THE IN A HURRY SERIES

Cryptography

in a Hurry

Secrets, keys, and trust

Max Brocklesby

BOOKSINAHURRY.COM

Contents

The Whole Thing in One Page

Cryptography is the art of making useful promises in the presence of an adversary. So are detecting alteration, proving possession of a key, agreeing fresh secrets across an exposed network, and arranging matters so that a stolen key does less damage than it otherwise would.

The basic move is to separate a public method from a controlled secret. A modern cipher may be published in full. Attackers may know every step and possess the software. What they should lack is the right key.

A key does not mean one thing. In symmetric cryptography the same secret underlies protection at both ends. In public-key cryptography a public value can be distributed while a private value remains controlled. Keys therefore behave like capabilities. Possessing the right one can let a machine decrypt, authenticate, sign or derive further keys. Security depends as much on where those capabilities live as on the mathematics that defines them.

Encryption on its own is rarely enough. If an attacker can change ciphertext and make the receiver accept the result, confidentiality has protected the wrong thing. Modern network protocols therefore use authenticated encryption, typically with a nonce that must satisfy strict uniqueness rules. A repeated nonce can wreck a sound construction without anyone solving the underlying cipher.

Hashes have no decryption key. They turn arbitrary data into a fixed-size digest and are designed to make useful reversal and collision-finding infeasible. They help fingerprint files, build message authentication codes, derive keys, commit protocols to what has already happened and store passwords more safely when combined with salts and deliberately expensive password-hashing functions.

Public-key cryptography solves a different problem: how to establish security between parties who do not already share a secret. Diffie-Hellman style key agreement lets two sides derive the same secret while an observer sees the exchange. Digital signatures let a verifier test whether someone

with a particular private key authorised particular bytes. Neither result answers the human question of whose key it is. Certificates, trust stores, key directories, verification rituals and organisational controls supply that missing binding.

Protocols decide what all these primitives mean. TLS 1.3 combines negotiation, ephemeral key agreement, certificates, signatures, transcript hashes, key derivation and authenticated encryption into a channel. Secure messaging adds asynchronous key establishment and ratchets so that keys change from message to message. Storage encryption adds another problem: where the release key comes from when the machine itself is asleep.

This is where cryptography becomes engineering rather than a catalogue of algorithms. Randomness can fail. Keys can be copied. Software can leak through timing or memory. Backups can bypass end-to-end encryption. A valid signature can authenticate malicious software. A secure channel can terminate at a compromised device. Cryptography protects defined properties between defined boundaries. It does not turn a hostile system into a trustworthy one.

And none of it lasts for ever. Algorithms age, assumptions weaken and computing changes. Large quantum computers would threaten the public-key mathematics behind RSA and widely used elliptic-curve systems, which is why post-quantum standards are already being deployed and tested before such machines exist. Good cryptography therefore includes an exit route.

The subject is less about hiding messages than about controlling who can do what, under which assumptions, for how long. Secrets matter. Trust enters wherever the mathematics stops.

That is the book.

Why You Should Care

Open a banking site on airport Wi-Fi. The network around you is shared with strangers, operated by organisations you do not know and connected through equipment you have never inspected. Some parts of the route can see your traffic. A malicious access point may be able to delay, copy, redirect or replace packets. Yet your browser can establish a channel in which altered records are rejected and the server proves control of a private key associated with the bank's name.

The remarkable part is what the protocol refuses to assume. It does not need the Wi-Fi operator to be honest. It does not need the routers between you and the bank to keep secrets. The network can be treated as hostile and useful communication can still emerge from it. That inversion is one of the central achievements of computer security.

The same ideas decide whether your phone accepts a software update, whether a messaging service can read the contents of a conversation, whether a stolen laptop reveals its files, whether a password leak becomes catastrophic, whether a browser accepts the wrong server certificate and whether a signed document can be checked years after it was sent.

It is also routinely described badly. A company says that data is encrypted, as though that settles the matter. It does not. Encrypted where? In transit, at rest, or end to end? Who holds the decryption key? Can an administrator reset it? Does a cloud backup contain a second copy under a different key? What detects tampering? Which identity was authenticated? What happens after a device is compromised? How long must the information stay confidential?

Those questions reveal why the subtitle matters. Secrets are only one part of the design. Keys distribute technical power. Trust determines why one key should be believed rather than another.

Cryptography also offers a disciplined way to think about adversaries. Instead of asking whether a system feels secure, define what the attacker

can do. Can they observe traffic? Alter it? Choose inputs? Steal one device? Obtain yesterday's key? Trick a user into accepting a new identity key? Security then becomes a property that should survive a stated set of attacks. If the attacker model changes, the promise may change with it.

This makes several apparently magical results understandable. Two strangers can derive a shared secret without sending that secret. A short authentication tag can cause an altered message to be rejected. A signature can be checked by anyone who has the public key. A messaging ratchet can delete old message keys so that stealing the phone today does not automatically reveal last month's conversation. None of these mechanisms depends on obscurity.

The failures are equally instructive. The mathematics can be flawless while the system fails because a random number repeats, a private key is copied, a certificate is issued to the wrong party, a password has too little entropy, a comparison leaks timing information or decrypted data remains in memory. Applied cryptography is full of attacks that go around the proof rather than through it.

There is a political dimension too. A service that holds recovery keys can restore access, moderate some abuse and comply with certain legal demands. A service designed so that only users hold the relevant keys removes those capabilities and accepts the cost of irrecoverable loss or harder moderation. Technical architecture therefore allocates power before any policy argument begins.

The useful goal is not to become a cryptographer in an hour. It is to acquire the mental model that prevents vague security language from doing your thinking for you. By the end, you should be able to distinguish secrecy from integrity, a key from an identity, a primitive from a protocol, and strong mathematics from a strong system.

The Core Ideas

1. Security Lives in the Key, Not the Secret Method

For most of history, ciphers were entangled with secrecy about the method. A substitution alphabet, codebook or mechanical setting could protect messages while the method remained unknown. That arrangement scales badly. Methods leak. Equipment is captured. Employees leave. Software can be copied. A system whose safety depends on nobody discovering how it works has made the largest part of itself into a secret.

Auguste Kerckhoffs gave cryptography a better design rule in the nineteenth century: the system should remain secure even if everything about it except the key is known. Claude Shannon later expressed the same adversarial assumption in modern information theory. Today the rule is so fundamental that a proprietary cipher advertised as safer because its design is hidden should trigger suspicion rather than confidence.

AES shows the alternative. Its structure is public. The transformations, key schedule and test vectors are available to anyone. Researchers have spent years trying to find attacks that beat brute force in meaningful settings. Users do not need to invent a new algorithm for every secret. They need independent keys generated with adequate randomness and handled correctly.

This shifts the economics. Algorithms become common infrastructure; keys become replaceable local secrets. If one user's key is stolen, the whole algorithm need not be abandoned. A session can use a fresh key. A device can have its own key. A compromised certificate can be replaced. Security becomes modular enough to operate at scale.

The strength of a symmetric key is easiest to see as a search problem. A uniformly random 128-bit key has 2^128 possible values. Exhaustive search across that space is beyond realistic classical computation. The practical attacker therefore prefers cheaper routes: steal the key, guess a weak password that protects it, exploit a protocol mistake, abuse a repeated

nonce, read it from memory or infer it through a side channel.

This distinction also explains why passwords are poor substitutes for cryptographic keys. A 128-bit random key is chosen from an enormous space with equal probability. Human passwords are chosen from a much smaller and badly distributed space. Adding a password to a strong cipher does not create a strong key unless a password-based key derivation process deliberately slows guessing and the password itself has enough unpredictability.

Perfect secrecy sits at the extreme. A one-time pad can make ciphertext reveal no information about the plaintext when the key is uniformly random, as long as the message, used exactly once and kept secret. The result is mathematically stronger than ordinary computational security. It is also operationally awkward because distributing and protecting the pad is at least as demanding as protecting the message channel. The perfect cipher pushes the problem into key management rather than abolishing it.

The larger lesson is that a key is a concentrated security dependency. It should be small enough to protect, easy enough to replace and separate from the public machinery that uses it. The algorithm can survive being known. The key cannot.

This public-method principle also creates a social process around cryptography. Proposed algorithms can be published, attacked by independent teams and compared before they become infrastructure. The AES competition is a good example. NIST did not ask the world to trust a cipher designed in secret. It solicited candidates, published them and subjected them to years of analysis before selecting Rijndael. That process did not prove AES immortal, but it concentrated scrutiny before deployment and left a public record of the assumptions being made.

Cryptanalysis then becomes part of engineering rather than evidence that the field has failed. A reduced-round attack, a related-key result or a new implementation technique can improve understanding even when it does not threaten ordinary use. The meaningful question is whether an attack

crosses the boundary from theoretical improvement to practical loss under realistic parameters. Public algorithms let that discussion happen before a crisis.

2. Cryptography Protects Properties, Not a Vague State Called Secure

People often talk about encryption as though it were the whole subject. It is one property among several. A system can keep data confidential while failing to detect alteration. It can authenticate a machine without knowing which human is using it. It can prove possession of a signing key without proving the signer's intent. Different promises require different mechanisms.

Confidentiality means unauthorised parties should not learn the protected content. Integrity means unauthorised changes should be detected. Data-origin authentication means the receiver gains assurance that data came from someone possessing the relevant secret or private key. Freshness means an old valid message should not be accepted as a new one. Forward secrecy limits the damage if a long-term key is stolen later. Post-compromise security asks whether a protocol can regain protection after an attacker temporarily learns current state.

These properties can conflict or leave gaps. Encrypting a bank instruction with a malleable scheme may hide the amount while permitting an attacker to change it. Signing a file may reveal who authorised the bytes while leaving the file public. End-to-end encryption can hide content from a provider while leaving timing and recipient information visible. A protocol that protects yesterday's messages from today's compromise may still expose tomorrow's messages until fresh entropy enters the system.

Modern authenticated encryption exists because confidentiality without integrity is a dangerous half-solution. An AEAD construction, authenticated encryption with associated data, protects plaintext and produces an authentication tag. It can also authenticate visible metadata that must not be encrypted, such as a record header. The receiver verifies the tag before accepting the plaintext.

AES-GCM and ChaCha20-Poly1305 are widely used examples. Their internal designs differ, but both rely on a crucial extra input: a nonce. The nonce generally does not need to be secret. It does need to satisfy the construction's uniqueness requirement under a given key. Reusing a nonce with some schemes can reveal relationships between plaintexts, enable forgery or expose authentication keys. A random-looking value is therefore not automatically safe; the rule depends on the construction.

This is a recurring pattern in cryptography. Security properties come with preconditions. A signature scheme may require unique or deterministic per-message values. A password hash needs a salt. A key-exchange protocol needs authentication if it is to resist an active man-in-the-middle attacker. A strong primitive does exactly the job defined for it and no more.

That precision is why good cryptographic reasoning starts with verbs. Hide what? Detect which alteration? Authenticate possession of which key? Prevent replay for how long? Recover after what compromise? The phrase secure system compresses all of these questions into one word and loses the boundaries that matter.

Block ciphers make the same lesson concrete. AES transforms one 128-bit block, which is far too small to be a complete file-encryption system. Modes such as GCM specify how many blocks, nonces, counters and authentication data fit together. Stream ciphers such as ChaCha20 generate a keystream that is combined with plaintext, while Poly1305 supplies authentication in the common AEAD pairing. The primitive category matters less to a user than the resulting property and the conditions under which it holds.

Freshness deserves special attention because a message can be perfectly authentic and still be wrong in time. If an attacker records a valid instruction saying transfer £100 and replays it tomorrow, the cryptography may verify every bit. Sequence numbers, timestamps, nonces, challenges or application-level transaction identifiers are needed to make old truth distinguishable from new authority.

3. Hashes, MACs and KDFs Turn Secrets Into Structure

A cryptographic hash function takes an input of almost any length and returns a fixed-length digest. For SHA-256 the digest is 256 bits. The function is deterministic: the same input produces the same digest. It is designed so that reversing a digest to find a useful input is infeasible, and so that finding collisions is computationally difficult.

A hash is therefore not encrypted data. There is no decryption key. The digest is evidence about an input, not a reversible container for it.

That simple object does many jobs. A software distributor can publish a file digest so that corruption is detectable. A Merkle tree can summarise a large collection of data while allowing compact proofs about individual pieces. Protocols can hash everything said during a handshake and later authenticate that digest, binding the negotiation to the final keys. Digital signature systems usually sign a digest or a structured representation rather than treating arbitrary files as raw mathematical integers.

But a bare hash does not prove who created the data. Anyone can hash the same message. To authenticate with a shared secret, systems use a message authentication code. HMAC combines a cryptographic hash with a secret key so that a party without the key should be unable to produce a valid tag for a new message. Both sides can verify the tag, which also means either side could have created it. That is useful for channels and APIs, but it differs from a public digital signature.

Key derivation functions solve another problem. Cryptographic protocols frequently begin with material that is not yet suitable as a collection of independent working keys. HKDF uses an extract-then-expand design: first turn input keying material into a strong pseudorandom key, then derive separate outputs for distinct purposes. TLS uses this kind of structure to produce different secrets for client traffic, server traffic, authentication and later updates rather than reusing one secret everywhere.

Domain separation matters here. If the same primitive and key are used in

several contexts, the protocol should label those contexts so that an output intended for one purpose cannot be confused with another. A short textual label inside a KDF or signature input can prevent two mathematically valid operations from becoming semantically interchangeable.

Passwords need a related but deliberately expensive treatment. A service should not store a raw password or a fast unsalted hash of it. If the database leaks, an attacker can guess passwords offline. A unique salt prevents identical passwords from producing identical stored values and defeats large precomputed tables. A memory-hard password hashing function such as Argon2id raises the cost of each guess by requiring substantial computation and memory. It cannot make a weak password strong, but it can make mass guessing more expensive.

Hashes, MACs and KDFs are easy to lump together because they often use the same underlying hash functions. Their security roles are different. A hash creates a public digest. A MAC creates keyed authenticity between parties sharing a secret. A KDF turns secret material into controlled new secret material. Much of protocol design consists of keeping those roles separate.

Collision resistance is also a reminder that digest length and security strength are related but not identical. For an ideal n-bit hash, generic collision search takes on the order of 2^(n/2) work because of the birthday effect, while a generic preimage search takes about 2^n. That is why a 256-bit hash is commonly described as offering about 128 bits of generic collision resistance. Real attacks can do better if the construction is weak. MD5 and SHA-1 both suffered practical collision attacks, which is why they were retired from important signature and certificate uses even though they still produce familiar-looking digests.

Hashes also enable commitment to structure. Git identifies objects by hashes. Content-addressed storage uses digests as names. Merkle trees let one small root authenticate a vast set of leaves. These are not secrecy mechanisms. They are ways of making later changes detectable and relationships compactly checkable.

4. Public-Key Cryptography Separates Powers

Symmetric cryptography has a distribution problem. If Alice and Bob already share a secret key, they can protect a channel efficiently. If they have never met, how do they obtain that secret without first having a secure channel?

Public-key cryptography changed the shape of the problem. A user creates a related pair: a public key that can be distributed and a private key that must remain controlled. The public value does not need to be secret. The system is useful precisely because knowledge can be asymmetric.

Diffie-Hellman key agreement gives the cleanest example. Alice chooses private material and publishes a corresponding public value. Bob does the same. Each combines their own private value with the other's public value and derives the same shared secret. An eavesdropper sees both public values but, under the hardness assumption of the group being used, cannot feasibly derive the shared secret.

The original idea used arithmetic modulo a prime. Modern systems often use elliptic curves because they can provide comparable classical security with much smaller public keys. X25519 is a widely deployed Diffie-Hellman function built on Curve25519. The important mental model is unchanged: public exchange, private contributions, common secret.

Unauthenticated Diffie-Hellman is vulnerable to an active intermediary. Mallory can establish one secret with Alice and another with Bob while relaying messages between them. The mathematics has created secrecy from passive observers but has not answered identity. Authentication must be added.

Digital signatures use the public-private split differently. The private key produces a signature over a message or structured digest. The public key verifies whether the signature is valid. Anyone with the public key can perform that verification, but they should not be able to forge a new signature.

This gives a different capability structure from a MAC. If two parties share a

MAC key, either can create valid tags. A third party cannot tell which one did so. With a signature, the signing capability can remain with one private key holder while verification is public. That asymmetry supports software signing, certificates and signed documents.

RSA historically served both encryption and signatures, though modern protocol design tends to separate purposes more carefully. Elliptic-curve systems brought smaller keys and efficient signatures such as Ed25519. Post-quantum systems introduce still different trade-offs in key size, signature size, speed and mathematical assumptions.

The important fact is that public-key cryptography does not replace symmetric cryptography. It changes how keys and powers are distributed. Real systems commonly use public-key mechanisms to authenticate or establish a shared secret, then use symmetric authenticated encryption for bulk data. Public-key mathematics handles the difficult introduction; symmetric cryptography handles the conversation.

There is a further distinction between key agreement and key encapsulation. Diffie-Hellman style systems have both parties contribute to a shared secret. A KEM instead lets one side use a recipient's public key to produce an encapsulation and a shared secret; the recipient decapsulates with the private key. Post-quantum migration has made the KEM abstraction especially prominent because ML-KEM is standardised in that form. Protocols can then feed the resulting shared secret into a KDF and symmetric encryption without treating the public-key primitive as a bulk-data cipher.

Key separation also protects against accidental capability expansion. A signing key should not quietly become a decryption key because an API happens to permit both operations. Separate keys, explicit algorithm identifiers and purpose constraints reduce the chance that one protocol can trick another into performing a mathematically valid operation with the wrong meaning.

5. A Public Key Is Not an Identity

Suppose a server hands your browser a public key and says, this belongs to your bank. The key may be mathematically valid. That does not make the claim true.

This is the trust problem hiding inside public-key cryptography. The mathematics can tell you that a signature verifies under a particular public key. It cannot tell you why that key should represent a company, person, domain name or software project. That binding is created by another system.

On the web, certificates are one part of the answer. A certificate binds a public key to names and other information and is signed by a certificate authority. Browsers and operating systems carry trust stores containing root certificates. A server normally presents a chain from its certificate through one or more intermediates towards a trusted root. The browser checks signatures, names, validity rules and other constraints before accepting the chain.

This arrangement scales to the public web, but it distributes trust across many organisations. A mistaken or malicious certificate authority can create danger if it issues a certificate for a domain to the wrong party. Certificate Transparency reduces the invisibility of such issuance by requiring publicly auditable logs for web certificates accepted by major browsers. Logging does not make every certificate correct; it makes unexpected issuance easier to detect and investigate.

Other systems choose different trust models. Secure messaging may display a safety number or identity fingerprint that two users can compare out of band. SSH commonly remembers a server key on first use and warns if it changes later. Organisations may operate their own internal certificate authorities. Software ecosystems may trust release keys, hardware roots or platform signing services.

Each model answers the same question differently: why should this key be believed?

The answer can never be purely cryptographic because identity begins outside the key pair. Someone must enrol the key, approve a name, compare a fingerprint, guard a root key, decide what to do when keys change or define who is authorised to sign. A valid signature therefore proves a narrower fact than everyday language suggests. It proves that the signature is consistent with possession of a private key and the signed bytes under a particular scheme. It does not by itself prove which human pressed a button, whether that human understood the document, whether they were authorised by an organisation or whether the private key had been stolen.

This is where trust enters the subtitle. Cryptography can concentrate trust, record it and make some forms of betrayal detectable. It cannot eliminate the need to decide which roots, enrolment processes and key custodians deserve authority.

Revocation exposes the difficulty. A certificate may have been valid yesterday and its private key may be stolen today. The ecosystem then needs a way to stop relying on it before the nominal expiry date. Certificate revocation lists and the Online Certificate Status Protocol were designed for this purpose, but web deployment has always had trade-offs involving privacy, latency, availability and stale information. Short-lived certificates reduce some dependence on revocation by shrinking the period for which a stolen credential remains useful.

This is a general rule: trust has time in it. A key may once have been correctly enrolled and later become compromised, reassigned or obsolete. Good systems therefore record validity periods, rotation events and replacement expectations instead of treating identity binding as permanent.

6. Protocols Are Where Cryptography Becomes a System

A cipher can be secure in isolation and still sit inside a broken protocol. The protocol decides which keys are used, how they are derived, which messages are authenticated, what counts as fresh, how failure is handled and what an attacker may replay or reorder.

History is full of failures caused by composition rather than primitive collapse. Padding oracles exploited systems that revealed whether manipulated ciphertext had valid structure. Downgrade attacks tried to make peers negotiate weaker options. Replay attacks reused valid old messages in a new context. Ambiguous encodings allowed the same bytes to be interpreted differently by different components. None required brute-forcing the underlying cipher.

TLS 1.3 is useful because it shows how many pieces a serious protocol must coordinate. A client and server negotiate parameters, establish fresh shared secret material, authenticate the server, hash the handshake transcript, derive several independent secrets and then protect application records with authenticated encryption. Each stage constrains the next. If an attacker changes the negotiation, the transcript authentication should fail. If a record is altered, the authentication tag should fail. If the server cannot prove control of the certificate's private key, the handshake should fail.

Fresh ephemeral Diffie-Hellman values give ordinary TLS 1.3 connections forward secrecy. If the server's long-term certificate key is stolen next year, recorded traffic from a properly completed ephemeral session should not become decryptable merely from that theft. The certificate key authenticated the exchange; it did not directly become the traffic-encryption key.

Secure messaging adds a harder operating condition: the recipient may be offline. A user can publish prekeys so that someone else can establish an initial shared secret asynchronously. After that, ratchets derive new keys repeatedly and delete old message keys. Signal's current protocol family also incorporates post-quantum key-establishment and ratcheting mechanisms, reflecting the same design principle under a changing threat model.

The crucial insight is that key evolution can create time boundaries. If old keys have been deleted, current compromise need not expose all past traffic. If fresh secret material later enters the ratchet, a protocol can aim to recover after temporary compromise. These properties are created by state transitions, not by choosing a larger static key.

Protocols also need explicit failure behaviour. Authentication failure must lead to rejection, not best-effort decoding. Unexpected key changes need a policy. Version negotiation must prevent silent fallback to weaker rules. Sequence numbers or nonces must stay synchronised without repeating. Error messages must not leak the exact secret-dependent condition that caused rejection.

This is why cryptography is harder than choosing strong algorithms. The primitive gives a local guarantee. The protocol gives that guarantee meaning across time, state and hostile interaction.

Formal protocol analysis exists because human intuition is weak at tracking all those states. Security proofs can model an attacker who controls the network and ask whether a protocol reduces to recognised assumptions. Automated tools can explore symbolic traces for replay or authentication flaws. Neither substitutes for deployment testing, but both force designers to state what a successful attack would mean.

Protocol versioning is part of the same problem. Supporting two versions for compatibility creates a choice point an attacker may try to influence. Secure negotiation therefore needs to bind the chosen version and algorithms into authenticated state. A system that can upgrade but cannot prove what it negotiated has created an agility mechanism that can be turned into a downgrade mechanism.

7. The Real Cryptosystem Includes Randomness, Storage, People and Replacement

A diagram of a cipher usually contains neat boxes labelled key, plaintext and ciphertext. Real systems add entropy sources, operating systems, hardware, backups, certificate enrolment, access controls, recovery procedures, software updates and people. Attackers choose whichever part is cheapest to break.

Randomness is a basic dependency. Keys, nonces, salts and ephemeral values often need unpredictable or unique values. A broken random-number generator can turn a mathematically strong scheme into a predictable one.

Past failures have produced repeated keys and repeated signature nonces, allowing private keys to be recovered without attacking the core algorithm.

Key storage is another. A full-disk encryption key must be available somewhere when the machine is open for use. It may be derived from a password, released by secure hardware, obtained from an enterprise key service or unwrapped by another key. Each option changes the failure mode. A weak login password can make a strong disk cipher easy to attack offline. A recovery key can save data and create another high-value secret. A hardware security module can reduce key extraction while becoming part of the trusted computing base.

Then come side channels. Cryptographic mathematics is normally described as if an attacker sees inputs and outputs. Real computers leak timing, cache behaviour, power consumption, electromagnetic signals and fault responses. Implementations therefore need constant-time techniques, blinding, careful memory handling and hardware-specific defences. The strongest algorithm in the book cannot protect a private key that software prints into a log file.

Finally, algorithms expire. DES became too small. MD5 and SHA-1 developed collision weaknesses that made them unsuitable for important signature uses. Protocol versions are retired. Implementations discover bugs. Long-lived systems must therefore know which algorithms they use, where the keys are, how certificates are renewed and how a new suite can be introduced without breaking compatibility or silently weakening security.

Quantum computing sharpens this maintenance problem. Shor's algorithm would break the mathematical foundations of RSA and the widely used finite-field and elliptic-curve systems if a sufficiently large fault-tolerant quantum computer were built. Symmetric cryptography and hash functions are affected differently; Grover's algorithm gives a quadratic search speed-up rather than the same kind of structural break.

NIST standardised ML-KEM for key establishment and ML-DSA and SLH-DSA for digital signatures in 2024. Migration work is happening now

because changing cryptography across protocols, hardware, certificates and long-lived data takes years. Information stolen today may still matter when future machines become stronger, creating the harvest-now-decrypt-later problem.

This closes the loop begun with public algorithms and replaceable keys. A cryptosystem designed for review should also be designed for replacement. The durable property is not that one algorithm stays safe for ever. It is that the system can change its assumptions without losing control of the secrets and trust relationships it was built to protect.

Key lifecycle management is the unglamorous machinery underneath this durability. A key has a birth, permitted uses, storage location, rotation interval, backup policy, compromise procedure and death. Long-term root keys may live offline and sign intermediates only occasionally. Session keys may exist for minutes. Message keys may be deleted after one use. Recovery keys may need dual control so that no single administrator can exercise them alone. Treating every key as the same kind of secret destroys these distinctions and makes incident response harder.

The seventh idea therefore repays the first. Once the method is public and the replaceable secret is isolated, security becomes a system that can be inspected, renewed and retired. The price is operational discipline.

How It Actually Works

A web connection from hostile network to trusted channel

Start with an ordinary HTTPS connection. Your browser knows a domain name and wants an authenticated encrypted channel to a server. It does not begin with a shared secret. The packets may cross networks controlled by parties neither endpoint trusts.

The client sends a TLS 1.3 ClientHello containing supported options and one or more key shares for ephemeral key exchange. In a common classical

configuration, that share may use X25519. The server chooses compatible parameters and sends its own key share. From the Diffie-Hellman calculation, both sides can derive the same shared secret without transmitting that secret itself.

At this point, a passive eavesdropper is in trouble, but an active intermediary still matters. An unauthenticated key exchange could be replaced with two separate exchanges. The server therefore sends its certificate chain and proves possession of the corresponding private key with a signature over the handshake context.

The browser validates the certificate. It checks that the name matches the requested domain, that the chain reaches a root it trusts under the platform's rules, that the signatures and relevant constraints are valid, and that the certificate is acceptable for the intended use. The exact web PKI rules are more detailed than this book needs, but the conceptual job is simple: bind the server's temporary handshake to a public key that the browser has reason to associate with the domain.

TLS then authenticates the transcript. The messages already exchanged are hashed into the key schedule and Finished messages. If an attacker altered the negotiation, substituted a key share or changed the certificate material in transit, the two sides should derive inconsistent authentication values and reject the handshake.

This is one of the most important patterns in modern cryptography: do not authenticate a vague session. Authenticate the exact bytes and context that created it.

The browser also has to decide what failure means. If the certificate name is wrong, the signature fails or the Finished message does not verify, the secure response is to abort. Continuing with a warning that users can click through can turn a cryptographic failure into a user-interface lottery. Protocol security therefore reaches all the way to error handling. The machine must reject invalid states before application data is treated as trustworthy.

Server authentication is common on the public web, while client

authentication is often handled at the application layer with passwords, passkeys, cookies or tokens after TLS is established. TLS can also authenticate clients with certificates, but that is a separate deployment choice. Transport security and user identity are related layers, not one mechanism.

From one shared secret to many keys

The Diffie-Hellman result is not used as a universal master key. TLS 1.3 feeds secret material and transcript hashes through HKDF, deriving separate traffic secrets for different stages and directions. Client-to-server traffic and server-to-client traffic therefore use distinct keys. Later key updates derive further traffic secrets.

This separation limits accidental cross-use. If the same raw secret were reused for encryption, authentication, exporting keys to an application and later resumption, subtle interactions would become harder to analyse. KDF labels and transcript inputs tell each derived key what job it belongs to.

Application data is then protected with an AEAD cipher. Each record has a nonce construction tied to sequence state. The receiver verifies the authentication tag before accepting plaintext. A network attacker can still drop packets, delay them or cut the connection. Cryptography cannot guarantee availability. What it can do is prevent the attacker from silently inventing accepted application records without the required keys.

Forward secrecy comes from the ephemeral exchange. The server's certificate private key authenticated the handshake. It was not the secret from which ordinary application traffic keys were directly computed. Stealing that certificate key later should therefore not reveal old traffic captured from sessions that used fresh ephemeral key agreement and whose ephemeral secrets are gone.

Resumption changes the path. A client that has connected before may use a previously established resumption secret to avoid repeating the full cost. TLS 1.3 also permits early data, often called 0-RTT, in some resumption cases. The speed comes with a security distinction: early data does not

have the same replay protection as ordinary application data. Applications must therefore avoid using it for operations whose duplication would be harmful unless they add their own replay defences.

A faster handshake is not free. The protocol exposes the price precisely enough that an application can decide whether to pay it.

Key updates are another example of limited protection rather than magic recovery. TLS can derive a new traffic secret from the current one and switch keys during a connection. This reduces the amount of traffic encrypted under one key and supports long-lived sessions. But if an attacker has already stolen the current traffic secret and continues to observe the connection, deriving the next secret deterministically from compromised state does not by itself restore secrecy. Post-compromise recovery needs fresh secret input that the attacker misses.

This is why forward secrecy and post-compromise security must not be blurred. Forward secrecy looks backwards after a later long-term compromise. Post-compromise security looks forwards after a temporary state compromise. TLS 1.3 provides strong forward secrecy in its ordinary ephemeral modes. Messaging ratchets are designed to pursue the second property as well.

A message sent to someone who is offline

A web handshake assumes both endpoints are present. Messaging has to work when Bob's phone is asleep for hours.

An asynchronous secure-messaging design therefore lets Bob publish a set of public key material in advance. Alice can fetch that material from the service and combine it with her own private contribution to derive an initial shared secret even though Bob is offline. When Bob later receives the message, his private prekey material lets him derive the same starting secret.

The server can deliver public prekeys and ciphertext without receiving the message plaintext. But the identity question remains. If the service

maliciously substitutes its own key material, or if a user's identity key changes, the parties need a way to detect or verify that change. Safety numbers, key transparency approaches and out-of-band comparison are attempts to make substitution visible rather than trusting delivery infrastructure blindly.

Once a conversation exists, keeping one static session key would be wasteful. A ratchet derives a fresh message key, uses it and deletes it. A Diffie-Hellman ratchet periodically mixes in new secret material from fresh public-private exchanges. Modern Signal specifications extend this idea with post-quantum mechanisms as well.

Two time-directed properties emerge. Forward secrecy makes it difficult to reconstruct deleted past message keys from later state. Post-compromise security aims to restore confidentiality after an attacker temporarily obtains current state, once fresh secret material enters that the attacker misses. Neither is magic. If malware remains on the device and sees every new key, no ratchet can outrun it.

This distinction is easy to miss because the user sees one uninterrupted chat. Underneath, the cryptographic state is moving constantly so that compromise at one moment does not define all moments.

Group messaging multiplies the state problem. Sending the same plaintext independently through a pairwise ratchet to hundreds of members would be expensive. Practical systems therefore use group key-management designs that balance efficiency, membership changes and compromise boundaries. When someone joins or leaves, the cryptographic state has to reflect that social event. The group name on screen is an application concept; underneath it lies a question about which devices currently possess the keys needed to speak and read.

Multiple devices create the same issue for one person. A laptop and phone may each have independent identity and session state. Linking a new device is therefore a security event, not a cosmetic synchronisation step. If a service can silently add its own device to an account, end-to-end encryption

has been defeated without touching the message cipher.

A locked laptop and the key behind the key

Storage encryption reverses the timing problem. The data and the device are together, but the system must decide when to release the key.

Imagine a laptop whose disk is protected by a randomly generated volume key. Encrypting every disk sector directly with a password-derived key would tie the whole disk to a weak and inconvenient root. Instead, many designs protect the fast random volume key with another key or access mechanism. Changing a password can then change how the volume key is wrapped without re-encrypting the entire disk.

Where does the access key come from? A password may contribute through a deliberately expensive derivation function. Secure hardware may release key material only after verified boot or local authentication. An organisation may escrow a recovery key. Some systems combine several factors.

Each design moves trust. Recovery is useful until an attacker steals the recovery database. Hardware protection is strong until the attacker compromises the trusted hardware or extracts the key after access is granted. A long password protects poorly if malware records it before the encryption software receives it.

Full-disk encryption is strongest against a powered-off stolen device. Once the user is logged in and the data is mounted, applications need plaintext and the operating system has access to working keys. The security boundary has moved from the disk to the running machine.

Database and cloud encryption add another distinction between data keys and key-encryption keys. Large datasets may use many random data-encryption keys, each wrapped under a smaller hierarchy of master keys. Rotating a master key can then mean rewrapping data keys rather than decrypting and re-encrypting terabytes of data. Enterprise key-management services and hardware security modules exist partly to

centralise control over these high-value wrapping and signing keys while limiting direct exposure.

Envelope encryption is useful, but it can be described misleadingly. If the same cloud provider controls both ciphertext and the service able to unwrap the key, the data may be protected strongly against lost disks or some internal mistakes while remaining accessible to the provider under authorised conditions. That is still encryption. It is not the same trust model as user-held end-to-end keys.

How strong cryptography fails without a cryptanalytic breakthrough

The cleanest attacks exploit assumptions the diagram left out.

In 2010 researchers analysing Sony's PlayStation 3 signature implementation found that the system had reused a nonce in ECDSA signatures. ECDSA requires a per-signature secret value with strict properties. Reuse created equations from which the long-term private signing key could be recovered. The elliptic-curve problem had not become easy. The implementation had supplied the attacker with the missing algebra.

Randomness failures have caused similar disasters elsewhere. If two devices generate the same supposedly random prime factors for RSA keys, computing a greatest common divisor across many public moduli can reveal shared factors and break both keys. Large scans of public keys have found exactly this class of problem in poorly generated embedded-device keys.

Padding oracles show a different route. Older encryption constructions could reveal, through error behaviour or timing, whether manipulated ciphertext decrypted into correctly padded plaintext. An attacker could adaptively query that tiny signal and recover information without learning the encryption key. The fix was not a larger key. It was authenticated encryption and disciplined failure handling.

Timing attacks exploit the fact that a computation may take slightly different

time depending on secret data. Cache attacks observe patterns in memory access. Fault attacks deliberately disturb a computation and analyse the wrong result. Physical cryptography therefore cares about how an algorithm runs, not merely what mathematical function it computes.

Key management failures are often simpler. A private key is checked into a public source repository. An old employee keeps a copy. A backup contains unencrypted secrets. A cloud service logs a token. A root certificate authority key is mishandled. These events may sound administratively mundane, which is precisely why they matter. The attacker does not receive extra points for solving the elegant part.

Oracles deserve one more look because they reveal how little information an attacker may need. Suppose a server returns one error for bad padding and another for a bad authentication value. Each response leaks one bit of structure about secret processing. Repeated thousands of times with carefully modified ciphertext, that tiny distinction can become a decryption tool. The lesson generalises to timing, error codes and memory access: information is information even when nobody intended it to be part of the protocol.

Cryptographic libraries therefore try to make dangerous choices hard. High-level APIs expose authenticated encryption rather than raw block-cipher modes, safe random generation rather than hand-built entropy, and standard signature encodings rather than arbitrary integer manipulation. The familiar instruction not to roll your own crypto is less about lack of intelligence than about the density of hidden preconditions. Experts reuse reviewed constructions because there are too many ways for an original composition to be subtly wrong.

The quantum migration

Public-key cryptography rests on assumptions about mathematical difficulty. RSA relies on the difficulty of factoring large integers. Classical Diffie-Hellman and elliptic-curve systems rely on related discrete logarithm problems. No efficient classical algorithms are known for the parameter sizes used securely today.

A sufficiently capable fault-tolerant quantum computer changes that model because Shor's algorithm can solve factoring and discrete logarithms in polynomial time. This is not a claim that today's quantum computers can break Internet cryptography. They cannot at the scale required. The risk is that infrastructure replacement is slow and some secrets must remain confidential for many years.

Post-quantum cryptography uses problems for which no efficient classical or known quantum attack is available. NIST's first three final PQC standards, published in 2024, are ML-KEM for key establishment and ML-DSA and SLH-DSA for signatures. These algorithms run on ordinary computers. The word quantum describes the attacker they are intended to resist, not the hardware required to use them.

Migration is more than swapping names in a configuration file. New keys and signatures may be much larger. Certificates, hardware modules, network packets, firmware formats and validation systems may have size assumptions built around RSA or elliptic curves. Old devices may be unable to update. Protocols need negotiation rules that do not create downgrade attacks. Organisations need inventories of where vulnerable public-key cryptography is embedded.

Hybrid schemes can combine a classical and post-quantum key exchange so that breaking the resulting secret requires defeating the intended combination rather than trusting a new family alone. RFC 9954, published in 2026, specifies a general construction for hybrid key exchange in TLS 1.3 without mandating one specific post-quantum algorithm pair.

The migration reveals the mature form of the subject. Cryptography is a

managed dependency on assumptions. Good systems know which assumptions they are using and can change them before the old ones become liabilities.

This is also why harvest-now-decrypt-later risk is uneven. A password reset token that expires in ten minutes does not need decades of confidentiality. Diplomatic archives, health records, trade secrets and some personal communications may. Migration priority should therefore follow both cryptographic exposure and data lifetime. The same algorithm can be urgent in one system and less urgent in another because the value of the protected secret decays at a different rate.

No one knows the exact date at which a cryptographically relevant quantum computer will exist. Forecasts vary because the engineering path depends on error correction, physical qubit quality, logical gate costs and architecture. Planning does not require that forecast to be precise. It requires recognising that replacement lead times can be measured in years and that some data must remain secret across those years.

How we know

Modern cryptography has an unusually strong evidence trail because important mechanisms are published as standards, specifications, security analyses and open implementations. AES, SHA families, NIST post-quantum algorithms and key-management guidance are public. TLS 1.3 is specified by the IETF, with RFC 9846 replacing the original RFC 8446 text in July 2026. Signal publishes specifications for PQXDH, Double Ratchet and its post-quantum ratcheting work.

Proofs and standards still have boundaries. A security proof covers a defined construction under stated assumptions. It does not certify every implementation, random-number generator, device, user interface or enrolment system. Attack papers matter because they repeatedly expose those gaps. The reliable picture comes from reading both sides: what a construction claims, and the conditions under which real systems have failed to realise that claim. The strongest evidence often comes from

disagreement between those two records. A standard tells you the intended state machine; attack research shows where implementations, APIs or deployments supplied states the model did not expect. That gap is where applied cryptography earns its engineering reputation. Reproducible test vectors and interoperability testing add a third check: independent implementations should compute the same result from the same inputs. They also expose accidental differences in encoding, byte order and edge-case handling before those differences become security bugs between independent implementations in production.

What People Get Wrong

“Encryption makes data anonymous”

Encryption hides content from parties who lack the required key. It does not automatically hide who is communicating, which server is contacted, when packets move, how large they are, which account logged in or where a device connected from. Those facts are metadata, and many systems need some of them in order to route traffic.

The confusion persists because a readable message is the most obvious thing an eavesdropper might want. In practice, communication patterns can reveal relationships and behaviour even when content remains opaque. Traffic analysis exists precisely because protected content does not erase observable structure.

Anonymity therefore requires additional techniques and a different threat model. Encryption may be one component, but the properties are distinct. Ask what the adversary can still observe after the plaintext disappears.

Even end-to-end encryption can reveal useful metadata to the service carrying the message. The provider may need to know which account should receive a ciphertext and may observe login times, device registrations and delivery attempts. Some systems minimise or protect parts of this metadata, but doing so requires additional protocol work. Confidential content and anonymous communication are separate engineering goals.

“A hash is encrypted data”

A hash digest is not ciphertext waiting for a secret key. A cryptographic hash has no decryption operation. It deterministically maps input data to a fixed-size output and is designed so that useful inversion is infeasible.

The misconception comes from passwords. Services often say passwords are hashed, then people imagine the original password has been hidden in reversible form. A correctly designed password-verification system instead stores a salted output of a password-hashing function and later tests guesses by recomputing it.

Because humans choose guessable passwords, attackers do not need to invert the hash mathematically. They can try candidate passwords until one matches. That is why salts, memory-hard functions and password strength matter.

The distinction also matters for file integrity. If you download a file and its SHA-256 digest matches a digest published on the same compromised website, you have proved little about authenticity because an attacker may have changed both. A hash becomes meaningful only through the trust path by which the expected digest reached you.

“Public-key cryptography encrypts all the data”

It can, but modern protocols commonly use public-key mechanisms for the part symmetric cryptography handles badly: introduction, key agreement, authentication or wrapping a small secret. Bulk data then moves under fast symmetric authenticated encryption.

The distinction matters because public-key operations are more computationally expensive and often have size constraints. It also clarifies hybrid encryption. The public-key layer establishes or protects a random symmetric key; the symmetric key protects the message.

Thinking in layers makes protocol design easier to understand. Public-key cryptography solves who can establish which secret. Symmetric cryptography efficiently protects the continuing stream.

This is why a short RSA or elliptic-curve operation can protect a file many gigabytes long without applying public-key arithmetic to every byte. The public-key step protects or derives a compact symmetric key. The bulk cipher then handles the file at high speed. The architecture is hybrid because the two families solve different problems. That division also lets systems replace one layer without redesigning every byte-processing operation around it. It also lets protocol designers use a KEM or Diffie-Hellman mechanism for a fresh shared secret while keeping a mature AEAD construction for the data path.

“A digital signature proves a person signed”

A valid digital signature proves that the signature verifies for specified bytes under a public key. Under the scheme's assumptions, that strongly supports the claim that the corresponding private signing capability was used.

The human statement is broader. Was the private key controlled only by one person? Was it released automatically by software? Had it been stolen? Did the signer intend to approve this version of the document? Were they authorised to act for the organisation? Cryptography does not answer those questions by itself.

Legal and organisational attribution therefore depends on key custody, enrolment, user interaction, audit records and policy. The mathematical signature is powerful precisely because its claim is narrow and testable.

The same caution applies to the word non-repudiation, which is sometimes attached to digital signatures as though mathematics can prevent a signer from later denying an action. A signature can supply strong technical evidence. Whether that evidence is legally or organisationally decisive depends on how keys were issued, controlled and used. Two organisations using the same signature algorithm can therefore reach different attribution conclusions because their key-custody procedures, approval rules and audit trails differ materially in daily practice.

“A 256-bit key is stronger than a 2,048-bit key”

Bit lengths from different algorithm families are not directly comparable. A 256-bit symmetric key, a 256-bit elliptic-curve public key and a 2,048-bit RSA modulus represent different mathematical structures and different attack costs.

RSA needs much larger numbers because the best known attacks exploit the arithmetic structure of factoring. Elliptic curves achieve comparable classical security with smaller public keys. Post-quantum systems introduce another set of sizes and assumptions.

Security strength should therefore be compared through estimated work factor against the relevant attack, not by asking which printed key is longer. More bits can mean a stronger parameter within one family, but cross-family bit counting is largely meaningless.

Key size also affects performance, bandwidth and storage. A post-quantum public key or signature may be far larger than a classical elliptic-curve equivalent while targeting a comparable security category. The extra bytes are a consequence of the construction, not evidence that the older system was thousands of times weaker.

“If the algorithm is secret, attackers have less to work with”

They may have less information temporarily. That is not the foundation a widely deployed cryptosystem should depend on.

Algorithms are hard to keep secret once software and devices are distributed. Hidden designs also miss the benefit of public cryptanalysis. A weakness discovered by researchers before deployment is far cheaper than the same weakness discovered privately by an attacker afterwards.

Kerckhoffs's rule does not say secrecy is useless. Keys, seeds and some operational details must remain secret. It says the system should not collapse when the attacker learns the method. Obscurity can be an extra hurdle. It should not be the lock.

Open design also gives defenders a chance to discover structural mistakes before deployment becomes universal. This is why major standards processes publish candidate algorithms and invite attack. A secret design may conceal a weakness from friendly reviewers more effectively than from a determined adversary who eventually obtains the implementation.

“Quantum computers will make cryptography useless”

A large fault-tolerant quantum computer would be a severe threat to widely used RSA and elliptic-curve public-key cryptography because of Shor's algorithm. It would not erase the concept of cryptography.

Symmetric algorithms and hashes are affected differently. Grover's algorithm gives a quadratic improvement for brute-force search, which can be countered in part by larger symmetric parameters. Post-quantum public-key schemes are designed around problems for which no efficient quantum attack is known.

The real challenge is migration. Cryptography is embedded in protocols, hardware and long-lived data. The risk is therefore operational as well as mathematical: can systems move to new assumptions before the old ones become exploitable?

There is also no single quantum deadline. A system protecting ten-minute session tokens has a different urgency from one protecting secrets that must survive for thirty years. The useful variable is the combination of data lifetime, migration time and exposure to collection today. A migration programme can therefore rationally prioritise long-lived confidential archives and high-value authentication systems before low-value data whose secrecy expires quickly. Quantum risk is a scheduling problem as well as a research problem. That is why organisations with long-retention secrets cannot sensibly wait for a dramatic quantum demonstration before starting the inventory and replacement work.

The phrase post-quantum can mislead in the opposite direction too. These algorithms do not require quantum computers and are not guaranteed safe

against every future discovery. They are classical algorithms selected because no efficient classical or known quantum attacks defeat their security assumptions at the chosen parameters. They still need cryptanalysis, careful implementation and replacement plans.

Use It

Name the property before admiring the algorithm

When a product says it uses strong encryption, translate the claim into properties. Is the goal confidentiality, integrity, authentication, replay resistance, anonymity, forward secrecy or recovery after compromise? Which of those does the system provide, and against whom?

This prevents a common category error. A database can be encrypted at rest while administrators retain decryption keys. A messaging service can offer end-to-end encryption while leaving contact metadata visible. A signed update can be authentic and still malicious because the authorised publisher shipped bad code.

The question “what property survives which attacker?” is more useful than “is it encrypted?”

For a practical comparison, imagine three services. One encrypts a database on disk with a key stored by the same server. Another encrypts each user's data under a key controlled by a separate key-management service. A third encrypts on the user's device with a key the provider never receives. All three may truthfully say encrypted, yet the attacker models and provider capabilities are different. Property-first thinking exposes that difference immediately.

Follow the key

Find the key that can perform the sensitive action and trace its life. How is it generated? Where is it stored? Who can copy it? What releases it? Is there a recovery copy? When is it rotated? What happens when an employee leaves or a device is lost?

This lens often reveals the true architecture faster than reading marketing material. A service that holds the decryption key has different capabilities from one that never receives it. A hardware-backed signing key has a different theft model from a file on disk. A password-derived key inherits the limits of the password.

Keys are technical power made portable. Follow them and you usually find the trust boundary. Then ask whether every copy of that key follows the same rules. A copy exported for disaster recovery can quietly bypass hardware protections that looked decisive in the primary system.

Do the same for backup and recovery. A key that exists in one secure hardware device may also exist in a recovery escrow, migration archive or printed emergency code. Security is governed by every usable copy. The weakest surviving copy may define the real compromise threshold, while the strongest copy receives the marketing attention.

Ask what must never repeat

Many cryptographic failures come from values treated as disposable details. Nonces, IVs, salts and signature randomness have different rules. Some must be unique. Some must be unpredictable. Some may be public. Confusing those requirements can destroy security.

The practical lesson is not to memorise every rule. It is to notice when a design depends on a per-message or per-session value and ask what guarantees its required property across crashes, backups, cloned devices and parallel processes.

A billion safe encryptions followed by one catastrophic nonce reuse is still a catastrophic design.

This lens is particularly useful for cloned virtual machines and embedded devices. A counter that was unique before a snapshot may repeat after rollback. A factory image copied to thousands of devices may accidentally carry identical seeds or keys. Uniqueness has to hold across the system's real lifecycle, not merely inside one running process.

Separate mathematical authentication from identity

When a signature verifies, ask what has been authenticated. Usually the answer is a key. Then ask how that key acquired its name or authority.

For a website, the chain may run through certificate authorities and a browser trust store. For a colleague's messaging key, it may depend on a safety-number comparison. For a company release key, it may depend on an internal approval process and secure signing hardware.

This distinction is useful far beyond cryptography. Evidence gains meaning through provenance. The signature gives a strong link in the chain, but the surrounding links decide what the chain means.

When an identity key changes, ask who is authorised to declare the new binding. Automatic replacement is convenient but may let a compromised directory substitute keys invisibly. Manual verification is stronger against that attack but burdens users. Trust models are partly choices about where to put friction. A high-value account may justify a conspicuous verification ceremony that would be intolerable for a casual consumer app. The correct amount of friction depends on the harm from silent key substitution, not on a universal preference for convenience or purity. Verification is strongest when the user can understand what changed and why the application is asking them to care.

Draw the endpoints

Encryption protects data between endpoints. Draw them literally. If a message is end-to-end encrypted from one phone to another, the phones are inside the trusted boundary. Malware on either phone may read the plaintext before encryption or after decryption. A cloud backup may create a different endpoint and a different key path.

This lens prevents the word encrypted from hiding where plaintext exists. Every usable system must reveal data somewhere to somebody or something. The security question is whether that exposure matches the

intended trust model.

Endpoints can also multiply without the user noticing. A browser extension, accessibility service, synced clipboard, notification preview or cloud backup may receive plaintext around an encrypted application. The cryptographic channel can be correct while the surrounding product broadens the trusted boundary. A useful architecture review therefore marks every place plaintext appears, every service that can request decryption and every path by which a new endpoint can be enrolled.

Demand an expiry plan

Ask how the system changes algorithms and keys. Can certificates be renewed? Can cipher suites be disabled? Can stored data be rewrapped under new keys? Is there an inventory of where RSA, elliptic-curve or legacy hashes are embedded? Can old devices receive updates?

Crypto agility is easy to praise and difficult to implement. Too much negotiability can create downgrade risk; too little makes migration impossible. The useful target is controlled replacement with authenticated versioning and clear policy.

A cryptographic choice without a replacement route is a future incident waiting for a date.

For organisations, the first migration tool is often an inventory rather than a new algorithm. If nobody knows which applications depend on RSA certificates, where old SHA-1 signatures remain, which devices cannot update or how long archived ciphertext must stay secret, migration plans are guesswork. Cryptographic agility begins with knowing what must change. The next step is to separate policy from mechanism. Decide which algorithms are allowed, how peers prove the negotiated choice, how exceptions expire and who can approve emergency fallback. Compatibility should have an owner and an end date. Otherwise old cryptography survives because nobody is authorised to remove it. Migration fails surprisingly often through ownership gaps rather than cryptanalysis.

The limits

Cryptography can make some attacks computationally infeasible and some tampering detectable. It cannot guarantee honest endpoints, correct software, available networks, sensible users or truthful institutions. It does not stop a recipient copying plaintext. It does not erase metadata by default. It does not decide whether a person deserves access. It cannot preserve confidentiality after the intended recipient deliberately reveals the content.

Nor does a proof turn an implementation into a theorem. Proofs depend on models. Real machines have timing, memory, power, bugs, operating systems and recovery processes. The discipline works best when those boundaries are stated rather than blurred.

The one thing to keep

Keep one question: **who can do what because they possess which key?**

That question cuts through most of the mystique. Encryption keys grant decryption capability. Signing keys grant authorisation capability. Root keys grant the power to vouch for other keys. Recovery keys grant a second path around the ordinary one. Ratchets try to make yesterday's capability disappear. Post-quantum migration changes the mathematics that protects tomorrow's capabilities.

Cryptography succeeds when these powers are narrow, inspectable and replaceable. It fails when a key quietly means more than the system admits, when identity is assumed rather than established, or when an old assumption cannot be changed.

Secrets are the data we want to protect. Keys decide who gets power over them. Trust begins where we decide which keys and systems deserve that power.

Terms

Plaintext. Data in its readable or original form before encryption. The word applies to bytes rather than only human-readable text.

Ciphertext. The output of encryption. Good ciphertext should not reveal useful plaintext information to an attacker who lacks the required key, within the scheme's threat model.

Cipher. An algorithmic construction for encryption and decryption under a key. A cipher is a primitive, not a complete secure protocol.

Key. Secret or private cryptographic material controlling an operation such as decryption, authentication, signing or derivation. Key handling often matters more than key length.

Symmetric cryptography. Cryptography in which communicating parties share secret key material. It is efficient enough for bulk data protection.

Public key. The distributable half of a public-key pair. It can support verification, key agreement or encryption depending on the scheme.

Private key. The controlled half of a public-key pair. Possession can enable signing, decryption or agreement and therefore represents a security capability.

Nonce. A value intended for one use under defined conditions. Many encryption schemes require nonce uniqueness under a given key but not secrecy.

IV. Initialisation vector. An auxiliary input used by some encryption modes. Its required properties depend on the construction and should not be guessed from the name.

AEAD. Authenticated encryption with associated data. A construction that encrypts plaintext, authenticates it and can authenticate specified visible metadata.

Authentication tag. A short value verified by the receiver to detect

unauthorised modification under the relevant key and construction.

Hash function. A deterministic function mapping arbitrary-length input to a fixed-size digest, designed for properties such as preimage and collision resistance.

Digest. The fixed-size output of a cryptographic hash function. It can act as a compact fingerprint of data but is not reversible ciphertext.

Collision. Two different inputs that produce the same hash digest. Collisions must exist mathematically for fixed-size hashes, but useful ones should be infeasible to find.

MAC. Message authentication code. A keyed tag that lets parties sharing a secret detect alteration and authenticate data origin within that shared-key relationship.

HMAC. A widely used MAC construction built from a cryptographic hash function and a secret key.

Salt. A usually public, unique value added to password hashing so identical passwords do not produce identical stored outputs and precomputation is less useful. It should be generated independently for each record.

KDF. Key derivation function. A construction that turns secret input material into one or more cryptographic keys with controlled separation between purposes. KDF labels and context inputs help prevent one derived key from being mistaken for another.

HKDF. An HMAC-based extract-and-expand KDF used in many modern protocols, including the TLS 1.3 key schedule. Its extract-then-expand structure is useful when one shared secret must safely generate several independent working keys.

Key exchange. A protocol through which parties establish shared secret material across a communication channel. Key exchange by itself does not necessarily authenticate either party.

Diffie-Hellman. A family of key-agreement techniques that lets parties

derive a common secret from private contributions and exchanged public values. Without authentication, an active intermediary can replace those public values and create two separate secrets.

Digital signature. A value created with a private signing key and checked with a public key to authenticate specified data under the scheme's assumptions. The signature authenticates a key relationship; human identity and authority still depend on enrolment and custody.

Certificate. A signed data structure binding a public key to names, uses or other attributes under a particular trust system. Certificates expire, can be revoked and may carry constraints on how their keys may be used.

Certificate authority. An entity trusted within a public-key infrastructure to issue or sign certificates subject to defined validation rules. The security of a public PKI depends on the governance and protection of these issuing authorities.

Forward secrecy. A property limiting exposure of past session data when a long-term key is compromised later. It is normally obtained by using fresh ephemeral key agreement and deleting the resulting session secrets.

Post-compromise security. A property by which a protocol can regain protection after temporary compromise once fresh secret material enters and the attacker no longer sees the state. Recovery requires fresh entropy or key material that the attacker misses.

Ratchet. A state-evolution mechanism that repeatedly derives and replaces keys, often deleting old key material to limit the time span of compromise. Ratchets are state machines, so secure deletion and correct handling of out-of-order messages matter.

Side channel. Information leaked through implementation behaviour such as timing, cache use, power consumption or electromagnetic emissions rather than through the intended mathematical interface. Constant-time code and hardware countermeasures exist because mathematical security proofs rarely model these leaks directly.

KEM. Key encapsulation mechanism. A public-key primitive that produces a shared secret and a corresponding encapsulation, now central to many post-quantum key-establishment designs. A KEM normally feeds its shared secret into a KDF and symmetric encryption rather than encrypting application data directly.

Crypto agility. The engineered ability to replace algorithms, parameters and keys without losing security or control of the surrounding system. It includes discovery, versioning, authenticated negotiation, staged rollout and retirement of old choices. Safe agility requires authenticated negotiation so that compatibility cannot be abused as a downgrade path.

Go Deeper

Jean-Philippe Aumasson, *Serious Cryptography*, 2nd ed. (No Starch Press, 2024). The best next step for a technical general reader who wants mechanisms rather than folklore. It moves from randomness and symmetric encryption through hashes, public-key systems and implementation concerns. Some code-level detail appears, but the conceptual explanations remain accessible. Use it to deepen the practical material on nonces, hashes, public-key primitives and post-quantum design without jumping straight into a graduate text. The second edition also gives useful treatment to implementation failures and newer primitives, which makes it a better bridge from this book than an older historical survey.

Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, revised 3rd ed. (CRC Press, 2025). Read this for the formal discipline behind claims such as confidentiality and unforgeability. It is mathematical and expects comfort with proofs, but it shows why modern cryptography defines attackers and security games instead of relying on intuition. The reward is a much sharper understanding of what a security claim means and why a proof is always conditional on an attacker model and mathematical assumption. Do not read it cover to cover at first. Begin with private-key encryption and message authentication, then public-key encryption and signatures, and return to the formal preliminaries as needed.

Eric Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 9846 (RFC Editor, 2026). Read the standard after the worked TLS sequence in this book. It is dense, normative and designed for implementers, but it shows exactly how key exchange, transcripts, authentication, derivation and records fit together in a deployed protocol. Read the overview and handshake sections first, then return to the key schedule after the high-level sequence is familiar. The document is also a useful example of how modern standards state failure conditions precisely. Keep the terminology section nearby. Standards prose is compact because every word carries interoperability consequences, and that precision is part of the lesson.

Ross Anderson, *Security Engineering*, 3rd ed. (Wiley, 2020). Read this for the boundary around cryptography: protocols, incentives, hardware, banking, authentication and system failure. It is much broader than this book and is the strongest antidote to thinking that mathematically strong primitives automatically create secure systems. Anderson is especially useful on key management, authentication, hardware and incentives, where cryptographic mathematics meets organisations and physical machines. It is long, but individual chapters stand well on their own. The chapters on authentication, distributed systems and hardware show why the hardest security failures so often occur outside the cipher itself.

Notes and Sources

The Whole Thing in One Page and Why You Should Care

The organising distinction between confidentiality, integrity, authentication and broader trust follows standard modern cryptographic treatment and NIST terminology. The description of TLS 1.3 uses RFC 9846, published July 2026, which obsoletes RFC 8446 while retaining TLS version 1.3. The discussion of post-quantum migration reflects NIST's final 2024 standards for ML-KEM, ML-DSA and SLH-DSA and NIST's continuing migration guidance.

Security lives in the key

Kerckhoffs's 1883 principles are the historical source for the requirement that a cryptosystem should remain secure when the system is known and only the key is secret. Shannon's 1949 paper formalised secrecy and the adversarial assumption for modern cryptography. The one-time pad's perfect secrecy result follows Shannon's information-theoretic treatment when its strict key requirements are satisfied. AES parameters and structure follow FIPS 197, updated by NIST in 2023 without changing the AES algorithm.

Properties, AEAD and nonces

The distinction between confidentiality and integrity is standard in modern security definitions. RFC 5116 defines an interface for authenticated encryption with associated data. AES-GCM is specified by NIST SP 800-38D. ChaCha20-Poly1305 for IETF use is specified by RFC 8439. Nonce requirements differ by construction; the body therefore avoids a universal claim that every nonce must be random.

Hashes, MACs, KDFs and passwords

NIST FIPS 180-4 specifies the SHA-2 family and FIPS 202 specifies SHA-3. HMAC is standardised by FIPS 198-1 and related RFCs. HKDF is specified in RFC 5869 and uses the extract-then-expand model described in the text. Argon2 is specified for Internet use in RFC 9106; Argon2id is the primary variant recommended there for password hashing where both side-channel and trade-off concerns matter.

Public keys, signatures and identity

Diffie and Hellman's 1976 paper introduced public-key distribution and key agreement ideas that transformed the field. RSA was introduced by Rivest, Shamir and Adleman in 1978. X25519 is specified by RFC 7748, Ed25519 through RFC 8032. The account of certificates and trust is grounded in the Internet X.509 public-key infrastructure specified by RFC 5280 and the web's certificate ecosystem. Certificate Transparency originates in RFC 6962 and subsequent browser policy and standards work.

Protocols and key evolution

TLS 1.3 handshake, key schedule, Finished authentication, resumption, early data and KeyUpdate follow RFC 9846. Early data has weaker replay guarantees than ordinary TLS application data, a limitation made explicit by the TLS specification. HKDF labels and transcript hashes are central to the TLS 1.3 derivation structure.

Signal's published PQXDH specification describes asynchronous post-quantum extended Diffie-Hellman key agreement. The Double Ratchet specification describes message-key evolution and Diffie-Hellman ratcheting. Signal's 2025 specifications and technical publications describe the Sparse Post-Quantum Ratchet and Triple Ratchet, which combine post-quantum and classical ratcheting properties. The body intentionally describes the design pattern rather than claiming every secure messenger uses Signal's exact protocol.

Storage, randomness and side channels

NIST SP 800-57 Part 1 provides general key-management guidance and emphasises key lifecycle, protection and compromise. The Sony PlayStation 3 ECDSA failure is a well-documented example of nonce reuse exposing a signing key; it is used as an implementation lesson rather than as a claim about ECDSA itself. Large-scale studies of RSA public keys have found shared prime factors caused by weak random generation in embedded and network devices. Padding-oracle and timing attacks are established classes of implementation and protocol attack, represented historically by work from Bleichenbacher, Vaudenay, Kocher and later researchers.

Quantum risk and migration

Shor's 1994 algorithm gives polynomial-time quantum algorithms for integer factoring and discrete logarithms, threatening RSA and standard finite-field and elliptic-curve public-key systems if sufficiently capable fault-tolerant quantum computers become available. Grover's 1996 search algorithm gives a quadratic speed-up for unstructured search, which affects symmetric key sizing differently.

NIST FIPS 203 specifies ML-KEM, FIPS 204 specifies ML-DSA and FIPS 205 specifies SLH-DSA. As of 10 August 2026, NIST states that migration away from quantum-vulnerable public-key algorithms should be completed by 2035, with high-risk systems moving earlier. RFC 9954, published April 2026, gives a general hybrid key-exchange construction for TLS 1.3 and explicitly does not mandate a specific post-quantum mechanism.

Bibliography

Primary papers, standards and specifications

Diffie, Whitfield, and Martin E. Hellman. “New Directions in Cryptography.” *IEEE Transactions on Information Theory* 22, no. 6 (1976): 644-654.

Dworkin, Morris. *Recommendation for Block Cipher Modes of Operation:*

Galois/Counter Mode (GCM) and GMAC. NIST SP 800-38D. National Institute of Standards and Technology, 2007.

Krawczyk, Hugo, and Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. RFC Editor, 2010.

Langley, Adam, Mike Hamburg, and Sean Turner. *Elliptic Curves for Security*. RFC 7748. RFC Editor, 2016.

National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. FIPS 197-upd1. 2023.

National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. FIPS 180-4. 2015.

National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS 202. 2015.

National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS 203. 2024.

National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. FIPS 204. 2024.

National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*. FIPS 205. 2024.

Nir, Yoav, and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. RFC Editor, 2018.

Rescorla, Eric. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 9846. RFC Editor, 2026.

Rivest, R. L., A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.” *Communications of the ACM* 21, no. 2 (1978): 120-126.

Shannon, Claude E. “Communication Theory of Secrecy Systems.” *Bell System Technical Journal* 28, no. 4 (1949): 656-715.

Shor, Peter W. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring.” In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994.

Signal Messenger. *The PQXDH Key Agreement Protocol*. Signal Protocol documentation, current specification accessed 10 August 2026.

Signal Messenger. *The Double Ratchet Algorithm*. Signal Protocol documentation, current specification accessed 10 August 2026.

Stebila, Douglas, Scott Fluhrer, and Shay Gueron. *Hybrid Key Exchange in TLS 1.3*. RFC 9954. RFC Editor, 2026.

Biryukov, Alex, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*. RFC 9106. RFC Editor, 2021.

Modern works

Anderson, Ross. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 3rd ed. Wiley, 2020.

Aumasson, Jean-Philippe. *Serious Cryptography: A Practical Introduction to Modern Encryption*. 2nd ed. No Starch Press, 2024.

Ferguson, Niels, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010.

Katz, Jonathan, and Yehuda Lindell. *Introduction to Modern Cryptography*. Revised 3rd ed. CRC Press, 2025.