

THE LOST ALGORITHMS

Volume Four — Cryptography & Data Hiding

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

Preface: The Mathematics That Guards Everything	4
How to Read This Book	7
A Note on Real-World Cryptography (Read This One)	9
Part I — Secrecy: Keeping a Message Private	11
The Endless Coin-Flip (Linear Feedback Shift Registers)	12
Agreeing on a Secret in the Open (Authenticated Key Exchange)	18
A Lock Anyone Can Close, Only You Can Open (RSA Signatures and Certificates)	24
Part II — Trust: Proving Without Revealing	30
A Fingerprint for a Whole Library (Merkle Trees)	31
Keys That Forget Their Past (Forward-Secret Key Ratchets)	37
Proving It's You Without Storing You (Tamper-Evident Identity Verification)	42
Part III — Concealment: Hiding the Message Itself	48
Hiding in Plain Sight (LSB Steganography)	49
Appendix — The Patents of This Volume	55
A Closing Note on the Companion Code — and a Warning	58
Where the Series Goes Next	60

PREFACE: THE MATHEMATICS THAT GUARDS EVERYTHING

Every private thing you do on a computer is protected by a handful of ideas most people never see. The padlock in your browser's address bar, the message only your friend can read, the update your phone trusts because it is signed, the password that is checked without ever being stored — all of it rests on a small body of mathematics worked out in the last third of the twentieth century, and then, crucially, patented.

That patenting is the thread this book pulls. Cryptography has an unusual history for a field of pure mathematics: many of its foundational techniques were not just published in papers but claimed in patents, because they were commercially explosive. The public-key revolution of the 1970s — the almost magical discovery that two strangers could agree on a secret in full public view, and that a message could be locked with a key everyone knew and

opened only with a key nobody else had — was patented by its inventors at Stanford and MIT. Those patents shaped the industry, were fought over, licensed, and resented. And then, one by one, they expired.

When the core Diffie–Hellman patent lapsed in 1997 and the famous RSA patent in 2000, it was front-page news in the technology world, because a twenty-year monopoly over the mathematics of privacy had ended and the ideas became free for anyone to build on. This book is a guided tour of that inheritance and the specific, datable filings around it — the stream generators, the key exchanges, the signatures and certificates, the tamper-proof trees, the forward-secret ratchets, the private identity checks, and the art of hiding a message inside another — each told as the story of a patent that has expired into the public domain, and each handed to you as code you can run.

A special honesty is owed here, more than anywhere else in the series. Cryptography's great ideas have names — Diffie, Hellman, Merkle, Rivest, Shamir, Adleman — and famous foundational patents of their own. Where a chapter cites a specific industrial patent, the Story is scrupulous about what that patent actually claimed versus what the underlying idea was and who first had it. This book never says "patent US-X invented public-key cryptography." It says who did, when, in which foundational (and now also expired) filing, and what the patent in front of us added. History with receipts matters most where the history is famous.

And every line runs on nothing. No cryptography library, no OpenSSL, no cloud — just Python's standard library. The key sizes here are small so the demonstrations are fast and readable; this is a book for understanding how these mechanisms work, not a hardened library for guarding real secrets (a caution repeated where it counts). But the algorithms are the real algorithms, and you can run every one on a five-pound board and watch the mathematics of privacy work in front of you.

HOW TO READ THIS BOOK

No mathematics beyond high-school algebra, and every idea is built from zero — modular arithmetic, hashing, and public keys are all explained as they arrive, assuming nothing. Install the companion library:

```
pip install patentml
```

Every chapter names the exact module it draws from, and every result printed in these pages came from a real run of that module's demo. When the text says an attacker's forged certificate was rejected: True, that is what the code did when the chapter was written. Run it and you will see the same.

Read in order or dip in anywhere; each chapter is self-contained. The book moves through three ideas of security in turn: secrecy (keeping

a message private), trust (proving something is genuine without necessarily hiding it), and concealment (hiding the very existence of a message).

A NOTE ON REAL-WORLD CRYPTOGRAPHY (READ THIS ONE)

This book teaches how cryptographic algorithms work by implementing them plainly, at small sizes, in readable Python. That is exactly the wrong thing to guard real secrets with. Production cryptography uses large, carefully chosen parameters and hardened, audited libraries that defend against a zoo of subtle attacks — timing side-channels, padding oracles, weak randomness — that a teaching implementation deliberately ignores for clarity. Learn from this book; do not protect anything valuable with its code. When you need real security, use a vetted library (and this book will have made you a far better judge of how to use it well). Understanding the machinery is the goal here; the machinery you deploy should be someone's audited life's work, not a chapter's illustration.

With that said — how do two people who have never met agree on a secret while everyone is listening? That impossible-sounding trick is

where modern cryptography begins, and we will get there in Chapter 2. But first, the oldest idea of all: a machine that flips an endless, unpredictable-looking coin.

PART I — SECRECY: KEEPING A MESSAGE PRIVATE

The oldest goal of cryptography is the simplest to state: let two people share a message that a third cannot read. For thousands of years the answer was a shared secret — a codebook, a cipher key — that both sides knew in advance and no one else did. This part tells how the twentieth century transformed that ancient goal. It begins with the humble engine that produces the endless stream of unpredictable-looking bits a cipher needs, moves to the astonishing 1970s discovery that two strangers can agree on a secret in full public view, and arrives at the public key — a lock anyone can close but only its owner can open. Three chapters on the machinery of secrecy.

CHAPTER 1

THE ENDLESS COIN-FLIP (LINEAR FEEDBACK SHIFT REGISTERS)

THE FILING

US Patent 5,090,035 — "Linear feedback shift register."
(with US Patent 5,363,448 on related sequence generation)
Filed 1990. Expired; public domain.

Before you can hide a message you need a source of bits that look random — a long, unpredictable-seeming stream to mix into your data. Generating that stream cheaply, in hardware or a few lines of code, is the job of the linear feedback shift register, and this patent concerns its construction. Its claims describe a shift register — a row of bits that marches forward each step — with a feedback rule that computes the incoming bit from a fixed combination of the others, producing a sequence of enormous length from a tiny amount of state. It is the endless coin-flip at the bottom of stream ciphers,

scramblers, and much more.

THE IDEA

Picture a short row of bits — say eight of them — and a clock. On every tick, all the bits shift one place to the right; the bit that falls off the end is the output, and a new bit marches in at the left. The only question is what that new bit should be, and the LFSR's answer is beautifully simple: it is the exclusive-OR (the parity) of a fixed handful of the register's current bits, called the taps. XOR a few chosen positions together, feed the result back in, shift, repeat. That is the whole machine.

From that trivial rule comes a remarkable property. Choose the tap positions correctly (they correspond to a "primitive polynomial," but you can just look them up) and an n -bit register cycles through every `non-zero pattern before repeating` — a period of bits give a sequence 255 steps long; thirty-two bits give a sequence over four billion long; and the output stream is balanced (almost exactly as many ones as zeros) and passes many statistical tests for randomness, despite being produced by nothing but shifting and XOR. This is a maximal-length sequence, or "m-sequence."

To use it for secrecy, you run the LFSR to produce a keystream and XOR it into your message — a stream cipher. Because XOR is its own inverse, the receiver, running the same LFSR from the same

starting state, produces the same keystream and XORs it back out to recover the message. The shared secret is just the register's tap choice and starting fill. (A caution the chapter will return to: a bare LFSR is predictable to anyone who sees enough output, so it is a teaching cipher and a building block, not a safe cipher on its own.)

THE STORY

The linear feedback shift register is old and deep. The theory of shift-register sequences was worked out by Solomon Golomb and others from the 1950s and 60s — foundational, long-public-domain mathematics with roots in coding theory and even earlier in the study of recurrences. What patents like US5090035 cover, decades later, are specific constructions and applications of LFSRs in hardware and communications systems, where their cheapness and good statistical properties made them indispensable. And indispensable they are: LFSRs generate the spreading codes in GPS (every satellite has its own m-sequence, which is how your receiver tells them apart), the scrambling sequences in CDMA mobile phones and in wired and optical links (data is XORed with an LFSR stream to avoid long runs of identical bits that confuse the hardware), the pseudorandom patterns used to test chips, and the keystreams of many stream ciphers. The endless coin-flip is one of the most widely deployed algorithms in the world, humming inside satellites, phones, and modems — and its mathematics has been free for the taking all along.

THE CODE

The engine is a shift plus an XOR of the tapped bits. Here is the step from patentml's lfsr:

```
def step(self):
    out = self.state & 1          # the bit falling off the end
    feedback = 0
    for t in self.taps:          # XOR the tapped positions
        feedback ^= (self.state >> t) & 1
    self.state = (self.state >> 1) | (feedback << (self.degree - 1))
    return out
```

The demo runs a degree-8 register and then scrambles and descrambles a message with it. It prints:

LFSR maximal-length sequences (from US5090035 / US5363448, expired):

1. Degree 8: period should be $2^8-1 = 255$
measured period : 255
balance : 128 ones / 127 zeros (m-seq has exactly $2^{(n-1)}$ ones = 128)
2. Scramble then descramble: recovered = True (scrambled looks like: [1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1])

Both properties are exactly as the theory promises. The eight-bit register ran for 255 steps — $2^8 - 1$, every non-zero value repeating, confirming it is a maximal-length sequence. Its output was balanced: 128 ones to 127 zeros, the precise count an m-sequence must have. And the cipher round-trips: a message XORed with the keystream came out looking like random bits, and XORing the same keystream back recovered it perfectly. A row of eight bits, a clock, and an XOR produced a 255-long balanced pseudorandom stream and used it to scramble and unscramble a message. The module fuzz-tests periods and the scramble/descramble round-trip across many tap configurations and reports PASS.

THE INHERITANCE

The LFSR is public domain — its mathematics always was — and it is the tool whenever you need a long, cheap, statistically-flat stream of bits from almost no state: scramblers that keep a data link balanced, pseudorandom test patterns, spreading codes, the pseudorandom core of a simple stream cipher or noise generator. It runs in a handful of instructions and a handful of bits, which is exactly why it lives inside so much hardware. The one lesson to carry out of this chapter is its limitation: an LFSR is linear, and linearity means predictable — see enough output and you can solve for the rest, which is why real stream ciphers add non-linear mixing on top. But as an engine of pseudorandomness and a first taste of how a tiny deterministic machine can imitate a coin-flip, it is

unbeatable, and it is the right place to begin, because everything that follows needs a source of unpredictability to build on. That endless coin-flip is now yours.

CHAPTER 2

AGREEING ON A SECRET IN THE OPEN (AUTHENTICATED KEY EXCHANGE)

THE FILING

US Patent 5,371,794 — "Method and apparatus for privacy and authentication in wireless networks" (the Station-to-Station protocol).

Filed 1993. Expired 2013; public domain.

Chapter 1's cipher needs both sides to share a secret key. But how do two people who have never met agree on a secret key while an eavesdropper listens to every word they exchange? The answer — one of the most beautiful and consequential ideas in all of computing — is public-key key exchange, and this patent concerns the crucial hardening of it: an exchange that also authenticates, so that the stranger you agree a secret with is provably who they claim to be, not an impostor sitting in the middle. It is the mathematics that

begins every secure connection you make.

THE IDEA

The problem sounds impossible. Alice and Bob want a shared secret; Eve hears everything they send. Surely any key Alice can compute from what she sends, Eve can compute too? The 1976 breakthrough of Diffie–Hellman key exchange showed that intuition is wrong, using a "one-way" mixing operation — easy to do, ruinously hard to undo.

Here is the trick in plain terms. Alice and Bob publicly agree on a shared "paint colour." Each privately picks a secret colour, mixes it with the public one, and sends the mixture across the open channel. Crucially, given a mixture, you cannot separate out the secret colour that went into it — that is the one-way part (in reality it is modular exponentiation, where mixing is easy but un-mixing means solving the "discrete logarithm," which is infeasible). Now Alice takes Bob's mixture and stirs in her own secret; Bob takes Alice's mixture and stirs in his. Both end up with the same final colour — the public base plus both secrets — but Eve, holding only the two public mixtures, cannot get there, because she is missing a secret and cannot un-mix to find one. Two strangers now share a secret they never sent.

There is a catch, and it is the whole point of this chapter's patent. Plain Diffie–Hellman authenticates nobody. Eve can sit in the

middle, run a separate exchange with each of Alice and Bob, and relay between them — a man-in-the-middle attack. Alice thinks she shares a secret with Bob; really she shares one with Eve, who reads and forwards everything. The fix is to authenticate the exchange: each party signs their contribution with a long-term identity key, so the other can verify that the mixture really came from the person they meant to talk to. The Station-to-Station (STS) protocol this patent describes does exactly that — Diffie–Hellman for the secret, digital signatures for the identity — turning a private-but-anonymous exchange into a private-and-authenticated one. Secrecy without authentication is a door with a strong lock and no idea who is on the other side.

THE STORY

The honesty this chapter owes: Diffie–Hellman key exchange was invented in 1976 by Whitfield Diffie and Martin Hellman (building on ideas from Ralph Merkle), and patented in the foundational US Patent 4,200,770, which expired in 1997 — an event the cryptography world watched closely, because it freed the basic mathematics of public key agreement. The idea itself is one of the landmark discoveries of the century. What US5371794, filed in 1993, contributes is the authenticated exchange — the Station-to-Station construction that welds signatures onto Diffie–Hellman to defeat the man-in-the-middle — hardened for the then-new problem of wireless networks, where anyone could listen

and anyone could impersonate. That authenticated-exchange idea is the direct ancestor of the handshakes that secure the modern internet: every time your browser establishes an HTTPS connection, it runs an authenticated key exchange in this lineage (today's TLS uses Diffie–Hellman variants with certificate-based authentication), agreeing on a fresh secret with a server it has never met and verifying that server's identity in the same breath. The 1976 idea made private connections possible; the authenticated version made them safe.

THE CODE

The exchange agrees a shared secret; signatures bind it to real identities. The demo runs an honest exchange and two attacks, so the difference authentication makes is visible. It prints:

Authenticated key exchange (from US5371794, expired 2013):

1. Honest Station-to-Station:

shared key agrees : True

both authenticated: True

2. Plain Diffie–Hellman under man-in-the-middle:

Alice & Bob agree : False (should be False)

Mallory reads both: True -> PRIVACY SILENTLY BROKEN

3. Same attack against authenticated STS:

Alice accepts imposter: False -> ATTACK DETECTED

Read those three results as a single argument. In the honest run, STS works: Alice and Bob agree on the same secret key, and each has cryptographically verified the other's identity. Then comes the lesson. Under a man-in-the-middle attack, plain Diffie–Hellman fails silently and catastrophically — Alice and Bob do not end up with the same key (each unknowingly shares one with the attacker, Mallory), and Mallory reads everything, with neither victim aware. Run the same attack against authenticated STS and it is caught cold: Alice refuses the impostor, because the signature does not match the identity she expected. The patentml module verifies the honest agreement, the silent break of unauthenticated exchange, and the detection by the authenticated one, reporting PASS.

THE INHERITANCE

Authenticated key exchange is public domain — the foundational Diffie–Hellman patent since 1997, this authenticated variant since 2013 — and it is the beginning of nearly every secure conversation. Any time two parties who do not already share a secret must establish a private channel — a browser and a web server, two chat clients, two devices pairing, two nodes in a network — this is the mechanism: agree a fresh secret in the open, and authenticate so you know who you agreed it with. The chapter's enduring lesson is the

one the three demo results teach at a glance: secrecy without authentication is an illusion, because an unauthenticated secret might be shared with an attacker. Understanding this is understanding the deepest reason the padlock in your browser can be trusted — not merely that the connection is encrypted, but that the other end proved who it was while the key was formed. That idea, and a working authenticated exchange to study, are yours now that both its patents have expired.

CHAPTER 3

A LOCK ANYONE CAN CLOSE, ONLY YOU CAN OPEN (RSA SIGNATURES AND CERTIFICATES)

THE FILING

US Patent 4,868,877 — "Public key/signature cryptosystem with enhanced digital signature certification."

Filed 1988. Expired 2008; public domain.

Chapter 2 let two parties agree a shared secret. This chapter is about the other half of public-key cryptography: a key that comes in two halves, one public and one private, such that anyone can lock a message (or verify a signature) with the public half, while only the holder of the private half can open it (or produce the signature). This patent concerns public-key signatures and their certification — binding a public key to an identity through a trusted authority. It is the machinery of digital signatures and the certificates that make the padlock in your browser mean something.

THE IDEA

A public-key cryptosystem gives each person a pair of keys with a magical relationship: what one key does, only the other can undo, and knowing one does not reveal the other. Call them the public key (published to the world) and the private key (kept secret). The pair supports two world-changing operations.

Encryption to someone: anyone can lock a message with your public key, and only you, with your private key, can unlock it — a lock with a public keyhole that only your private key opens. Signing: you can lock a message's fingerprint with your private key, and anyone can verify it with your public key — which proves you wrote it, because only your private key could have produced something your public key unlocks. Signing is encryption run backwards, and it gives the world digital signatures: unforgeable proof of authorship and integrity.

But a signature is only as trustworthy as your confidence that a given public key really belongs to who you think. If an attacker convinces you that their public key is your bank's, they can sign as your bank. The answer is a certificate: a trusted authority (a "Certificate Authority," CA) uses its private key to sign a statement binding a name to a public key — "this key belongs to yourbank.com." Anyone who trusts the CA's public key can verify that binding. And CAs can vouch for other CAs, forming a chain of trust: a root

authority signs an intermediate, which signs a leaf, so verifying a leaf certificate means following the chain up to a root you already trust. This is exactly what happens, invisibly, every time your browser checks a website's certificate — and it is the subject of this patent's "enhanced digital signature certification."

THE STORY

The honesty this chapter owes is the most famous in the series. The RSA cryptosystem was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT, and patented in US Patent 4,405,829 — one of the most important and most-discussed patents in the history of computing — which expired on 20 September 2000. Its expiry was a genuine event: RSA Security had famously controlled the patent, licensed it widely, and even released the algorithm into the public domain two weeks early in a bit of theatre; the lapse freed the single most-used public-key algorithm on Earth. That is the foundational filing. US4868877, cited here and itself now expired, is a later (1988) patent in the public-key-signature-and-certification space — concerned with the signature and certificate machinery built on top of public-key cryptosystems. The distinction matters: RSA-the-cryptosystem is Rivest–Shamir–Adleman's landmark; the certificate infrastructure is the industrial layer that turned raw public keys into a usable web of trust. Its descendants run the world. Every HTTPS website presents an RSA-or-successor certificate chained to a trusted root; every signed software update, signed document, and

signed email uses public-key signatures; the entire "public-key infrastructure" (PKI) that lets billions of strangers trust each other online is this chapter, at planetary scale.

THE CODE

Sign a message's hash with the private key; verify with the public key; a CA signs certificates that bind names to keys, and chains verify up to a trusted root. The demo runs signing, certification, and a chain of trust — each with its forgery attempt. It prints:

RSA signatures + certification (from US4868877, expired 2008):

1. Sign / verify:
 genuine signature verifies : True
 tampered message rejected : True
2. Certificate — a CA binds a name to a key:
 CA-issued cert verifies : True
 forged cert rejected by CA : True
3. Chain of trust (root -> intermediate -> leaf):
 full chain verifies to root: True
 tampered chain rejected : True

Three mechanisms, each proven to accept the genuine and reject the

fake. A genuine signature verifies, while a message altered after signing is rejected — that is integrity and authorship. A certificate issued by the CA verifies, while a forged certificate (someone claiming a binding the CA never signed) is rejected — that is trusted identity. And a full chain of trust from a leaf up through an intermediate to the root verifies, while a chain with any link tampered is rejected — that is exactly how your browser decides a website is genuine. Every "True" is the system accepting truth; every rejection is it refusing a forgery. The `patentml` module verifies all six behaviours and reports PASS. (As the front matter warned: these keys are small for speed and clarity — real certificates use far larger ones and audited libraries.)

THE INHERITANCE

Public-key signatures and certificates are public domain — RSA itself since 2000, this certification machinery since 2008 — and they are the foundation of trust online. Any time you need to prove who wrote something, prove something has not been altered, or let strangers verify each other's identities through a shared authority, this is the mechanism: sign with a private key, verify with a public one, and bind keys to names with certificates that chain to a trusted root. It is worth grasping deeply, because it is the answer to the question the whole internet depends on — "how do I trust a party I have never met?" The answer is not that you trust them; it is that you trust a root authority that vouches, through a verifiable chain of

signatures, for them. Understanding that chain demystifies the padlock, the signed update, and the certificate warning your browser sometimes throws. The machinery of digital trust, one of the true pillars of the modern world, is now yours — and Part I closes with secrecy and identity both in hand. Part II turns from hiding messages to proving things: integrity without necessarily secrecy.

PART II — TRUST: PROVING WITHOUT REVEALING

Secrecy hides a message from those who should not read it. But a great deal of security is not about hiding at all — it is about proving: proving a document has not been altered, proving a key belongs to whom it claims, proving that you are you, proving that a secret from yesterday cannot be recovered even if today's is stolen. This part is about integrity and proof rather than concealment. It builds a single fingerprint that guards a whole library of documents, a key schedule that forgets its own past so a breach cannot reach backwards, and an identity check that verifies you without ever storing you. Three chapters on trust.

CHAPTER 4

A FINGERPRINT FOR A WHOLE LIBRARY (MERKLE TREES)

THE FILING

US Patent 5,606,609 — "Electronic document verification system and method."

Filed 1994. Expired; public domain.

How do you prove that one document, out of thousands, is genuine and unaltered — without rehashing all the others, and without revealing them? This patent's claims describe an electronic document verification system built on a tree of hashes: many documents are combined, pair by pair, into a single root fingerprint, and any one document can then be proven to belong to that root with a short proof. It is the Merkle tree, one of the most quietly powerful structures in computing — and, as it happens, a direct ancestor of the blockchain.

THE IDEA

Start with a hash function — a one-way fingerprinting machine that turns any data into a short, fixed-size string, such that the same data always yields the same fingerprint, different data almost never collides, and you cannot run it backwards. A hash proves a single document unaltered: keep its fingerprint, rehash the document later, compare. But for a thousand documents you would keep a thousand fingerprints, and to trust one you would need it delivered safely.

The Merkle tree collapses all of them into one fingerprint. Hash each document; then hash each pair of fingerprints together into a parent; then hash the pairs of parents; and so on up, halving the count at each level, until a single root hash sits at the top. That one root summarises the entire collection: change any bit of any document and its fingerprint changes, which changes its parent, and its grandparent, all the way up — the root is different. So publishing one small root hash commits you to the exact contents of the whole library.

The magic is the inclusion proof. To convince someone that a particular document belongs to a published root, you do not send the whole library. You send the document plus the handful of sibling hashes along its path to the top — for a thousand documents, about ten hashes. The verifier hashes the document, combines it with each sibling in turn up the path, and checks that they arrive at the

published root. If they do, the document is provably part of the committed set; if a single bit was altered, or the document was never in the set, the recomputed root will not match. A short proof, logarithmic in the size of the collection, verifies membership against a single trusted fingerprint — without the verifier ever seeing the other documents.

THE STORY

The honesty this chapter owes: the Merkle tree was invented by Ralph Merkle, whose foundational work on hash-based authentication dates to the late 1970s (his patent US4309569, "Method of providing digital signatures," issued 1982). The structure bears his name. US5606609, filed in 1994, is a specific document verification and timestamping application of hash-tree ideas — part of the lineage of cryptographic timestamping and digital notarisation developed in that era (the work of Haber and Stornetta on securely timestamping digital documents). That lineage is historically remarkable, because it runs straight into the most famous application of Merkle trees of all: the blockchain. Bitcoin's design uses a Merkle tree to commit all the transactions in a block to a single root in the block header, so that a lightweight client can verify a transaction's inclusion with a short proof — and the Bitcoin whitepaper cites exactly this line of 1990s document-timestamping work. Beyond blockchains, Merkle trees verify downloads in peer-to-peer systems (BitTorrent), detect tampering in file systems (ZFS, Git's tree

objects), synchronise distributed databases efficiently, and underpin "transparency logs" that catch misissued certificates. A structure for proving one document among thousands turned out to be a foundation of trust for the decentralised age.

THE CODE

Build pairwise up to a root; prove membership with the sibling hashes along one path. The demo builds a tree over nine documents and tests inclusion, tampering, and forgery. It prints:

Merkle tree document verification (from US5606609, expired):

9 documents -> one root hash: 24069ba2a3cdf2a25729138f...

1. Inclusion proof for doc #5 (4 sibling hashes):
 genuine doc verifies : True
 tampered doc rejected: True
2. Tamper the set (1 doc changed): root changes: True
3. Forged document with someone else's proof:
 rejected: True

Every guarantee holds. Nine documents collapsed to one root hash. A genuine document, presented with the four sibling hashes on its

path, verified against that root — a proof of just four hashes for a set of nine (and it would be about twenty for a million). Tampering with that document caused its proof to miss the root: rejected. Changing any single document in the set changed the root itself, as commitment requires. And a forged document waved through with someone else's proof was rejected, because its own hash does not lead to the root. Short proofs, one trusted fingerprint, tampering caught every way it was tried. The `patentml` module verifies inclusion, tamper-detection, root-sensitivity, and forgery-rejection, reporting PASS.

THE INHERITANCE

The Merkle tree is public domain, and it is the tool whenever you must prove membership or integrity over a large collection cheaply: verifying a file downloaded in pieces from untrusted peers, committing to a dataset with a single publishable fingerprint, synchronising two large stores by comparing roots and drilling only into the branches that differ, or building any "transparency log" or tamper-evident ledger. Its genius is compression of trust: one small root stands in for an entire library, and a logarithmic proof connects any member to it without revealing the rest. Understanding it demystifies not only blockchains — whose "chain of blocks each committing a Merkle root" suddenly reads as obvious — but a whole family of modern systems that need to prove integrity at scale without shipping everything. That structure, and a working Merkle

tree, are yours now that this filing has expired.

CHAPTER 5

KEYS THAT FORGET THEIR PAST (FORWARD-SECRET KEY RATCHETS)

THE FILING

US Patent 6,539,092 — "Leak-resistant cryptographic method and apparatus."

Filed 1998. Expired; public domain.

Every key in this book so far has been a fixed secret. But a fixed secret has a terrible failure mode: if it ever leaks — stolen from memory, extracted by a side-channel, seized from a device — everything it ever protected is exposed, past and future alike. This patent addresses that directly. Its claims describe deriving each new key from the previous one by a one-way step, so that a key which leaks today cannot be used to recover yesterday's keys. It is the forward-secret ratchet, the idea that makes a breach a limited disaster instead of a total one.

THE IDEA

Imagine a key that is used once and then transformed into the next key by a one-way function — a hash. Key 0 becomes Key 1 becomes Key 2, each derived from its predecessor by hashing, and because the hash cannot be run backwards, holding Key 5 tells you nothing about Keys 0 through 4. The keys march forward, and the past is sealed behind them. This is a ratchet: it turns in one direction only.

The property this buys is forward secrecy. Suppose an attacker breaks into a device today and steals its current key. With a fixed key, they could now decrypt every message ever sent — the entire history is compromised. With a ratchet, they get only the current key; every earlier key was derived by a one-way step they cannot reverse, so all past traffic stays secret. The breach is contained to the moment it happened. The patent's framing — "leak-resistant" — points at an even harsher threat model: side-channel leakage, where an attacker learns fragments of a key not by stealing it outright but by measuring a device's power draw or timing as it computes. Ratcheting the key on every use means that even a slow leak, dribbling out bits of the current key, cannot reach backwards to the keys already retired and forgotten. Each key protects its own moment and then the machine forgets it, forever.

THE STORY

The honesty this chapter owes points to a person and a moment. US6539092 comes from the work of Paul Kocher and Cryptography Research in the late 1990s, a period when Kocher stunned the field by showing that real cryptographic devices leak their secrets through physical side-channels — timing attacks (1996) and differential power analysis (1998), which could extract a smart-card's key by statistically analysing its power consumption. Suddenly the mathematics being perfect was not enough; the implementation leaked. The "leak-resistant" patents were part of the defensive response: design key schedules that ratchet forward, so that whatever an attacker skims off the current computation is useless against everything already done. The idea of forward secrecy through one-way key evolution became foundational. Its most visible descendant is the ratchet at the heart of modern secure messaging — the Signal Protocol's "double ratchet," which advances keys with every message so that compromising a phone today cannot decrypt yesterday's conversations, and which now secures the messages of billions of people through Signal, WhatsApp, and others. Forward secrecy is also why modern TLS prefers ephemeral key exchange: so that recording today's encrypted traffic and stealing the server's key tomorrow still reveals nothing. A key that forgets its past turns out to be one of the most important defensive ideas in applied cryptography.

THE CODE

Each key is the seed hashed one more time than the last; the one-way hash seals the past. Here is the ratchet from patentml's

leak_resistant_key:

```
def key(self, index):
```

```
    """The key at position index = seed hashed (index+1) times.
```

```
    One-way: no going back."""
```

```
    k = self._seed
```

```
    for _ in range(index + 1):
```

```
        k = _h(b"ratchet" + k)    # hashing forward is easy; reversing
```

```
is infeasible
```

```
    return k
```

The demo derives successive keys and encrypts a message under one of them. It prints:

Leak-resistant indexed key ratchet (from US6539092, expired):

```
key(0)[:8] : c8bd982669cd4272
```

```
key(1)[:8] : 51b6b1d9913a1064 (unrelated one-way step)
```

```
encrypt@5 :
```

```
0d45f53082e88db167fb347e6ae0253f1d9d5ac36b2d5e4de8
```

```
decrypt@5 : transfer 500 to account 8
```

The two keys tell the story. `key(0)` and `key(1)` are entirely unrelated-looking strings, even though the second is derived from the

first — because the one-way hash between them scrambles all structure, so seeing key(1) gives no foothold on key(0). That is forward secrecy in miniature: an attacker who captures key(1) cannot walk back to key(0) and the traffic it protected. The message encrypted under key 5 decrypts cleanly with the same key 5 (transfer 500 to account 8), so the scheme is usable, not just theoretical. The patentml module verifies determinism, the encrypt/decrypt round-trip, and that each index yields a distinct key, reporting PASS.

THE INHERITANCE

The forward-secret ratchet is public domain, and it is the defence to reach for whenever a long-lived secret would be catastrophic if it ever leaked — which is to say, almost any long-lived secret. Ratchet your keys forward with a one-way step and a breach becomes a bounded event: the attacker gets the present, never the past. This is the idea behind secure messaging's forward secrecy, behind ephemeral TLS, behind any system that wants "even if you steal the device, you cannot decrypt its history." The mental shift it teaches is profound and simple: stop thinking of a key as a fixed possession and start thinking of it as a moment that the system uses and then deliberately forgets. A key that cannot remember its past cannot betray it. That defensive idea, and a working ratchet to build on, are yours now that this filing has expired.

CHAPTER 6

PROVING IT'S YOU WITHOUT STORING YOU (TAMPER-EVIDENT IDENTITY VERIFICATION)

THE FILING

US Patent 5,841,886 — "Security system for photographic identification."

Filed 1996. Expired; public domain.

Verifying identity has an uncomfortable cost: to check that you are you, a system usually stores something about you — a photo, a fingerprint, a template — and every stored identity is a target to be breached, swapped, or forged. This patent's claims describe a security system for photographic identification that enrolls an identity as a feature template plus a keyed integrity tag, and at verification accepts a match only if the probe is close enough and the stored template validates as untampered. It is identity verification built to resist the two attacks that matter most: a fake trying to get in, and an

insider tampering with the records.

THE IDEA

Identity verification compares a fresh sample (a probe) against something stored at enrolment. The naive design stores the raw biometric — the actual photo or print — and checks similarity. It has two weaknesses. First, biometrics are fuzzy: two photos of the same person are never bit-identical, so you cannot check equality; you must check closeness against a tolerance, accepting genuine variation while rejecting impostors. Second, and worse, the stored record is a liability: anyone who can alter it can swap in their own template and become you in the system's eyes.

This chapter's design addresses both. At enrolment, it reduces the identity to a compact feature template — a vector summarising the distinguishing features — and computes a keyed integrity tag over that template (a keyed hash, a MAC, that only the system can produce and that changes if a single number of the template changes). At verification, two things must hold before a match is accepted. The probe must be within tolerance of the stored template — close enough to be the same person, defeating impostors whose features are too different. And the stored template's integrity tag must validate — proving the template has not been altered since enrolment, defeating an insider who tampers with the records. Similarity alone is not enough; the record itself must be proven

authentic. Genuine person and untampered template, or no entry.

THE STORY

US5841886 comes from the mid-1990s, as photographic identification and early biometric systems were being computerised for access control, credentials, and secure documents — and as it became clear that a database of stored identities is itself a high-value attack surface. The patent's dual defence — fuzzy matching for the front-door threat, integrity tags for the insider threat — anticipates a principle that has only grown in importance: a stored identity must protect itself. Its descendants run through modern identity and biometric security. The idea of never storing the raw biometric but rather a protected template that can be verified and, ideally, revoked, became the field of template protection and cancelable biometrics. The integrity-tag idea — a keyed authentication code guarding a stored record against tampering — is everywhere records must be trusted: signed and MAC-protected credentials, tamper-evident logs, the secure elements in phones that store a protected representation of your fingerprint or face rather than the image itself. The deep lesson, which this 1996 filing already embodies, is that authentication systems must defend not only the front door (is this the right person?) but the vault (has someone altered who the right person is?).

THE CODE

Enrolment binds a template to a keyed integrity tag; verification requires closeness and an intact tag. Here is verify from patentml's photo_id_verify:

```
def verify(record, probe, tolerance):
    template, tag = record
    if _tag(template) != tag:          # tampered / swapped template ->
reject outright
        return False
    return _dist(template, probe) <= tolerance # AND the probe must
be close enough
```

The demo enrolls an identity, then tries a genuine probe, an impostor, and a tampered record. It prints:

Photo-ID verification (from US5841886, expired):

```
genuine probe  accepted: True
impostor probe accepted: False
tampered template accepted: False
```

Three outcomes, three threats answered. A genuine probe — the enrolled person, with the small natural variation of a fresh sample — falls within tolerance and validates: accepted. An impostor, whose features are too far from the template, fails the closeness check: rejected. And a tampered template — one where an attacker altered

the stored record to try to let someone in — fails the integrity check before closeness is even considered: rejected, because its keyed tag no longer matches. The system admits the real person, turns away the impostor, and refuses to trust a record that has been meddled with. The patentml module fuzz-tests thousands of identities, confirming genuine acceptance, impostor rejection, and tamper detection, and reports PASS.

THE INHERITANCE

Tamper-evident identity verification is public domain, and its two-part discipline is the model for any system that checks identity or trusts stored credentials: match with a tolerance because real-world samples are fuzzy, and guard the stored record with an integrity tag because the record itself is a target. Access control, credential checking, biometric enrolment, any "prove you are who you claim" flow — all benefit from the principle that verification must confirm both the subject (are you close enough to the enrolled identity?) and the record (is the enrolled identity itself authentic?). It also points toward the modern best practice this chapter's design foreshadows: do not store the raw biometric at all; store a protected, verifiable template, so that a breach of the database does not hand attackers the actual faces and fingerprints of everyone enrolled. That principle — prove it's you without storing you — and a working tamper-evident verifier, are yours now that this 1996 filing has expired. With it, Part II closes: integrity, forward secrecy, and

trustworthy identity all in hand. Part III turns to the last idea of security, and the strangest — hiding the message so well that no one knows there is a message at all.

PART III — CONCEALMENT: HIDING THE MESSAGE ITSELF

Every technique so far has protected a message you admit exists — encrypted so it cannot be read, signed so it cannot be forged, verified so it cannot be faked. This final part is about a different and older art: hiding the message so completely that no one knows there is a message at all. Cryptography scrambles the content; steganography hides the container. A locked box announces that something is worth locking; a message written in the low bits of an ordinary photograph announces nothing. One chapter, on the quiet art of hiding in plain sight.

CHAPTER 7

HIDING IN PLAIN SIGHT (LSB STEGANOGRAPHY)

THE FILING

US Patent 6,546,112 — "Security document with steganographically-encoded authentication data."

Filed 1998. Expired; public domain.

Encryption makes a message unreadable, but it also makes it obvious — an encrypted blob loudly announces that someone had something to hide. Sometimes the goal is the opposite: to hide the very fact that a message exists. This patent's claims describe embedding authentication data into a security document steganographically — concealed in the document's own signal, invisible to the eye but recoverable by a machine. It is steganography, the art of hiding a message inside another, and its most elegant digital form: writing in the least-significant bits.

THE IDEA

Steganography and cryptography solve different halves of secrecy. Cryptography hides the content of a message; steganography hides its existence. The two compose beautifully — encrypt, then hide the ciphertext — but the hiding is the distinct trick, and its digital form rests on a simple observation: most data has room to spare in the places that matter least.

Consider a photograph. Each pixel's brightness is a number, say 0–255, and the least-significant bit of that number — the difference between an even and an odd value — is imperceptible; changing it shifts the brightness by one step out of 256, invisible to any eye. So a photograph of a few thousand pixels has a few thousand bits of hiding space that no one will ever see. LSB steganography writes a secret message into exactly those bits: take the message's bits and overwrite the least-significant bit of each carrier sample with them. The carrier looks unchanged — every value moved by at most one — but the message is now woven through it, one bit per sample. To recover it, read the least-significant bits back out. A small length header, hidden first, tells the extractor how many bits to read, so the exact message comes back out clean.

The result is a photograph, an audio clip, a document that looks entirely ordinary and carries a hidden payload. Unlike an encrypted blob, it draws no attention — which is the whole point. And when

combined with encryption, it is doubly protected: an observer sees an innocent picture; even if they suspect a message, the payload is ciphertext they cannot read.

THE STORY

Steganography is far older than computers — the word is Greek, and the practice runs from Histiaeus tattooing a message on a slave's shaved scalp in the fifth century BC, through invisible inks and microdots, to the digital age. US6546112's specific setting is a modern, important one: a security document — think identity cards, certificates, currency, tickets — carrying authentication data hidden in its own signal, so that the document silently proves its own genuineness to a scanner while showing nothing to a forger's eye. That is a hugely practical use of the idea: the hidden data is a covert watermark that authenticates the document and resists copying, because a forger reproducing the visible design does not know the invisible payload is there. Its descendants are everywhere. Digital watermarking embeds ownership and tracking information into images, audio, and video for copyright and forensics; the faint yellow tracking dots that colour printers hide in every page are steganography; covert channels smuggle data through innocent-looking traffic; and the marriage of steganography with encryption underlies techniques for hiding the existence of protected data entirely. The quiet art of hiding a message inside another turned out to be central to authenticating documents, protecting media, and

tracing leaks.

THE CODE

Write the message bits into the carrier's least-significant bits; read them back to extract. Here is the embed from patentml's `lsb_steganography`:

```
def embed(carrier, payload):
    header = len(payload).to_bytes(4, "big") # so the extractor knows
the length
    bits = _to_bits(header + payload)
    out = bytearray(carrier)
    for i, b in enumerate(bits):
        out[i] = (out[i] & 0xFE) | b        # overwrite ONLY the
least-significant bit
    return bytes(out)
```

The demo hides a short authentication string inside a 4096-byte carrier. It prints:

LSB steganography (from US6546112, expired):

```
carrier 4096 bytes, payload 21 bytes
samples altered by <=1: 118 (only LSBs touched)
extracted: auth:martin-carr:2026
```

The numbers show how invisible the hiding is. A 21-byte payload was woven into a 4096-byte carrier, and only 118 samples changed at all — each by at most 1 (only least-significant bits were touched), a perturbation no eye or ear would notice in a real image or sound. The carrier looks unchanged; nothing announces that it holds a message. Yet reading the low bits back recovered the payload exactly — `auth:martin-carr:2026`, the hidden authentication string, extracted intact. A message hidden in plain sight and pulled back out perfectly. The `patentml` module fuzz-tests three thousand carriers and payloads, confirming that embed-then-extract round-trips exactly, and reports `PASS`.

THE INHERITANCE

LSB steganography is public domain, and it is the tool whenever the goal is to hide the existence of data, not merely its content: watermarking images and audio with ownership or authentication marks, embedding covert authentication into documents, tucking metadata invisibly into media, or — combined with the encryption of Part I — hiding a protected message so it draws no attention at all. Its lesson completes the book's arc: security is not one idea but three, and this is the third. Cryptography makes a message unreadable; integrity makes it unforgeable; steganography makes it unseen. The strongest protection often uses all three — a message encrypted so it cannot be read, signed so it cannot be forged, and hidden so no one knows to look. The oldest of the three arts, the hiding, is now yours

in its cleanest digital form. The vault of cryptography's expired patents is open; go and build carefully, and remember the front matter's warning — understand these mechanisms deeply here, and deploy audited ones in earnest.

APPENDIX — THE PATENTS OF THIS VOLUME

Cryptography's ideas are famous and named, so this appendix separates two things carefully: the specific patents each chapter's code is anchored to, and the foundational filings — the landmark patents on the underlying inventions, all also long expired — that the Story sections were careful to credit. This book never claims a cited patent invented a famous idea; where it did not, the foundational filing and its inventors are named. All filings below have reached the end of their twenty-year term.

PATENTS ANCHORING THE CHAPTERS

Ch	Patent	Title	Filed
----	--------	-------	-------

----	-----	-----	-----
------	-------	-------	-------

1	US5090035 · US5363448	Linear feedback shift register / sequence generation	1990
---	-----------------------	--	------

| 2 | US5371794 | Privacy and authentication in wireless networks
(Station-to-Station) | 1993 |
| 3 | US4868877 | Public key/signature cryptosystem with enhanced
digital signature certification | 1988 |
| 4 | US5606609 | Electronic document verification system and
method | 1994 |
| 5 | US6539092 | Leak-resistant cryptographic method and apparatus
| 1998 |
| 6 | US5841886 | Security system for photographic identification |
1996 |
| 7 | US6546112 | Security document with
steganographically-encoded authentication data | 1998 |

FOUNDATIONAL FILINGS CREDITED IN THE STORY SECTIONS

These are the landmark patents on the underlying inventions —
named, dated, and now public domain — that this book is scrupulous
never to conflate with the application patents above.

Idea	Foundational patent	Inventors	Expired
Public-key / Diffie–Hellman key exchange	US4200770	Hellman,	
Diffie, Merkle	1997		
The RSA cryptosystem	US4405829	Rivest, Shamir, Adleman	
2000 |

| Hash-based digital signatures (Merkle) | US4309569 | Ralph Merkle
| ~1999 |

The expiry of the Diffie–Hellman patent in 1997 and the RSA patent in 2000 were landmark moments — the twenty-year monopolies over the core mathematics of public-key cryptography ending, and those ideas passing into the public domain. This volume is, in part, a celebration of that inheritance.

CHAPTER 8

A CLOSING NOTE ON THE COMPANION CODE — AND A WARNING

Every algorithm in this book ships in the `patentml` library:

```
pip install patentml
```

Each module names its patent, carries a runnable demo that builds its own test data, and is verified against a reference — the same `PASS` quoted at the end of each chapter's Code section, with nothing outside Python's standard library.

But read this twice: the implementations here are for understanding, with small keys and no defence against the timing side-channels, weak-randomness pitfalls, and implementation attacks that real cryptography must resist. Do not protect anything valuable with this book's code. Use it to learn how the mechanisms work — and you will use audited, hardened libraries far more wisely for having done

so. The machinery you deploy should be someone's audited life's work; the machinery you learn from is here.

CHAPTER 9

WHERE THE SERIES GOES NEXT

The Lost Algorithms of Cryptography & Data Hiding is Volume Four. Volumes One and Two told machine learning's patent history; the Computer Vision volume told the story of the pixel; Volume Three told how the world's data got fast. Ahead lie Compression (understand exactly why a file gets smaller) and Sound (how space and speech fit down a wire) — each drawn, like this one, from patents that expired into the public domain and were picked up and handed over as working code.

A patent is a bargain: a twenty-year monopoly in exchange for publishing exactly how an invention works. Nowhere was that bargain more consequential, or more fiercely fought, than in cryptography — and nowhere is its second half, the passing of the knowledge to everyone, more worth celebrating. The vault is open. Go and build — carefully.