted by whoever got there first. The word on the tin says *no one is in

charge*; the door says otherwise.

So this chapter removes the doorman. By the end of it, a total stranger — someone we've

never met, vouched for by nobody we appointed — will walk up to the mesh, knock, and be

let in by the neighbourhood itself. And a cheat, including one of the founders, will be

thrown out by pure arithmetic, with no vote taken.

A NAME NOBODY CAN STEAL

First, a small repair with big consequences. Until now a node's name was just a label —

"node-manchester-01" — and nothing stopped a liar claiming someone else's. So the mesh stops using names. A node's real identity is now the fingerprint of its own key: a

short string computed from the key, the way a fingerprint is computed from a finger.

You can't claim my identity without my key, and the key never leaves my chip. Names like

"manchester-01" are still there for humans — but they're decoration, like the name on a doorbell. The lock only knows the fingerprint.

Run `demo_two_nodes.py` again and you'll see a new trick in it: a genuine node

re-broadcasts another node's chain under a false identity, and it's refused instantly —

"node_id not derived from the presented key." Identity theft isn't detected in this system; it's impossible to express.

A CHIP FROM ANY SHOP

Next, the certificates. The mesh now recognises two kinds of root.

The founder

root is the original one — it identifies the handful of nodes that started the mesh, and

they're grandfathered in, because somebody has to be first. But alongside it the mesh

accepts vendor roots: the certificate of the actual chip maker. A real TPM arrives from

the factory with the maker's certificate already burned in — that's true of the real

five-pound part, and the prototype mirrors it.

Understand exactly what a vendor certificate says, because the whole design turns on it.

It says: this is a genuine chip. It does not say: *this chip's owner is welcome

here*. The factory vouches for the silicon, never for the neighbour.

Anyone can buy a

chip; nobody can buy membership. Membership has to be earned at the door — and the door

is what we build next.

KNOCKING

Here is how a stranger joins, and notice as we go that no single party can say yes.

The stranger broadcasts a hello: its fingerprint-identity, its public key, and its chip's factory certificate. Any member who's willing to consider newcomers answers with a challenge — a large random number, made up fresh, used once. This is an old and lovely trick: because the number has never existed before, whatever the stranger does with it can't be a recording of something done earlier. A thief who taped yesterday's conversation has taped nothing of value.

The stranger answers each challenge with a proof: it has its chip sign the challenge together with a promise of what the first page of its ledger will say. That signature proves three things at once — I hold the key I claim (only my chip could sign), I'm answering you, now (your fresh number is inside), and I'm binding myself to *this one chain* (the first page is inside too, so I can't get admitted and then show up with a different history).

Each member now checks, alone, on its own box, with nobody's permission: is the factory

certificate genuine? Does the fingerprint match the key? Does the signature hold? Is this stranger on the banned list? Have we admitted too many newcomers this epoch? If it's all clean, the member signs a receipt — the same kind of self-contained, anyone-can-verify receipt the gossip already uses — saying I vouch for letting this one in.

When the stranger has collected receipts from k different members — three, five, as high as the neighbourhood likes — it bundles them into an admission certificate and writes it into the very first entry of its own ledger. That's worth sitting with.

Membership isn't a row in someone's database. It's written into the member's own chain, carried everywhere the chain goes, checkable by anyone against the roots alone. From that moment, every node's gossip simply ignores any newcomer chain whose first page doesn't carry a valid admission — the door enforces itself.

Count the people in charge of that decision: none. Three independent chips said yes, each applying the same checks you can read. Run python

demo_join.py and watch it happen — and watch a rogue chip collect zero receipts, a taped replay bounce off the fresh numbers, and a name-thief get caught by the fingerprint.

THE WAITING ROOM

Being let in is not the same as being believed. That distinction is the mesh's defence against its most obvious attack, so let's face the attack squarely: chips are five pounds.

What stops someone buying fifty, admitting them, and out-voting the honest neighbourhood?

Two things. First, the admission cap: members will only co-sign a few admissions per epoch (an epoch is just a fixed slice of time — a day, say). Fifty chips take weeks to get through the door, in full view, at a pace the neighbourhood watches.

Second, and better: probation. A newly admitted node can trade immediately — it can host transactions, be checked, be vouched for. What it cannot do is vouch. Its receipts about other nodes count for nothing until its own chain has stood at quorum through the

most recent stretch of epochs — until it has simply *been there, visibly honest, for a while*. And the window slides: a node that sleeps through its probation serves it when it wakes, and a full member that goes dark slips back into the waiting room until it has shown up again. Membership is maintained, not vested — exactly like standing in a real village.

Run `python demo_probation.py` and you can watch the whole biography: admitted on day one, trusted as a subject immediately, worthless as a witness until it has served two clean epochs — and a receipt it co-signs during probation quietly not counting towards someone else's admission. The swarm of fifty chips doesn't crash into the mesh; it queues at the door and then sits in the waiting room, watched, gaining influence at the speed of demonstrated honesty. Which is the only speed trust should ever move at.

THROWN OUT BY ARITHMETIC

A door that only opens isn't a door. Membership must also be losable

— and here the
mesh does something I find genuinely beautiful: for the worst
offence, it needs no vote,
no trial, and no judge.

The worst offence is double-signing: a node keeping two versions of
its own history —
telling you one story and me another, each version signed by its own
chip. Remember, the
chain's whole promise is one past. A node that signs two different
entries for the same
position has broken the promise in a way that leaves evidence: two
signatures, from one
chip, that cannot both be true. Put those two signed entries side by
side and you have a
fraud proof — a small, portable object that anyone can check against
the public
roots, and that convicts absolutely. Every honest node that sees it
reaches the same
verdict independently, drops the cheat from its view of the mesh, and
passes the proof
along. No meeting was held. The cheat's own signatures did all the
talking.

And notice two properties that matter more than the punishment.
Nobody can be framed —

the only signatures that can convict a node are its own, so planting evidence on an honest neighbour is impossible; the demo tries, and fails. And founders aren't special — the demo's double-signer is one of the original nodes, and it goes down exactly like anyone else. Whatever privilege being first carried, it does not include immunity.

For misbehaviour that can't be proven — a node that's simply always unreachable, or floods the mesh with noise — eviction takes the expensive road: two-thirds of the active members must co-sign the expulsion. That asymmetry is deliberate. Where the math can speak, punishment is instant and free. Where it's only opinion, it should cost a supermajority — because "we don't like them" must never be cheap.

The list of the evicted travels the mesh like everything else, so a node that was switched off during the scandal catches up on who was expelled while it slept — and re-checks every piece of evidence itself before believing a word of it. Run `python demo_fraud.py` and watch all of it: the conviction, the failed

framing, the
founder falling, the two-thirds vote, the latecomer catching up.

BOOTING INTO THE MESH

Finally, the part that makes this real rather than clever: the whole ceremony is folded into the ordinary life of an ordinary box. A Raspberry Pi that has never met the founders needs exactly two things on its card: its chip's factory identity, and a small public file naming the roots the mesh accepts. Switch it on and it knocks, answers its challenges, earns its receipts, writes its admission into page one of its ledger, and starts gossiping. No controller. No enrolment call. Nobody's permission. The command is one line, and `demo_boot_join.py` runs the entire arc — including the reboot.

The reboot hides my favourite detail in the chapter. A node's ledger now persists on disk, and it must — not for convenience, but because of the fraud proofs we just built.

A node that forgot its chain and innocently started a fresh one would re-sign its first

entry a second time — two signatures, one position — and hand the mesh a perfect fraud proof against itself. The system we built to catch liars is so strict that honest amnesia is indistinguishable from lying. So nodes remember. The discipline isn't in a rulebook; it's structural.

And the same front door works over the radio. Every message in the join ceremony — the hello, the challenges, the proofs, the receipts, the fraud proofs — fits through the same tiny 184-byte packets as the gossip, survives the same 30% packet loss, and recovers with the same repair. `demo_join_lora.py` admits a stranger, refuses an impostor, and convicts a double-signer entirely over simulated air. The governance of the mesh needs no better network than the mesh itself.

THE DOOR WITH NO DOORMAN

Step back and look at the whole thing now, because chapter 4's claim — **nobody in charge** — is finally true all the way down. Nobody is in charge of the records: the chain

and the chip. Nobody is in charge of the checking: gossip and quorum. And now nobody is in charge of the membership: strangers admitted by any k members, newcomers proven before they're believed, cheats convicted by their own signatures, and the founders' only remaining privilege the fact of having been first. The one power that every "decentralised" platform quietly kept — the door — has been handed to the neighbourhood.

Now we can build the village on top of it.

PART FOUR — WHAT RUNS ON TOP

. . . .

CHAPTER 9

THE TWO PERCENT PLATFORM

We have an honest, ownerless memory. Now we build a life on it — the actual things people do: hail a ride, order food, sell a chair, tip a musician, install an app. This is where the argument stops being about cryptography and becomes about economics, because the whole reason to do any of this is that removing the landlord changes the arithmetic of a community, and it changes it more than you'd guess.

THE CUT, REVISITED

Go back to the four bars from Chapter 1. Three of them — the platform, the record, the off switch — we've dismantled: the node is the platform, the chain is the

record, and no one holds the switch. That leaves the cut. And the cut is where the whole thing pays for itself, or doesn't.

A modern platform takes something like 20 to 30 percent of a transaction, and it has to, because it's feeding an enormous apparatus: data centres, engineers, marketing, investors expecting a return, a head office, a legal department, shareholders. That machine is the middle, and the cut is its rent. The transaction between the driver and the rider is, in a sense, subsidising a corporation neither of them will ever see.

Now count the cost of a Vault transaction. A node hosted it — a Raspberry Pi on a shelf, drawing a few watts, run by a local person. The record was checked by a handful of other such nodes. That's the entire apparatus. There's no data centre, because the neighbourhood is the infrastructure. There's no head office to feed. The only real cost is the node operator's electricity and their time keeping the box alive — and, recall from Chapter 3, we can count exactly how many transactions each node honestly

hosted, because each one carries an unforgeable signature. So the operator can be paid *per transaction they actually hosted*, a tiny amount, provably earned — and the rest of the value stays with the two local people who created it.

That's the shape of the Vault economics the prototype's app store is built around: a platform cut of 2 percent instead of 30, and a marketplace fee of around 5 percent that stays in the local system rather than leaving town. Not because anyone is being generous — because there's no longer a landlord in the middle who has to be fed. The difference between 2 and 30 isn't a discount. It's the measure of how much a distant corporation was costing a community to stand where it didn't need to stand.

A STORE WITH NO GATEKEEPER

On top of the node runs a small operating system — Vault OS proper, the part a person actually touches — and on top of that runs a store of apps. Here the design mirrors the

super-apps you already know (the ones where a chat app also hails taxis and pays bills),
but with the gatekeeper removed.

A Vault app is a self-contained little program that runs in a sandbox on the phone and talks to the system through a narrow, safe bridge — it can ask "who is this user?" and "take this payment," but it can't reach past its box and rummage through your life. A developer writes one, publishes it — and here's the turn — the app's files are *hosted on the nodes*, the same neighbourhood boxes, not on the store owner's servers. When someone installs it, they're pulling it from a node. When they pay inside it, the payment splits automatically: the developer keeps 98 percent, and 2 percent goes to run the platform. No approval committee, no 30 percent toll, no company that can pull your app because it competes with theirs.

And because the developer brings their own AI key to build the app — their own inference, their own cost — the 2 percent doesn't have to cover a mountain of compute the platform is

subsidising. It's close to pure coordination overhead. The economics only close because there's no landlord: every layer that would normally take a cut has been either removed or handed back to the person doing the work.

A PHONE THAT ISN'T WATCHING YOU

The consumer face of all this — the phone shell in the prototype — is worth describing because of what's absent from it. There are no ads. Your identity isn't a phone number or an email rented from a telecom and a mail provider; it's a key you hold, the same kind of key the nodes use, so "logging in" is proving you hold it rather than asking a company's permission. Your reputation — the record that makes you trustworthy to the next driver or the next buyer — lives in the ledger you can inspect, not in a database you'll never see.

The comms run over the encrypted mesh from Chapter 4. The number of ads you've been shown, in the prototype's own profile screen, is a proud zero.

I want to be careful and honest here, because this book earns its keep by drawing the line

between what runs and what's a sketch. The ledger — the node, the chip, the gossip, the quorum, the mesh and the radio — is real, and you've run it. The consumer ecosystem on top — the store, the phone shell, the taxi and marketplace apps — exists as working prototypes I built and can demonstrate, but it is not a shipped, populated platform with a town's worth of real drivers and shops on it. It's the cabin built on the foundation, not the lived-in village. The foundation is the hard part and the honest part, and it's solid. The village is the work ahead, and the next chapter is honest about exactly how much of it is standing.

WHY THE ORDER MATTERS

Notice that we built this in a very particular order, and the order is the whole argument.

Most platforms start with the app — the shiny thing — and bolt trust on afterward as terms of service, an account you don't control, a promise to behave. We did the reverse. We built the trust first, in hardware and mathematics, so that it belongs to no one and can't be revoked — and only then put the app on top. Which means the app

inherits the property of
its foundation: it runs between local people, on hardware they own,
with no one in the
middle able to take the cut, hold the record, or throw the switch.

That's the two percent platform. Not a cheaper version of the thing
we have — a
different thing, where the middle was dismantled instead of merely
discounted, and the
value that used to leave town stays in it. What a community could do
with the eighteen
percent it stops exporting is the beginning of a much longer story.
The last two chapters
are about where that story goes, and about telling you, without a
salesman's gloss, exactly
what you could pick up and build tomorrow.

CHAPTER 10

WHERE IT GOES

The prototype is small on purpose — a handful of nodes on a desk, a market town's worth of imagination. But the idea underneath it, "put the witness back in the village and let the value stay local," points at a horizon much wider than a taxi app. This chapter walks that horizon. Read it as a set of directions the same tiny core points in — not as things I'm claiming to have shipped. The next chapter draws the honest line between the two; this one is allowed to dream, so long as it dreams from the working core outward.

A STREET, THEN A TOWN

Start small and real. Five nodes in one building is a cluster; the prototype provisions exactly that. But the same design scales by addition, not redesign. A café runs a node.

The corner shop runs one. The community centre, the barber, the taxi firm — each a node, each hosting its own transactions, all gossiping and co-signing each other's records over the mesh. The town's economic memory becomes a fabric woven from boxes on shelves, no one of which is in charge, none of which can be switched off from outside. Add a node and the fabric gets stronger; lose one and it heals around the gap. There's no capacity to purchase, no plan to upgrade — the neighbourhood's ability to host itself grows one shelf at a time.

THE PLACES THE NETWORKS SKIPPED

The radio chapter wasn't a party trick. There are enormous parts of the world where the platforms never came, or came only to extract — where a cell contract costs a day's wages and the payment rails are somebody else's and take their cut in hard currency. A node that

runs on a few watts, proves its own honesty with a five-pound chip, and gossips over license-free radio doesn't need any of that to have arrived first. It can stand up a local market, a local ledger, a local identity that isn't rented from a foreign company, in a place with intermittent power and no data centre for a thousand miles. The Vault work has a strand aimed squarely at this — identity you can hold without documents, value that moves over the mesh, an economy that doesn't have to phone another continent to function. Of everything on this horizon, this is the part I care about most, because it's the difference between technology that reaches a place and technology that belongs to it.

WHEN THE LIGHTS GO OUT

The same properties that serve a place the networks skipped serve any place when the networks fail. A storm takes out the cell towers; the internet goes dark for a week. The platforms — the ride app, the payment app, the marketplace — all go silent, because their brains are in a data centre that can't be reached. But a neighbourhood

of nodes gossiping
over radio doesn't know the difference. Transactions keep being
hosted, checked, and
agreed; the shop can still sell, the record still holds, and when the
wider world comes
back the local ledger reconciles with it, intact. Resilience stops being
a feature you hope
your provider has and becomes a property of owning the middle
yourself. The town that
hosts itself is the town still standing when the infrastructure everyone
leaned on has
fallen down.

A RECORD THAT REACHES BACK DOWN THE SUPPLY CHAIN

There's a subtler direction, and it follows from the ledger being
honest rather than
merely cheap. If every transaction can carry a proven,
tamper-evident record, then a
product can carry its history: where it was made, by whom, on what
terms, at each hop from
origin to shelf. A neighbourhood ledger that can't be quietly
rewritten is exactly the tool
for making an extractive supply chain visible — for letting a buyer
see, and a maker prove,

what actually happened between the field and the counter. The same honesty that stops a node inflating a fare can stop a chain hiding where the value was taken. It's a long road and I won't pretend it's paved, but the honest ledger points down it.

THE NODE GETS ITS OWN HARDWARE

Everything in this book runs on general parts — full Raspberry Pis, breakout chips, jumper wires — and that's the right way to prove an idea: loose, cheap, solder-free, so anyone can rebuild it. But once the design is proven, the natural next step is to integrate it: a purpose-built node board with the compute, the TPM, the mesh radio, and the power all on one small, rugged, low-power unit you could screw to a wall and forget. Not because the loose version doesn't work — it does — but because a village shouldn't need a hobbyist to keep its ledger alive. The point of dedicated hardware is the same as the point of the whole book: push the ability to host yourself down to something so cheap, so robust, and so free of middlemen that you can drop it anywhere, including everywhere a platform decided wasn't

worth its time.

THE THROUGH-LINE

Every one of these — the street, the town, the skipped places, the blackout, the supply chain, the dedicated board — is the same idea at a different size: a node someone owns, hosting an honest record, checked by neighbours, answerable to no one in the middle. Nothing on this horizon needs a new invention. It needs the working core you've already run, carried further by people who want to carry it. Which is exactly why the last chapter is the most important one in the book: it tells you, with no gloss, precisely where the solid ground ends and the horizon begins — so you know exactly what you'd be picking up if you decided to build the next mile yourself.

CHAPTER 11

WHAT'S BUILT, WHAT ISN'T, AND WHAT YOU COULD BUILD

I promised a builder's map, which means an honest one. The last chapter painted a wide horizon — streets, towns, the places the networks skipped, a world of nodes hosting themselves. This chapter draws the line between what you can run today and what is still a drawing, because that line is the most useful thing I can give you. A vision that hides its own edges is just a sales pitch, and the value of this book is in showing you the ground you're standing on as clearly as the horizon you're looking at.

THE HONEST LEDGER

Real, and running in your hands right now — the vault-ledger

prototype:

- The node and its chain. Transactions hosted, hash-linked, and summarised by a Merkle

- root; a tamper caught and located automatically.

demo_two_nodes.py, and you watched

- it flag an inflated fare at the exact entry.

- The chip. Each entry signed by a TPM whose key never leaves it, so even the node's owner

- can't forge its history; a manufacturer certificate so a stranger can trust a node cold;

- and a forger from a rogue foundry refused at the root. The signatures use an RSA scheme

- from an expired patent — public-domain, no dependency on anyone.

- The neighbourhood. Many nodes gossiping, checking each other, and reaching quorum —

- trust as a count of independent chips, with no central tally.

demo_gossip.py, including

- the tamper and the impostor collecting zero receipts.

- The same gossip over a blind encrypted mesh

(demo_gossip_b8m.py) and over cheap

- license-free radio (demo_gossip_lora.py), fragmented into tiny packets, reassembled,

- and — with packets recovered after loss — surviving a 30% failure rate all the way back to

- full quorum, with every security check still holding.
- The cluster set-up. A controller that acts as the manufacturer, provisioning each node's identity over the network with a proof it really holds its own key; and a one-command installer that stands a node up on a Raspberry Pi, TPM overlay and all, into a service that boots straight into the gossip.
 - The front door. Identities that are fingerprints of keys, so a name can't be stolen; strangers with factory-certified chips admitted by any k members, their admission written into the first page of their own chain; probation, so a newcomer is checked from day one but believed only after serving visible clean time; and the admission rate cap. All of Chapter 5, in `demo_join.py` and `demo_probation.py`.
 - Eviction. Fraud proofs — a double-signer convicted by its own two signatures, with no vote, no possibility of framing an innocent node, and no immunity for founders — plus the two-thirds supermajority for offences arithmetic can't prove, and a revocation list a returning node re-verifies rather than believes. `demo_fraud.py`.
 - The stranger's boot. The whole join ceremony folded into the node agent: a Pi with

nothing but its chip identity and a small file of accepted roots earns admission from

the mesh over the live encrypted relay, persists its ledger and its admission

certificate, and survives a reboot without double-signing itself into a fraud proof.

demo_boot_join.py — and the same ceremony over lossy radio in demo_join_lora.py.

That is not a mock-up. It is a complete, if small, working demonstration that a community can keep an honest, ownerless, resilient record of its own transactions — with no company in the middle to trust, to pay, or to be switched off by. Everything the first four chapters argued, you can rerun and take apart.

Working prototypes, but not a shipped platform — the consumer ecosystem from Chapter 6.

The app store, the phone shell, the 2%-split payments, the sandboxed apps, the developer studio built from expired patents: these I built and can demonstrate, but they are cabins on the foundation, not a lived-in village. There is no town's worth of real drivers, shops, and customers on them. They show the shape of what runs on top of the

ledger; they are not a
populated product, and I won't dress them up as one.

Drawings, not products — everything in Chapter 7. The town-wide
fabric, the places the
networks skipped, the blackout resilience, the supply-chain history,
the dedicated node
board: these are directions the same idea points in, honest
extrapolations from a working
core, labelled as such precisely so you can tell the ground from the
horizon.

Between the solid foundation and the horizon is the interesting work,
and it's work I'd love
other people to do.

THE ROAD FROM DESK TO WALL

The demos run the mesh over local wires and simulate the radio's
noise as a number. Two real
steps bridge from here to a node on a wall.

The first is the radio itself. The gossip already fragments,
reassembles, and recovers lost
packets exactly as a real long-range radio requires — that code was
written to mirror a real

companion radio, so putting it on actual LoRa hardware is a bridge, not a rebuild. When you cross it, the "loss" stops being a parameter and becomes a real wall, a real distance, a real rainy afternoon, and the same recovery that survived simulated loss has to survive the genuine article.

The second is the chip. The prototype's TPM is emulated in software with a clean boundary around it, written so that swapping in a real Infineon SLB9670 — the five-pound part, plugged onto a Pi's pins — is a change inside that boundary and nothing else. The node key stops living in a file and starts living in silicon that will never hand it over. That is the single most valuable experiment left, and it needs no permission and no cloud: a Pi, a five-pound chip, and the code you already have.

From there it's well-trodden hardware engineering, not new science. Everything a node does — the chain, the signing, the gossip, the radio — fits comfortably on parts you can buy today, and eventually onto one small purpose-built board. The point of integrating it is the same as the

point of the whole book: push the ability to host yourself down to something so cheap and so free of middlemen that you can drop it into any street, anywhere.

BUILD YOUR OWN

The prototype is meant to be forked. A few concrete places to start, in rising order of adventure:

1. Break the ledger and watch it notice. Open `demo_two_nodes.py`, change a fare after

it's signed, and see the tamper caught at the exact entry. Then try to fake it properly —

and feel where the chip stops you.

2. Raise the bar on trust. In `demo_gossip.py`, change the quorum threshold. Watch how

many independent chips a lie would have to corrupt, and see that it can't corrupt even one.

3. Turn up the storm. In the radio demo, push the packet loss higher and higher. Find the

point where recovery can't keep up — and understand exactly what a real town's radio budget

would have to beat.

4. Try to get in without knocking. In `demo_join.py`, give a node a genuine vendor chip

and no admission certificate, and watch the mesh ignore it entirely.
Then try to frame an
innocent node in `demo_fraud.py` — move the evidence onto
someone else — and see why only
your own signatures can convict you.

5. Stand up a real cluster. Run the installer on a few Raspberry Pis,
provision them from

a controller, and watch physical boxes on a shelf reach quorum
over a real network. Then

give a fifth Pi nothing but a vendor identity and an anchors file, run
the agent's join
mode, and watch the mesh let a stranger in with nobody's
permission. That's Vault OS off

the screen and on the wall — genuinely within reach of a
determined hobbyist.

6. Add the five-pound chip. Fit a real SLB9670 to one Pi and move
the node's key into it.

The moment the key lives in silicon that won't give it up, the whole
thing stops being a
simulation of trust and becomes the real article.

THE LANDLORD, AND THE OPEN DOOR

We started with a taxi ride: two people two feet apart, forced to route
the whole
transaction — the booking, the payment, the record — through a

company on another continent
that took a fifth and held the off switch. Four bars held them: they
needed the platform,
they paid the cut, they surrendered the record, and they lived under
the switch.

Look at what you can now run. The platform is gone — the node on
the shelf is the platform.

The record is gone from the company's database and into a chain that
its own owner can't

rewrite and a stranger can verify cold. The switch is gone — no one
holds it, because no one

is in the middle; trust is a count of independent chips, and the
memory survives any node you

take away. The door is gone too — membership is granted by any k
neighbours, proven before

it's believed, and lost to arithmetic if you cheat, founders included.

And the cut has

collapsed from a corporation's rent to a few watts and a fair penny
for the person who keeps

the box alive. A community that shares a mesh and a handful of
five-pound chips can host its

own economic life, and there is no one in the middle to ask, to pay,
or to shut it down —

and no one in the porch deciding who gets to exist.

One honest sizing note from the eviction demo, because a builder should carry it: a five-node founding committee that loses two members to eviction can no longer certify at a threshold of three. Small meshes need spare founders the way small villages need spare keyholders. That's not a flaw in the design; it's a fact about arithmetic, and now you know it before you size your street.

I don't think this replaces every platform on earth, and I'm not claiming it does. I think it removes a false assumption — that people must have a landlord in the middle to transact, remember, and trust — and hands the alternative to anyone who wants to build with it. I built the working core on my own, on ordinary hardware, without a data centre, because I'd rather better people build the village on this foundation now than watch the middle stay rented forever.

The door is open. You've watched the four bars fall in code you can run yourself. What a neighbourhood does once nobody stands in the middle of it — that's up to whoever builds next.

Go and put a node on the shelf.

*The code for the foundation of this book is in the accompanying vault-ledger prototype:

the node, the chain, the TPM attestation, the gossip and quorum, the encrypted mesh, the

radio with its packet recovery, the cluster provisioning, the Pi installer — and the front

door: the join protocol, probation, the fraud proofs, and the agent that boots a stranger

into the mesh with nobody's permission. Run it, fork it, and build the village I haven't

built yet.*