Signal

PRIVACY GUIDE

Signal in depth

Complete guide

This guide describes how Signal works (November 2026) in enough detail to understand each of its pieces: keys, protocol, metadata, servers and limits.

From end-to-end encryption to post-quantum cryptography

Contents

INTRODUCTION AND BASICS

· How to use this guide 3

· The basic concepts 3

· The minimum maths 5

CHAPTERS

· Notation 8

01 Overview: what Signal protects and what it does not 9

02 The cryptographic primitives 11

03 Identity keys and prekeys 12

04 X3DH: the initial handshake 13

05 PQXDH: the post-quantum handshake 16

06 The Double Ratchet 18

07 SPQR and the Triple Ratchet 20

08 Messages, attachments and several devices 22

09 Sealed Sender and metadata 24

10 Groups and anonymous credentials 26

11 Calls 28

12 Key verification 29

13 Account, contacts, PIN and usernames 30

14 Secure backups 32

15 Trust model and limits 33

16 Good practices 35

17 Sources and further reading 36

INTRODUCTION

How to use this guide

Audience and prerequisites. You do not need to know cryptography or networking: the "Before you start" sections explain the concepts and the minimum maths from zero. After that the guide climbs gradually until it reaches the formulas.

State of the protocol. Signal today combines a post-quantum key agreement (PQXDH) with a classical ratchet (the Double Ratchet) and a post-quantum one (SPQR), which together form the Triple Ratchet. The rollout of SPQR is gradual and coexists with older clients, so details may change; when in doubt, the source of truth is the specifications published by the project (chapter 17).

How to read it. If you are starting from zero, begin with "Before you start" and consult the notation table just before chapter 1. Each chapter opens with a sentence that sums up the idea; key formulas carry a label, a plain-words explanation and a symbol-by-symbol breakdown, while secondary ones are reduced to an idea. Chapters 2 to 7 are the cryptographic core and should be read in order. Chapters 8 to 11 (messages, metadata, groups, calls) and 12 to 14 (verification, account, backups) can be read separately. Chapters 15 and 16 summarise the limits and the good practices.

BEFORE YOU START

The basic concepts

Five ideas are enough to understand Signal. If you already know them, skip to the notation table.

End-to-end encryption

When you send a message over the internet, it passes through several machines in between: your carrier, the app's servers, the recipient's carrier. In a service without end-to-end encryption, the company can read it along the way even if it protects it with encryption between your phone and its server. In an **end-to-end** service the message is encrypted on your device and decrypted only on the recipient's: the servers in the middle see only unreadable data.

The key question is how two people who have never met agree on the key they are going to use, if everything they say to each other can be overheard. That is the hard part, and most of this guide is devoted to it.

Keys and signatures

A **key** is a secret number used to encrypt or decrypt. There are two kinds. In **symmetric** encryption the same key locks and unlocks, like a house key; it is fast, but both parties must already share it. In **public-key** encryption each person has a **private key**, which never leaves their device, and a **public key** derived from it, which can be given to anyone. With public keys two people can build a shared secret without ever having met.

A **digital signature** is a seal that only the holder of the private key can apply and that anyone can check with the public one. It proves that a piece of data was published by the owner of that key and has not been modified.

Hashes and key derivation

A **hash** turns any data into a fixed-size fingerprint. It is deterministic (the same data gives the same fingerprint), impossible to reverse, and the smallest change produces a completely different fingerprint. A **key derivation function** (KDF) uses hashes to turn one secret into several independent keys: from a single secret you can get, for example, one key for encrypting and another for authenticating, and knowing one does not reveal the other. A **MAC** is a fingerprint with a key: it is used to check that a message was not altered and that it was created by someone who knew the key.

Forward secrecy and recovery

Forward secrecy means that stealing today's keys does not let anyone read yesterday's messages, because old keys were deleted. **Post-compromise security** is the symmetric property: if someone copies your current keys, the system heals itself and future messages are secure again as soon as you exchange fresh material. Signal was designed to have both.

Metadata

Metadata is data about the communication rather than its content: who talks to whom, when, from which IP address, how often. Encrypting the content does not automatically hide the metadata, and an important part of Signal's design consists of reducing it.

Quick glossary

TERM	WHAT IT IS
End-to-end encryption	The message is encrypted at the sender and decrypted only at the receiver
Private / public key	The private one is never shared; the public one is derived from it and is shared
Digital signature	A seal that proves who published a piece of data and that it did not change
Hash	A fixed-size fingerprint of some data, impossible to reverse
KDF	A function that derives several independent keys from one secret
MAC	A keyed fingerprint that proves a message was not altered
Forward secrecy	Stealing today's keys does not open yesterday's messages
Post-compromise security	After a key theft, future messages are secure again
Metadata	Data about the communication: who, when, from where
Quantum / post-quantum	A quantum computer could break part of today's cryptography; post-quantum is the kind that resists it

BEFORE YOU START

The minimum maths

All of Signal's cryptography rests on three tools: elliptic curves (to exchange keys and to sign), hash functions (to derive keys) and, more recently, a mechanism that resists quantum computers. Here is just enough to follow the rest.

Modular arithmetic

On a clock, after 12 comes 1: the numbers "wrap around". **Modular arithmetic** works like that with a limit number, the modulus. `a mod p` denotes the remainder of dividing `a` by `p`. Cryptography uses enormous moduli (around 255 bits), and in them there are operations that are easy to compute one way and infeasible to undo the other way.

Elliptic curves and Diffie-Hellman

Signal uses the curve **Curve25519**, defined over the numbers modulo the prime `2^255 - 19`. Its points form a set in which you can "add" a point to itself. There is a fixed, public starting point, the **base point** `G`. Adding `G` to itself `k` times gives another point, written `k·G`.

Computing `k·G` from `k` is fast; recovering `k` from `k·G` is, with today's computers, infeasible (the *discrete logarithm problem*). That gives the key pair: the **private key** is a 32-byte number `k` and the **public key** is the point `K = k·G`. The **X25519** variant uses only one coordinate of the point and is the one Signal uses to exchange keys.

The **Diffie-Hellman exchange** lets two people manufacture a shared secret in plain view of everyone. Alice has `a` and publishes `A = a·G`; Bob has `b` and publishes `B = b·G`. Then:

KEY FORMULA.

$$a \cdot B = a \cdot (b \cdot G) = (a \cdot b) \cdot G = b \cdot (a \cdot G) = b \cdot A$$

In words: each one mixes their secret with what the other made public, and both end up with exactly the same point; someone who only watches the public values cannot reach it.

What each piece is:

- `G`: the base point, public and fixed; it is the starting point that everybody knows.
- `a` and `b`: Alice's and Bob's secrets. They are numbers and never leave their device.
- `A = a·G` and `B = b·G`: the public keys. They are sent in plain view of everyone.
- `a·B` and `b·A`: what each one computes. Alice combines her secret with Bob's public key; Bob combines his with Alice's.
- `(a·b)·G`: the common result, which is the shared secret.

Why it works. Multiplication is commutative: `a·b` is the same as `b·a`. Alice computes `a·(b·G)` and Bob `b·(a·G)`, and both are `(a·b)·G`. An eavesdropper sees `A` and `B`, but to get to `a` or `b` they would have to solve the discrete logarithm, which is infeasible today.

An example with small numbers. The real curve cannot be computed by hand, but the same idea works with 23-hour clock arithmetic, where "multiplying a point by k" is replaced by "raising 5 to the power k":

1. Alice chooses the secret `a = 6` and publishes `5^6 mod 23 = 8`.
2. Bob chooses the secret `b = 15` and publishes `5^15 mod 23 = 19`.
3. Alice computes `19^6 mod 23 = 2`.
4. Bob computes `8^15 mod 23 = 2`.

Both obtain 2 without ever having sent 6 or 15. With 255-bit numbers, deducing 6 from 8 is infeasible.

In the rest of the guide, the whole operation "my secret with your public key" is written `DH(a, B)`. That point, passed through a KDF (key derivation function), is the shared key.

Signatures

An EdDSA-type signature proves that a private key `k` is known without revealing it. The signer picks a random number `r`, turns it into a point `R = r·G` and mixes that point, their public key `K` and the message into a challenge `h`. The signature is the pair `(R, s)`, where `s` combines the random number with the challenge and the private key in a way that does not give it away.

The verifier checks an equality in which only public values appear:

$$s\,G = R + h\,K$$

> **In words:** if the signature is genuine, both sides give the same point; only someone who knows `k` could have manufactured an `s` that satisfies it.

Signal needs a single identity key to serve both for exchanging keys (X25519) and for signing. For that it uses **XEdDSA**, a way of signing with keys from the Diffie-Hellman curve using the same scheme as above.

Key encapsulation (KEM)

A **KEM** is an alternative way of agreeing on a secret. It has three operations: generate a key pair, **encapsulate** with the public key (produces a secret `ss` and a ciphertext `ct`) and **decapsulate** with the private key (recovers `ss` from `ct`). Whoever holds the public key can create a secret and send `ct` in the open; only the owner of the private key recovers it. Post-quantum KEMs such as **ML-KEM** (formerly called Kyber) are based on lattices, mathematical problems different from curves, and that is why they are much bigger: on the order of a kilobyte per key and per ciphertext, compared with 32 bytes for X25519.

Why the quantum threat matters

A sufficiently large quantum computer could solve the discrete logarithm of elliptic curves with Shor's algorithm and break X25519 and signatures. It does not exist yet, but an adversary can **store today** the encrypted traffic to **decrypt it later** (*harvest now, decrypt later*). That is why Signal is adding post-quantum layers now. Symmetric encryption (AES-256) and hashes (SHA-256) are affected far less: Grover's algorithm only halves the effective security bits.

REFERENCE

Notation

This table collects the symbols used in the following chapters. Come back to it whenever a formula looks opaque.

SYMBOL	MEANING
`a`, `b`, `k`, `r`	Secret numbers (private keys, random values)
`A`, `B`, `K`, `R`	Curve points (public keys)
`G`	Curve25519 base point
`DH(x, Y)`	Diffie-Hellman: the point `x·Y`, converted to bytes
`IK_A`, `IK_B`	Identity key (long-lived) of Alice and of Bob
`SPK_B`	Bob's signed prekey (renewed from time to time)
`OPK_B`	Bob's one-time prekey
`EK_A`	Alice's ephemeral key, created for a single session
`PQPK_B`, `ss`, `ct`	Bob's KEM public key, shared secret and KEM ciphertext
`‖`	Byte concatenation
`KDF`, `HKDF`	Key derivation (HKDF with SHA-256 in Signal)
`SK`	Initial shared key of a session
`RK`, `CK`, `MK`	Root key, chain key and message key of the Double Ratchet
`HMAC`	Hash-based message authentication code

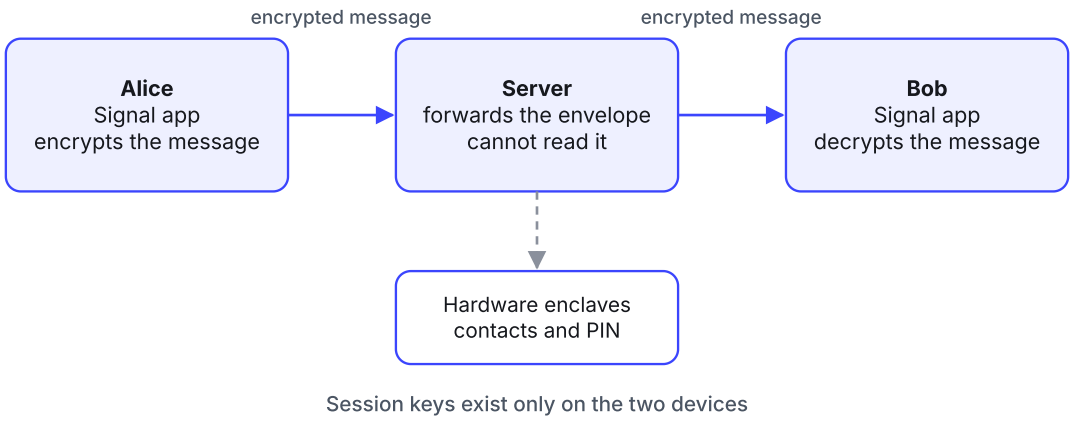
CHAPTER 1

Overview: what Signal protects and what it does not

Signal protects the content of your messages and calls with end-to-end encryption and strives to make its server know as little as possible; it does not hide from your carrier that you use Signal, and it does not protect a compromised device.

The parts of the system

Signal is an application (for Android, iOS and desktop) plus a set of services run by the Foundation. It is worth distinguishing each piece because each one sees different things.



The server carries the message, but the keys to read it exist only on the devices.

PART	WHAT IT DOES	WHAT IT SEES
Client (the app)	Generates the keys, encrypts and decrypts	Everything: it is the endpoint
Message server	Stores public keys and delivers encrypted messages	Recipient, IP address, time and size; not the content
Hardware enclaves	Contact discovery and PIN recovery	Hardware-protected data that not even the server's operator can read
Call servers	Relay encrypted audio and video	Who connects and when; not the content
Attachment storage	Stores files that are already encrypted	A block of bytes with no keys

What a message goes through

1. Your app fetches the recipient's public keys from the server and uses them to compute a shared secret (chapters 4 and 5).
2. It encrypts the message with a key that changes with every message (chapter 6).
3. It sends it to the server, which only sees who it is for (and, with *Sealed Sender*, does not see who it is from; chapter 9).
4. The server holds it until the recipient connects and delivers it. If there is a notification, the operating system signals that something new has arrived, but the text does not travel over that channel.
5. The recipient's device decrypts it and deletes the key used.

What the design guarantees

- **Confidentiality:** only the participants' devices can read the content.
- **Integrity and authenticity:** any alteration is detected and messages can only come from someone who knows the session key.
- **Forward secrecy:** the keys of old messages are deleted.
- **Post-compromise security:** if your keys are copied, the session heals itself as soon as there are new exchanges.
- **Cryptographic deniability:** messages are authenticated with shared-key codes rather than signatures, so a third party cannot prove from that data who wrote what.
- **Hybrid post-quantum resistance:** the key agreement and the ratchet add a post-quantum component alongside the classical one (chapters 5 and 7).

What it does not guarantee

- **A compromised device.** If someone controls your phone, they read the messages when you read them. No cryptography prevents this.
- **Anonymity.** Registering requires a phone number and the server sees your IP address when you connect (chapters 13 and 15).
- **Minimal metadata, not zero.** The server needs to know whom to deliver each message to. Chapters 9 and 10 explain what is hidden and what is not.
- **That your contact is who they say they are.** Signal provides tools to check this (chapter 12), but you have to use them.
- **That the other person will not copy the conversation.** A screenshot or a forward cannot be prevented.

CHAPTER 2

The cryptographic primitives

Signal does not invent algorithms: it combines standard, heavily studied pieces, and its contribution lies in how it assembles them.

FUNCTION	ALGORITHM	WHERE IT IS USED
Key exchange	X25519 (Curve25519)	Initial handshake and ratchet
Signatures	XEdDSA (over Curve25519)	Signing prekeys and certificates
Post-quantum agreement	ML-KEM / Kyber	PQXDH and SPQR
Key derivation	HKDF with SHA-256	Session and message keys
Authentication	HMAC-SHA256	Key chain and message tag
Message and attachment encryption	AES-256 in CBC mode + HMAC	Message and file contents
Field encryption	AES-256-GCM	Profile and stored data
Fingerprints	SHA-512	Safety numbers
PIN stretching	Argon2	Recovery PIN
Anonymous credentials	Ristretto255	Groups and usernames

Deriving keys with HKDF

HKDF turns an input secret into as many keys as needed. First it "extracts" it into a master key with an HMAC function and then "expands" it into a series of blocks, mixing into each one a label that says what the key will be used for.

The output is the concatenation of the generated blocks, trimmed to the requested length. Changing the label gives completely different keys from the same secret, and that makes it possible to separate uses.

Encrypting and authenticating a message

Each message uses a one-time message key `MK` (chapter 6). From it, HKDF derives three values: a 32-byte encryption key, a 32-byte authentication key and a 16-byte initial vector. The message is encrypted with AES-256 in CBC mode using the encryption key and the initial vector.

Over the ciphertext, both parties' public keys and the message header, an HMAC-SHA256 is computed with `k_mac`, trimmed (in Signal's implementation, to 8 bytes) and appended. The receiver verifies the tag before decrypting: "authenticate then decrypt" prevents attacks that alter the ciphertext. Since the message key is used only once, the short tag is sufficient.

CHAPTER 3

Identity keys and prekeys

Each Signal device publishes on the server a bundle of public keys (the *prekey bundle*) so that anyone can start an encrypted conversation with it even if it is switched off.

The problem

To encrypt the first message to Bob, Alice needs a secret shared with him, but Bob may be offline. The solution is for Bob to leave public material on the server in advance, from which Alice can derive that secret without asking him anything.

Which keys each device has

KEY	LIFETIME	ROLE
Identity key `IK`	Long (months or years)	Identifies the account (its devices share it); authenticates the key agreement
Signed prekey `SPK`	Medium; renewed periodically	Exchange material, signed with `IK`
One-time prekeys `OPK`	Once	Provide forward secrecy to the first message
Signed KEM prekey `PQSPK`	Medium	Post-quantum component, signed with `IK`
One-time KEM prekeys	Once	Same as the `OPK`, but post-quantum

All private keys stay on the device. Only the public ones are on the server, and each signed prekey carries the signature of Bob's identity key, which shows that it belongs to him.

How they are distributed

On registration, the device uploads its identity key, a signed prekey, a signed KEM prekey and a batch of one-time prekeys, and replenishes the batch when it runs out. When someone wants to talk to Bob, the server hands over the identity key, the signed prekeys and **one** one-time prekey of each type, and deletes it from its list so that it is not handed out twice.

If the batch runs out, the agreement still works without the one-time prekeys: it is slightly less robust against replays, but it does not break. For the post-quantum component there is a "last resort" KEM prekey that is never deleted, precisely so that there is always one available.

CHAPTER 4

X3DH: the initial handshake

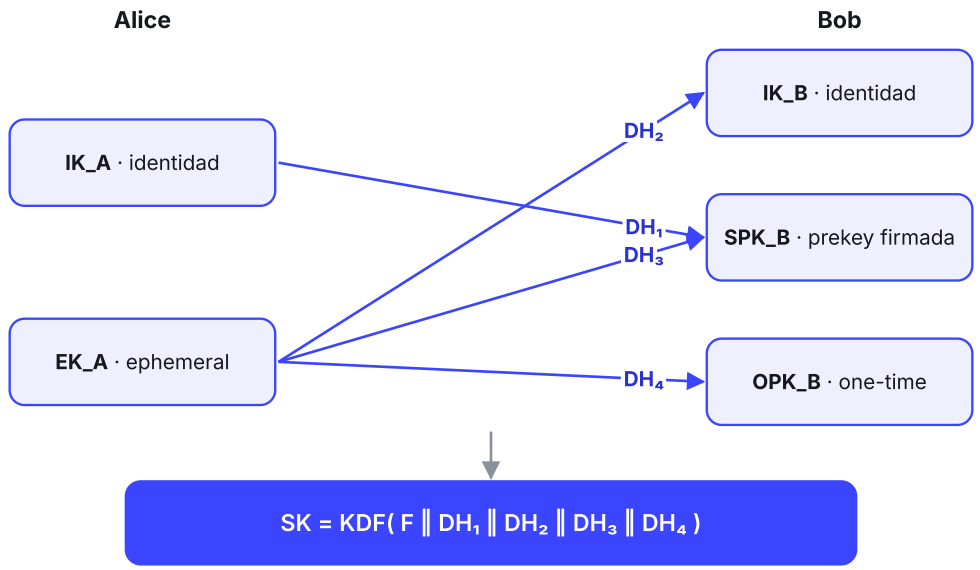
X3DH (*Extended Triple Diffie-Hellman*) combines three or four Diffie-Hellman exchanges so that Alice and Bob agree on an initial key even if Bob is offline.

The steps

1. Alice asks the server for Bob's bundle: `IK_B`, `SPK_B`, its signature and, if there is one, an `OPK_B`.
2. She verifies the signature of `SPK_B` with `IK_B`. If it does not match, she aborts.
3. She creates an ephemeral key `EK_A`, used only for this session.
4. She computes four Diffie-Hellman exchanges and joins them.
5. She derives the initial key `SK` with HKDF and deletes the private part of `EK_A` and the intermediate exchanges.
6. She sends Bob her first encrypted message, together with `IK_A`, `EK_A` and which prekeys she used.

The four exchanges

Alice mixes four different Diffie-Hellman exchanges into a single initial key. Each one provides a different guarantee; the table below summarises them.



Each line is a Diffie-Hellman between one of Alice's keys and one of Bob's; their results are joined into the initial key.

KEY FORMULA.

$$DH_1 = DH(IK_A,\ SPK_B)$$
$$DH_2 = DH(EK_A,\ IK_B)$$
$$DH_3 = DH(EK_A,\ SPK_B)$$
$$DH_4 = DH(EK_A,\ OPK_B)$$

In words: Alice performs four mixes of "a secret of mine with a public key of yours", combining her long-lived keys with Bob's and with an ephemeral key of her own.

What each piece is:

- `IK_A`, `IK_B`: identity keys of Alice and Bob (long-lived).
- `SPK_B`: Bob's signed prekey, which he left on the server.
- `OPK_B`: one of Bob's one-time prekeys, if any were left.
- `EK_A`: Alice's ephemeral key, created only for this session.
- `DH(x, Y)`: the Diffie-Hellman from the previous section: my secret `x` with your public `Y`.

$$SK = \mathrm{KDF}\left(F \,\|\, DH_1 \,\|\, DH_2 \,\|\, DH_3 \,\|\, DH_4\right)$$

In words: the four results are pasted together, a constant is added to separate this use from others, and everything goes through a key derivation function that returns the session's initial key.

What each piece is:

- `DH_1 … DH_4`: the four shared secrets above.
- `F`: a 32-byte constant with all bits set to one; it separates this use from others.
- `||`: concatenate, paste one value after another.
- `KDF`: the function that mixes everything and gives a uniform key of the requested length.
- `SK`: the initial shared key of the session, which will feed the Double Ratchet.

Why it works. Each `DH_i` is a Diffie-Hellman, so Bob can compute the same value with his private keys (for example, `DH(IK_A, SPK_B)` is obtained by him as `DH(spk_B, IK_A)`) and arrives at the same `SK`. None of the four secrets travels over the network, and an attacker would need private keys from both sides to reconstruct them.

If no `OPK_B` was left, `DH_4` is omitted.

What each exchange provides

EXCHANGE	WHAT IT PROVIDES
`DH_1`	Authenticates Alice: only the holder of the private key of `IK_A` can compute it
`DH_2`	Authenticates Bob: only the holder of the private key of `IK_B` can compute it
`DH_3`	Forward secrecy: mixes an ephemeral key of Alice with Bob's prekey
`DH_4`	Adds a value used only once and prevents replays of the first message

Associated data and deniability

The two identity keys form some *associated data* `AD = IK_A || IK_B` that is incorporated into the messages' authentication tag, so that a message is tied to this particular pair. Since authentication is done with Diffie-Hellman (something either of the two can compute) rather than with a signature over the message, neither of them can prove to a third party that the other wrote something: this is *cryptographic deniability*.

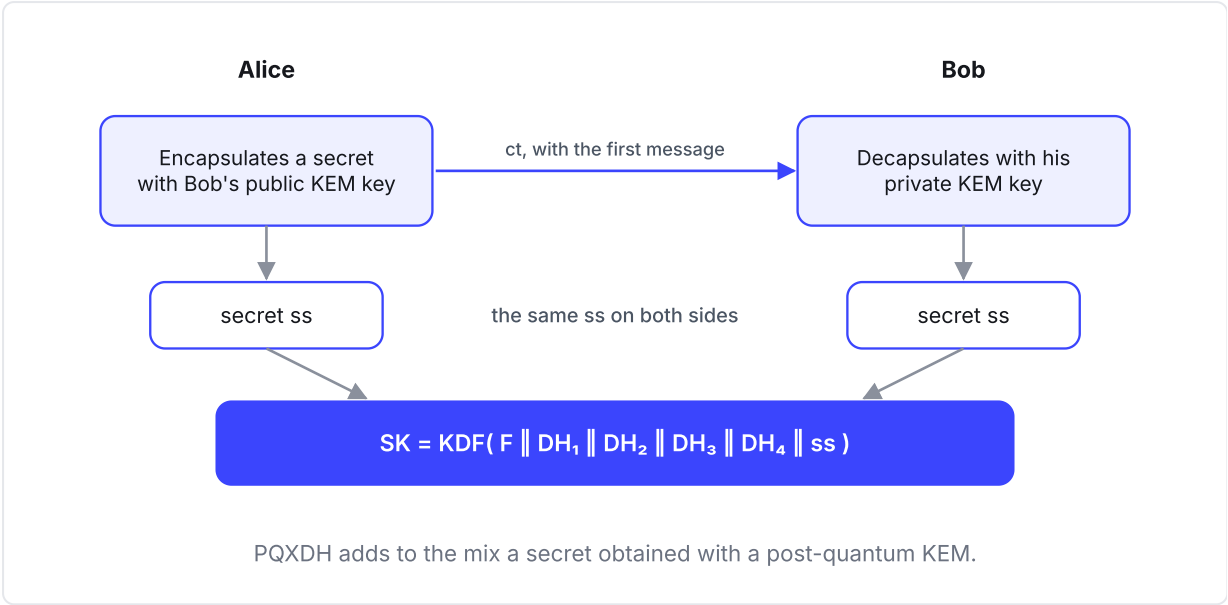
CHAPTER 5

PQXDH: the post-quantum handshake

PQXDH adds a post-quantum key exchange to X3DH so that an adversary who stores traffic today cannot decrypt it once a quantum computer exists.

What changes compared with X3DH

Signal introduced PQXDH in 2023 as a first step towards quantum resistance. In addition to the classical prekeys, Bob publishes a signed KEM prekey (and optionally one-time ones). Alice uses it to **encapsulate** a secret `ss` that only Bob can recover, and sends the ciphertext `ct` along with the first message. Then she adds that secret to the mix:



PQXDH adds to the mix a secret obtained with a post-quantum KEM.

$$SK = \mathrm{KDF}\big(F \,\|\, DH_1 \,\|\, DH_2 \,\|\, DH_3 \,\|\, DH_4 \,\|\, ss\big)$$

In words: it is the same mix as X3DH with one more ingredient, the KEM secret; to recover `SK` you must know both the classical secrets and the post-quantum one.

Bob recovers `ss = Decaps(pq_B, ct)` with his private key and obtains the same `SK`. The KEM that the protocol specifies is CRYSTALS-Kyber in its highest-level variant (1024), the one NIST later standardised under the name ML-KEM.

Why it is hybrid

`SK` depends at once on the value from the curves and the value from the KEM. To recover it you have to break **both** things: if someone discovers a flaw in Kyber, the four curve exchanges still protect; if a quantum computer breaks the curves, `ss` still protects. It is a prudent bet because post-quantum schemes are much younger than curves.

What it protects and what it does not

- **It protects the confidentiality of the initial secret** against someone who records traffic today and has a quantum computer in the future.
- **It does not protect authentication against an active quantum attacker.** Identity keys and signatures are still elliptic-curve; an adversary who already had such a computer could impersonate someone in a new session. The risk PQXDH covers is "store now, decrypt later".
- **It does not cover the whole session.** PQXDH protects only the initial secret; the later Double Ratchet exchanges were still classical. Chapter 7 closes that gap.

CHAPTER 6

The Double Ratchet

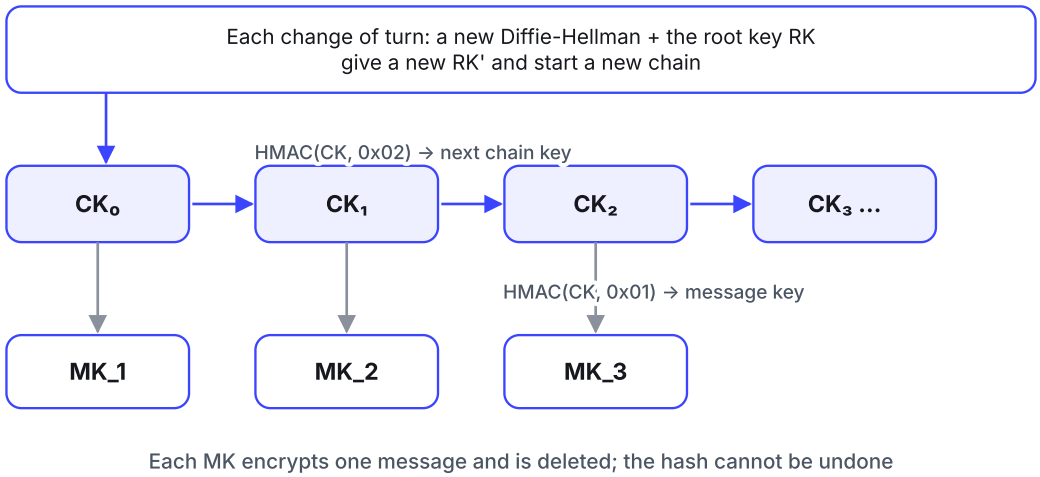
The Double Ratchet renews the encryption key with every message and with every change of turn, so that stealing one key gives access to only a small piece of the conversation.

The idea

A ratchet is a mechanism that turns in only one direction. Here there are two. The **symmetric ratchet** advances a key with a hash function after every message; since the hash cannot be reversed, whoever steals the current key cannot calculate the previous ones (forward secrecy). The **Diffie-Hellman ratchet** mixes in new material every time the conversation changes direction; since the new material is secret to the attacker, the attacker is locked out even if they copied the earlier keys (post-compromise security).

The three chains

Each party maintains a **root key** `RK` (which comes from the initial handshake, chapters 4 and 5) and two key chains: one for sending and one for receiving, each with its own **chain key** `CK`. Every time the turn changes, a step of the Diffie-Hellman ratchet is taken: whoever is about to speak generates a new key pair, performs a Diffie-Hellman with the other party's latest public key and uses it to update the root key and start a new chain:



A chain that only moves forward: from each chain key come a message key and the next chain key.

KEY FORMULA.

$$(RK',\ CK) = \mathrm{KDF}_{RK}\big(RK,\ DH(dh_s,\ DH_r)\big)$$

In words: when the turn changes, the party about to speak creates a new key pair, mixes it with the other party's most recent public key and, with that new secret and the previous root key, obtains a new root key and a new chain.

What each piece is:

- `RK`: the current root key, which comes from the initial handshake.
- `dh_s`: the sender's new Diffie-Hellman private key.
- `DH_r`: the last Diffie-Hellman public key received from the other party.
- `DH(dh_s, DH_r)`: the fresh secret that neither had before.
- `KDF_RK`: the function that mixes the old root with the fresh secret.
- `RK'` and `CK`: the new root key and the key of the first new chain.

Why it works. The fresh secret depends on a new private key that an attacker who copied the previous state does not have, so the new root is out of their reach: that is post-compromise security.

Within a chain, each message consumes a chain key and produces a one-time message key and the next chain key:

KEY FORMULA.

$$MK = \mathrm{HMAC}(CK,\ 0x01), \qquad CK' = \mathrm{HMAC}(CK,\ 0x02)$$

In words: from the current chain key come, separately, the key for a single message and the next chain key; the previous one is deleted.

What each piece is:

- `CK`: the chain key of this step.
- `HMAC(CK, 0x01)`: a keyed fingerprint of `CK` and the constant 1; it gives the message key `MK`.
- `HMAC(CK, 0x02)`: the same with the constant 2; it gives the next chain key `CK'`.

An example. It is a tape that only moves forward: from `CK_0` come `MK_1` and `CK_1`; from `CK_1` come `MK_2` and `CK_2`; from `CK_2` come `MK_3` and `CK_3`. Each `MK_n` encrypts one message and is deleted.

Why it works. HMAC cannot be reversed: with `CK_2` in hand there is no way to calculate `CK_1` or `MK_2`. Whoever steals the current key reads only the following messages, not the earlier ones.

After encrypting or decrypting, `MK` and the previous `CK` are deleted. `MK` goes to HKDF to obtain the encryption key, the authentication key and the initial vector (chapter 2).

The header of each message

Each message travels with a header that includes the sender's current Diffie-Hellman public key, the message number within its chain and the length of the previous chain. With that the receiver knows whether it must take a Diffie-Hellman ratchet step and how many messages were skipped.

An example

TURN	WHAT HAPPENS	NEW KEY
1. Alice sends 3 messages	Symmetric step after each one	`MK_1`, `MK_2`, `MK_3` of her chain
2. Bob replies	Bob generates a new pair and takes a Diffie-Hellman step	New chain for Bob
3. Alice answers	Alice generates a new pair and takes a step	New chain for Alice

If an attacker copies Alice's state after turn 1, they can read the later messages in that chain; but as soon as Bob replies with a key the attacker does not have, the root changes and the attacker is locked out.

Out-of-order messages

The network can deliver messages out of order or lose some. If message 5 arrives before message 4, the receiver advances the chain to 5 and stores the message key for 4 in case it arrives later. So that this is not a security leak, the number of stored keys is limited and they are deleted over time.

CHAPTER 7

SPQR and the Triple Ratchet

The Sparse Post-Quantum Ratchet (SPQR) adds to the conversation a third, post-quantum ratchet that runs in parallel with the Double Ratchet and completes what PQXDH left half done.

The problem

PQXDH protects the initial secret, but from then on the Diffie-Hellman ratchet uses elliptic curves. A quantum adversary who had stored the traffic could break those steps later. The natural solution would be to replace the curve steps with a post-quantum KEM, but ML-KEM has keys and ciphertexts of around a kilobyte each (in the 768 variant, 1184 and 1088 bytes), whereas an X25519 key takes 32 bytes. Putting that in every message would inflate the traffic and the cost of every conversation.

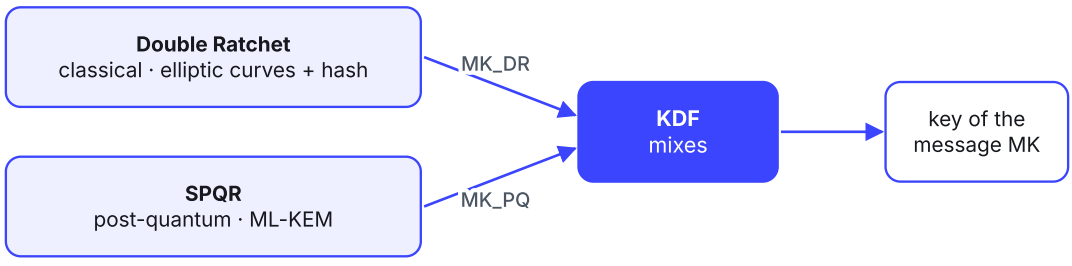
The solution: a sparse ratchet

Signal published SPQR in October 2025, developed together with academic researchers and with the support of cryptography firms for its formal analysis. Its main ideas are:

- **A post-quantum KEM (ML-KEM 768)** instead of Diffie-Hellman to advance the ratchet.
- **Erasure codes:** the public key and the ciphertext, which are large, are chopped up and spread across many ordinary messages, and the receiver can reconstruct them even if some pieces are missing. That way each message carries only a small fraction.
- **Advancing by epochs:** the ratchet does not advance with every message but from time to time, when an exchange has been completed. This is what makes it "sparse".

The Triple Ratchet

SPQR does not replace the Double Ratchet: it runs beside it. Each message is encrypted with a key that mixes the two outputs through a KDF:



To recover the key both ratchets must be broken at once

The post-quantum ratchet runs in parallel with the classical one and their outputs are mixed.

$$MK = \mathrm{KDF}\left(MK_{DR} \,\|\, MK_{PQ}\right)$$

In words: each message's key mixes the one that comes out of the classical ratchet with the one that comes out of the post-quantum one.

As with PQXDH, the mix is **hybrid**: you must break both the classical and the post-quantum ratchet at the same time to recover the key. Keeping the proven Double Ratchet preserves the guarantees already known and adds the new ones.

Rollout and compatibility

So as not to interrupt the service, sessions can temporarily downgrade to the Double Ratchet when the other end does not yet support SPQR. Signal has announced that, once all clients support it, it will archive the sessions that do not use it so that every conversation has post-quantum protection. Until then the protection depends on both ends being up to date. Its real strength also depends on a correct implementation and on ML-KEM continuing to withstand analysis, hence the hybrid design.

CHAPTER 8

Messages, attachments and several devices

The Double Ratchet encrypts individual messages; this chapter explains how they are packaged, what happens with files and how one account works across several devices.

The message envelope

Each encrypted message carries a header and a body. The first message of a session also includes the handshake data (Alice's identity and ephemeral keys, the prekeys used and, since PQXDH, the KEM ciphertext); the following ones carry only the ratchet header. Before encryption, the content is padded to fixed sizes to hide the exact length of the text.

Attachments

Large files do not travel inside the message. The app generates a random 64-byte key (32 for encrypting and 32 for authenticating), encrypts the file with AES-256-CBC and computes an HMAC-SHA256 for it, and uploads the encrypted result to external storage. The download link, the key and the file's fingerprint travel **inside** the end-to-end encrypted message. The storage server sees only a block of bytes with no key.

One account, several devices

An account can have a primary phone and several linked devices (desktop, tablet). Each device has its own device identity and an independent session with each of your contacts' devices. When you send a message, your app encrypts it **once for each device** of the recipient and of your own devices, and sends all the copies; the handling of these multiple sessions follows a protocol called *Sesame*.

To link a device, a QR code is scanned on the primary phone. That code contains an ephemeral key from the new app; the phone uses it to hand over, encrypted, the account material (profile and identity keys). **Important consequence:** a linked device is part of your account and can read your messages; scanning a linking QR that you did not generate yourself amounts to giving your account to someone else.

Disappearing messages and receipts

Disappearing messages carry a timer inside the encrypted message itself, and each app deletes them when it expires. It is an agreement between apps and not a technical guarantee: a contact with a modified client or a screenshot can keep them. Delivery and read receipts are also encrypted messages and can be turned off.

What the server stores

While the recipient is offline, the server stores the encrypted message in a queue and deletes it when it is delivered. It does not keep the conversation history: that lives on the devices and, if you turn it on, in an encrypted backup that the server cannot read (chapter 14).

CHAPTER 9

Sealed Sender and metadata

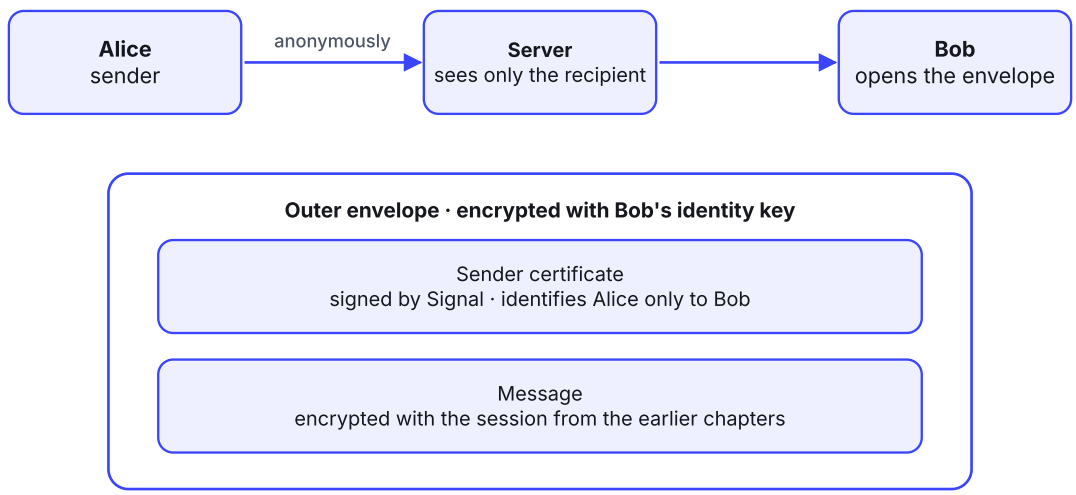
Sealed Sender hides from the server who sends each message: the server knows whom it is delivering to, but not whom it comes from.

The problem

In an ordinary service, every message you upload comes with your credentials so that the server knows it is you and can limit abuse. That means the server knows the sender and the recipient of every message, which is precisely a social graph. Encrypting the content does not prevent it.

The solution

The sender keeps their identity **inside** the encrypted envelope, so that only the recipient can read it:



The sender's identity travels inside the envelope: the server does not see it, the recipient does.

1. The sender obtains from the server a **sender certificate**, signed by the service itself, short-lived, which attests that their identity key belongs to their account.
2. They encrypt the message with the session from the previous chapters and put the result, together with the certificate, in a second envelope encrypted with the recipient's identity key and an ephemeral key.

3. They upload that envelope to the server **without authenticating as the sender**. Instead they present proof that they are allowed to write to the recipient, an access key derived from the recipient's profile key.
4. The server checks that access key, does not know who the sender is and delivers the envelope.
5. The recipient opens the outer envelope, reads the certificate, verifies the signature and then decrypts the message with the session.

The access key is derived from the recipient's profile key, which normally only their contacts know. Someone who does not have it cannot send sealed messages; the recipient can allow sealed messages "from anyone" with a setting, at the cost of accepting more possible spam.

What the server still knows

Sealed Sender improves the situation but does not fully resolve it. The following table summarises what the service sees.

DATA	DOES THE SERVER SEE IT?
Content of messages and calls	No
Sender of a sealed message	No (the recipient does)
Recipient and delivery time	Yes
Approximate message size	Yes
Your IP address when you connect	Yes
Your contacts	Not in the clear (private discovery, chapter 13)
Members and title of your groups	Not in the clear (chapter 10)
Profile name and photo	Only encrypted with the profile key
Your phone number	Yes (it is used to register)
Account creation date and last connection	Yes

The last row has practical value: in the court subpoenas that Signal has made public, those were the only user data it was able to hand over, along with the phone number.

Correlation attacks

A server that sees who delivers and who receives at the same time, together with the IPs and the times, can try to pair senders and recipients by simple temporal coincidence: if Alice connects and sends an envelope just before Bob receives one and replies, the correlation is strong even

though the envelope is sealed. Academic work has shown that with enough observations conversations can be inferred. Sealed Sender raises the cost of that surveillance, but you have to trust that the operator does not do it; hiding the IP also requires a VPN or a network such as Tor.

CHAPTER 10

Groups and anonymous credentials

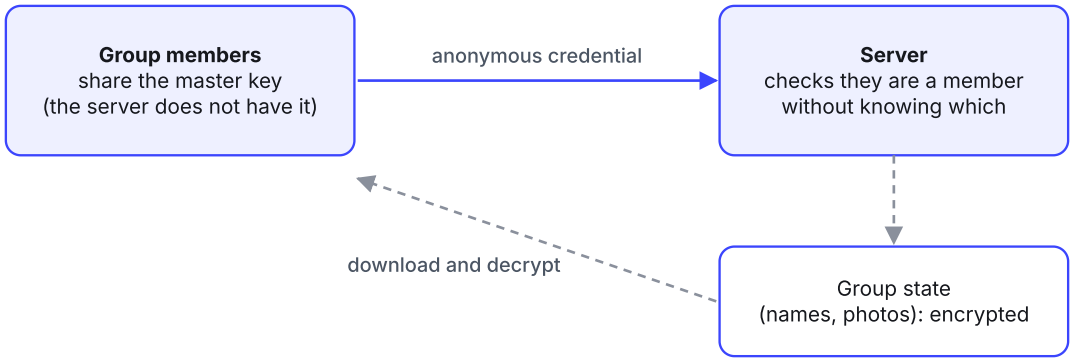
Signal organises groups so that the server stores their state encrypted and checks who may act on it without knowing who they are.

The problem

A group has a name, a photo and a list of members, and the server has to decide who can read or modify them. If the server saw that list, it would know the relationships between people. The goal is that it does not see it and that, even so, it can deny access to anyone who does not belong to the group.

Master key and encrypted state

When a group is created, the app generates a **group master key** of 32 random bytes, which it distributes to the members through ordinary end-to-end encrypted messages. From it are derived the keys that encrypt the title, the photo, the description and the member list, and a public identifier for the group. The server stores that state encrypted and versioned, and members are identified in it through a deterministic encryption of their identifier: the server can see that two entries are different, but not to whom each one corresponds.



The server admits members without knowing who each one is and stores a state it cannot read.

Anonymous credentials

Signal needs the server to accept only members without them identifying themselves. It achieves this with **keyed-verification anonymous credentials**, the construction called *zkgroup*, the work of Chase, Perrin and Zaverucha and based on the Ristretto255 group:

1. Each day the server issues each account an authentication credential that signs its identifier and the date.
2. To act on a group, the member presents a **zero-knowledge proof**: it shows that they hold a valid credential for an identifier that appears (encrypted) in the group's list, without revealing which one.
3. The server verifies the proof and authorises the action, but does not learn which member performed it.

So that new members can see the others' profiles, a similar scheme exists with **profile-key credentials**. This way the server can enforce administrator or write permissions without seeing the list.

How group messages are sent

Encrypting a message separately for each member is expensive in large groups. Signal uses **sender keys**: each member creates a key chain of their own and hands it to the others over their individual sessions. After that, each message is encrypted only once with the sender's chain and distributed to everyone. To send those messages in sealed form without revealing group membership, Signal has added anonymous membership proofs called *group send endorsements*.

What remains exposed

- Group members see the list and, depending on settings, the others' phone numbers; usernames allow inviting or joining without sharing the number.
- The server knows which accounts connect, from which IP and at what times, and how many envelopes it delivers, but not which group they belong to.
- A member with a modified client can copy the messages: in a group everyone is an endpoint.

CHAPTER 11

Calls

Signal's voice and video calls use WebRTC technology with the keys agreed inside end-to-end encrypted messages, and they hide your IP if you ask them to.

One-to-one calls

To start a call, the two apps exchange signalling messages (the offer, the answer and the possible network addresses) and they do so as ordinary Signal messages, that is, encrypted with the session from the previous chapters. Inside that signalling they agree on the keys that encrypt the audio and video (SRTP). The server that helps to connect never receives those keys.

The call can go **directly** between the two devices or **through a relay server** (TURN) run by Signal, which forwards packets that are already encrypted. In a direct call, each party sees the other's IP address. There is a setting, "Always relay calls", that forces the call through the server: in exchange for some latency, your contact does not see your IP, and Signal does not see the content either. Calls from someone who is not in your contacts are relayed by default.

Group calls

In a call with several people, each one sends their audio and video to a server, called an **SFU**, which forwards it to the others without decrypting it. End-to-end encryption is maintained like this:

1. A participant creates a media key and distributes it to the others through individual messages encrypted with their sessions.
2. Each audio and video packet is encrypted with that key before leaving the device, so the server forwards opaque bytes.

3. When someone joins or leaves, a new key is generated and distributed, so that whoever left cannot keep reading and whoever joins cannot read what came before.

In call links, the key travels in the part of the link that the browser does not send to the server.

What the server sees

The call server sees the IP addresses of those who connect, when they join and leave and how much traffic they generate; it does not see the content. If your IP worries you, turn on relaying and, for more protection, use a VPN or Tor.

CHAPTER 12

Key verification

All end-to-end encryption depends on the public key you receive really being your contact's; safety numbers and automatic verification exist to check this.

The problem of initial trust

Signal distributes public keys through its server. If that server (or someone who controls it) handed Alice a key that is not Bob's but an attacker's, Alice would encrypt for the attacker, who could forward the message to Bob and read it in the middle. This is a *man in the middle* attack. Encryption does not detect it on its own: you have to compare the keys over an independent channel.

Safety numbers

Each conversation has a 60-digit **safety number** that depends on the two identity keys. Each party contributes a 30-digit fingerprint, calculated by applying SHA-512 thousands of times (more than five thousand iterations, to make brute force more expensive) to their identity key and their identifier. The two fingerprints are sorted and concatenated so that both parties see the same number.

If Alice and Bob see the same number (in person, over a call or by scanning the other's QR code), they know that each has the other's genuine key: an attacker in the middle would have produced a different number. When a contact's identity key changes, Signal warns that "your

safety number has changed". It is usually harmless (they reinstalled the app, changed phones) but it is also what you would see under an attack.

Automatic key verification

Hardly anyone compares numbers by hand. That is why Signal added in 2026 **automatic key verification**, based on a transparency log:

- Signal records changes of phone number and username in an **append-only log** (a hash tree). The identifiers are hidden with verifiable pseudorandom functions so that auditors do not see user data.
- **Independent auditors** (Cloudflare and Trail of Bits) check that the log is consistent and that everyone sees the same thing. Your app also periodically checks your own identifiers.
- On the conversation screen, "View safety number" offers "Verify automatically"; if everything matches, "Verified encryption" appears.

It has limits: it needs the contact's phone number (it does not work if you only know them by username), it fails if that contact changed number until you confirm it by another route, and it proves that *everyone sees the same key for that identifier*, not *who controls* the account: a stolen account is not detected. It can be turned off under Privacy > Advanced.

METHOD	WHAT IT PROVES	WHAT IT REQUIRES
Safety number	The key you see is the one your contact has	Comparing the number or the QR over another channel
Automatic verification	Everyone sees the same key for that number	Having the contact's phone number

CHAPTER 13

Account, contacts, PIN and usernames

This chapter covers how an account is created, how Signal finds your contacts without knowing them, how a PIN is recovered without the server guessing it, and how to hide your number with a username.

Registration and registration lock

To create an account, Signal sends a code by SMS or call to the number given; whoever receives it proves they control that number. This step exists mainly against spam: numbers cost money and are scarce. Its consequence is that Signal has always been tied to your number, and that whoever hijacks your number at the carrier could register it on another phone. **Registration lock** additionally requires your PIN to register the number on a new device, and closes off that route.

Private contact discovery

To tell you which of your contacts use Signal without the server seeing your address book, your app sends the numbers encrypted to a **secure hardware enclave** (Intel SGX) of the service. The app checks through remote attestation that the enclave is running the published code, and the enclave looks up its list in such a way that not even the server's operator can see which numbers were queried or which ones match.

PIN and secure value recovery (SVR)

A PIN of 4 to 6 digits is easy to guess. SVR makes it possible to use it anyway without letting anyone brute-force it:

1. The client passes the PIN through **Argon2**, a slow function that makes each attempt more expensive, and obtains a 32-byte value from which it derives an authentication token `auth` and a secret `c1`.
2. It also generates 256 random bits `c2`, which together with `c1` form the master key from which come the keys that encrypt your account data in the cloud.
3. `c2` is stored in a server enclave, associated with `auth`. The enclave hands it over only for the correct `auth` and **counts failed attempts**: after a few (five in Signal's example) it stops responding.
4. The state lives only in encrypted memory and is replicated between enclaves with the Raft consensus protocol, so that nobody can reset the counter by resetting the storage.

Since `c2` is random, the master key cannot be attacked from the PIN alone, and since the enclave limits attempts, an attacker with access to the server does not have unlimited attempts either. Later versions (SVR3, described in a 2024 academic paper) split the secret across several protected-hardware environments, so that compromising just one is not enough. Choosing a long PIN, with letters and numbers, strengthens the whole system.

Usernames and number privacy

Since 2024 you can create a **username** (`name.42` : a nickname plus a mandatory number that prevents anyone from hoarding names) and share it instead of your phone number. The two settings that accompany it are who sees my number? and who can find me by my number?; on new accounts the number is no longer shown by default. Usernames are not stored in the clear

on the server: they are stored as a hash and resolved with cryptographic proofs, and the username's link or QR code contains a secret that prevents enumerating them. You can change or delete it at any time to break the links that have been shared.

Important nuance. A username does not make the account anonymous: Signal still knows the number and, faced with a court order about a username, could hand over that number together with the creation date and the last connection.

Accounts without a number: in development

Dependence on the number is the most repeated criticism. The code for registering without a phone number already appears in Signal's codebase, but it is not yet usable and the schedule is provisional. The difficulty is spam: one possibility that has been mentioned is charging a small fee per numberless account so that creating thousands becomes expensive. Until it exists, the tie to a number remains a real limit on Signal's privacy.

CHAPTER 14

Secure backups

Signal's secure backups store your history in the cloud in such a way that only your recovery key can open them.

The problem

For years Signal did not offer cloud backups because they are a risk: what is stored on a server can leak or be demanded by an authority. In return, losing or breaking the phone meant losing the history. Secure backups, which are enabled voluntarily, try to avoid that dilemma.

How they work

1. When you enable them, the device generates a **64-character recovery key**. That key is never shared with Signal.
2. From it, a KDF derives the keys that encrypt the backup and the identifier under which it is stored.
3. Every day the device creates a new file with your messages, encrypts it and uploads it, and it replaces the previous day's.

4. To restore on a new phone you enter the recovery key, which lets you download and decrypt the backup.

The server sees only an encrypted block of bytes whose size and update date it knows; neither Signal nor anyone without the key can read it. The flip side is harsh: **if you lose the recovery key, nobody can recover the backup**, not even Signal.

Plans

Backups arrived first on Android and then on iOS. There is a free tier, which covers text and a limited amount of recent files (around 100 MiB), and a paid one of up to 100 GB for about two dollars a month, which also helps fund the Foundation. Signal has said that the same technology will serve as the basis for more options, among them transferring history between Android, iPhone and desktop.

What risks they add

A backup is always a second attack surface: whoever obtains your recovery key and the file can read your past history. That is why it is advisable to keep it outside the phone (in a password manager or on paper) and to weigh whether the convenience of being able to restore outweighs the risk. If your threat model is high, you can leave backups off and use disappearing messages.

CHAPTER 15

Trust model and limits

Signal greatly reduces what you have to believe, but not to zero: this chapter lists whom and what you still trust when using it.

Who is behind it

Signal is maintained by the **Signal Foundation**, a non-profit organisation founded in 2018 with an initial contribution of 50 million dollars from Brian Acton, co-founder of WhatsApp. It is funded by donations and, now, by the paid tier of the backups. Moxie Marlinspike, creator of the protocol together with Trevor Perrin, stepped down from leadership in 2022; it is presided over by Meredith Whittaker. The code of the apps and of the server is open, and the protocols are documented in detail.

Whom you trust with what

PARTY	WHAT IT CAN DO IF IT FAILS OR IS HOSTILE
Your device	Read everything: it is one of the endpoints
Your contact's devices	Read, copy or leak what you send them
Operating system and manufacturer	Touch the screen, the keyboard and the apps' memory
Signal's server	Hand over a false key (detected by verifying), see metadata, deny service
Hardware enclaves	A manufacturer flaw could weaken contact discovery and the PIN
Your carrier	See that you use Signal and hijack the number (registration lock slows it)

What you can check and what you cannot

You can read the protocol and its code, and the protocol has been analysed in academic publications and partly formally verified. What you cannot check on your own is that the server you use runs exactly the published code; for the enclaves, remote attestation covers a good part of that gap, but for the rest it is an act of trust. Key verification (chapter 12) is what reduces trust in the server.

The limits, one by one

- **The phone number.** It remains the account identifier, even though usernames hide it from other users. Numberless accounts are in development.
- **A central server.** Signal is deliberately not federated (it allows the protocol to be updated quickly), but this implies a single point of failure and of legal pressure, and that availability depends on a single operator.
- **The IP address and correlation.** The server sees where you connect from; Sealed Sender does not prevent it (chapter 9).
- **Compromised endpoints.** Spyware, an unlocked phone in someone else's hands, a notification preview on a locked screen, a cloud backup of the system that includes the app's data. End-to-end encryption covers none of this.
- **Linked devices.** Each one widens the attack surface, and a deceptive QR linking gives access to the account.
- **Push notifications.** They pass through Apple's and Google's services. They do not include the text, but they reveal the moment when something arrives.

- **The legal context.** The service is operated from the United States. Signal has stated publicly that it would rather stop operating in a country than weaken encryption, which shows that laws requiring messages to be scanned are a real risk for the service, not a hypothesis.

CHAPTER 16

Good practices

Most of the privacy gains with Signal come from a few settings and habits, ordered here from highest to lowest impact.

1. **Keep the app and the operating system up to date.** Post-quantum protection and security fixes arrive by update; always install from the official source.
2. **Set a long PIN and turn on registration lock.** With letters and numbers if possible. It protects your account against theft of the number (chapter 13).
3. **Lock the screen and the app,** and disable content in notifications.
4. **Create a username and hide your number** in the privacy settings, and limit who can find you by number.
5. **Verify the keys of important contacts,** with a safety number or with automatic verification.
6. **Use disappearing messages** in sensitive conversations, knowing it is an agreement between apps.
7. **Review your linked devices** from time to time and never scan a linking QR that you did not generate yourself.
8. **If your IP worries you,** turn on "Always relay calls" and use a VPN or Tor on the network you connect from.
9. **Decide calmly about backups.** If you enable them, keep the recovery key outside the phone.
10. **Distrust anyone who asks for your registration code or your PIN.** Signal never asks for them.

What to combine depending on the adversary

YOU WANT TO PROTECT YOURSELF FROM...	WHAT TO DO
Someone intercepting the network	Nothing special: encryption already covers it
Someone taking your number	Long PIN and registration lock
Your number being shared with strangers	Username and number settings
A server that swaps keys	Verify safety numbers
Your location being known	VPN or Tor, call relaying
A stolen or unlocked phone	Screen lock, disappearing messages, no previews
Targeted spyware	No Signal setting is enough: update, minimise what you store and seek specialist help

CHAPTER 17

Sources and further reading

The guide relies on Signal's official specifications and articles and on the publications cited below. For details that change over time (SPQR rollout, numberless accounts, prices), always consult the original source.

Signal specifications and articles

- Signal documentation and specifications: X3DH, Double Ratchet and Sesame.
- The PQXDH specification
- Signal Protocol and Post-Quantum Ratchets (SPQR)
- Technology preview for secure value recovery
- Introducing Automatic Key Verification

Academic and technical articles

- Secret Key Recovery in a Global-Scale End-to-End Encryption System (SVR3)

- PQShield: Diving into Signal's New Post-Quantum Protocol
- Migrating Protocols to the Post-Quantum Setting: The Case of Signal's Double Ratchet

Specialist press

- TechCrunch: Signal now lets you keep your phone number private
- The Intercept: usernames and number privacy
- BleepingComputer: secure backups
- Daring Fireball: secure backups on iOS
- CyberInsider: Triple Ratchet
- Privacy Guides: numberless accounts in the Android beta