Concept · May 18, 2026

End-to-end encryption, truly explained

What providers say when they say E2EE, and what they don't say. A didactic explanation of the mechanism and its limits, without the marketing wrapper.

Let's be clear: WhatsApp says your messages are end-to-end encrypted. It's true — and it's not enough. If the backup goes to iCloud or Google Drive without additional encryption, the encryption is broken on your own phone. The operational question is not whether it is encrypted, but where the keys reside.

What encryption really means

Encrypting a message means transforming it into something that looks like noise to anyone who does not possess certain information called a key. The operation is done on the device of the sender and, with the correct key, is undone on the device of the receiver. In between, the message travels as a succession of bytes with no apparent meaning. That is the simple idea. The rest of the article deals with the nuances that convert it, depending on the case, into a real guarantee or into a marketing label.

The adjective *end-to-end* — in English *end-to-end*, abbreviated E2EE — adds a precision. Encryption is not done so that an intermediate server can read it and deliver it. It is done so that only the two ends — the device of the sender and the device of the receiver — possess the key. Any server through which the message passes sees the noise, not the message. That is the technical difference with encryption *in transit*, where the content travels encrypted from one server to the next, but each server it passes through decrypts it to forward it, temporarily recovering the text in clear.

The paradox of the shared secret

There is an obvious problem. For two people to be able to encrypt and decrypt messages between themselves, both need the same key. But how do they agree on this key if everything they send each other, by definition, passes through a channel where someone could be listening? Agreeing on the key in the same channel where they will later use it seems impossible: if the attacker hears it when agreeing on it, they will be able to decrypt everything subsequent. For decades, classical cryptography solved this the hard way: keys were delivered in person, before starting to use them, in physical encounters. Ambassadors carried briefcases of keys sewn into the lining of their coat.

In contemporary email, that solution does not scale. If we had to go physically to each person's house with whom we intended to communicate encrypted, we would not get to talk to anyone. The question posed fifty years ago by the cryptographic community was this: is it possible for two people who do not know each other and who only share a public channel to agree, in that same public channel, on a secret that no one listening to the channel can know?

The elegance of Diffie-Hellman

In 1976, two mathematicians named Whitfield Diffie and Martin Hellman demonstrated something apparently impossible: that two people, talking only through a public channel — a channel where anyone can hear everything they say — can agree on a secret password without any listener being able to discover it. It sounds like magic. It is not: it is mathematics. The Diffie-Hellman key exchange, as it has been known since then, is the basis of practically all encrypted communication on the internet, and half a century of intensive use and global academic scrutiny confirm its solidity. Anyone who wants to see the visual intuition or mathematics can continue reading. Anyone who prefers to trust that it works can also continue without losing the thread of the article.

For anyone who wants to intuit it in an image, there is a known analogy with colors. Imagine that Alice and Bruno agree openly on a base color — say yellow — in front of Eva, who listens to them. Each chooses a second secret color in private and mixes their secret with the yellow. Alice gets a particular orange; Bruno gets a particular green. They exchange the results in front of Eva. Now each mixes the received color with their own secret, and both arrive at the same final color, because the order of the mixes does not matter. Eva has seen the yellow and the two intermediate mixes, but not the secrets; without any of the secrets they cannot arrive at the final color. The real mathematics changes the colors for exponentiations in modular groups or elliptic curves, but the idea is the same: the shared secret is built in public without anyone in the channel being able to reconstruct it.

In arithmetic, for whoever prefers to see the mechanism: Alice chooses a secret number a, Bruno chooses b. They exchange g^a and g^b in the open over the channel. Alice calculates $(g^b)^a$ and Bruno calculates $(g^a)^b$; both arrive at the same g^{ab}. Eva sees g, g^a and g^b pass through the channel, but recovering a from g^a — the so-called discrete logarithm problem — requires an astronomical computing time superior to the age of the universe when g is chosen in a suitable mathematical group.

For whoever wants to check it with small numbers. The Diffie-Hellman exchange can be walked through entirely with figures small enough to do the math by hand. Anyone who prefers not to get into arithmetic can skip this block without losing the thread of the article; whoever wants

to see the mechanism working step by step will find it here. **The public rules**, which anyone can read: a prime $p = 11$ (in the real Diffie-Hellman it's about three hundred digits; we use eleven so the math fits on one page), a base $g = 2$, and the convention that all arithmetic is done *modulo p* — you calculate, divide by p, and keep the remainder, like an eleven-position clock that resets to zero when passing ten. **The private choices**, one each and never shared: Alice chooses $a = 4$. Bruno chooses $b = 7$.

Step 1. Alice calculates $2^4 = 16$, then $16 \bmod 11 = \mathbf{5}$. She sends the five. Eva writes it down.

Step 2. Bruno calculates $2^7 = 128$, then $128 \bmod 11 = \mathbf{7}$. He sends the seven. Eva also writes it down. After the two transmissions, Eva's notebook contains four pieces of data: $p = 11$, $g = 2$, A = 5, B = 7. She is missing the shared number that Alice and Bruno are about to derive — and which Eva will not be able to reconstruct.

Step 3. Alice takes the seven that Bruno sent her and raises it to her private exponent $a = 4$. To avoid handling $7^4 = 2401$, it is calculated in parts applying the modulo at each step:

$7^2 = 49$

$49 \bmod 11 = 5$

$7^4 = (7^2)^2 = 5^2 = 25$

$25 \bmod 11 = 3$

Alice obtains the number **3**.

Step 4. Bruno takes the five that Alice sent him and raises it to his private exponent $b = 7$. Again in parts:

$5^2 = 25 \bmod 11 = 3$

$5^4 = (5^2)^2 = 3^2 = 9 \bmod 11 = 9$

$5^6 = 5^4 \times 5^2 = 9 \times 3 = 27 \bmod 11 = 5$

Finally $5^7 = 5^6 \times 5 = 5 \times 5 = 25 \bmod 11 = \mathbf{3}$.

Bruno also obtains **3**.

Both have arrived at the same number, 3, working in parallel. Neither sent their private exponent at any time. Alice does not know that $b = 7$; Bruno does not know that $a = 4$. Each used the public value the other sent combined with their own private exponent, and they met at the same destination. **Why do they reach the same number?** What each calculated: Alice, $(g^b)^a = 2^{7\times4} = 2^{28} \bmod 11$. Bruno, $(g^a)^b = 2^{4\times7} = 2^{28} \bmod 11$. It is the same amount because the order of multiplication of exponents does not matter ($7 \times 4 = 4 \times 7$). Each arrived via a different path to the same destination.

And Eva? She has in her notebook $p = 11$, $g = 2$, A = 5, B = 7, and she would like the 3. To calculate it she would need to know a or b — but neither has traveled through the channel. Her only way is to ask herself: «for what exponent a is $2^a \bmod 11 = 5$?». With p so small she can try 0, 1, 2, 3, 4... and find it in less than a minute. But when replacing 11 with a prime of three hundred digits, the space of possible exponents has more elements than there are atoms in the observable universe. **There is currently no algorithm known to humanity that can traverse that space in less than billions of years.** This is the so-called *discrete logarithm problem*: easy forward, computationally impossible backwards. And it is the reason why the encryption resists even if Eva has followed the entire conversation letter by letter.

Three simple ingredients — clock arithmetic, exponentiation, and the commutativity of multiplication ($a \cdot b = b \cdot a$) — combined produce a protocol that half of humanity depends on every day for their private communications. None of the three pieces, separately, seems special. What is decisive is the assembly.

From Diffie-Hellman to the Signal protocol

The end-to-end encryption used by today's professional messaging applications rests, almost without exception, on an elegant and hardened version of the Diffie-Hellman exchange. The Signal protocol, designed by Trevor Perrin and Moxie Marlinspike between 2013 and 2016, is the reference. It combines two key ideas. The first, the key exchange in elliptic curves (X25519), which produces the initial shared secret between two devices. The second, the so-called Double Ratchet — double gear — which renews the keys automatically with each message, so that compromising the device today does not allow decrypting past messages, nor future messages once the ratchet has been rotated.

In Zig, the X25519 exchange that produces the shared secret between two devices fits in six lines, using the standard library:

```
const std = @import("std");
const X25519 = std.crypto.dh.X25519;

// Alicia y Bruno generan cada uno un par (privada, pública).
const par_alicia = X25519.KeyPair.generate(io);
const par_bruno  = X25519.KeyPair.generate(io);

// Cada parte recibe la clave pública de la otra y deriva el mismo secreto.
const secreto_alicia = X25519.scalarmult(par_alicia.secret_key, par_bruno.public_key) catch unreachable;
```

```
const secreto_bruno  = X25519.scalarmult(par_bruno.secret_key,  par_alicia.public_key) catch unreachable;
// secreto_alicia == secreto_bruno  (32 bytes)
```

What happens in those six lines: The public keys travel openly. The private keys never leave the respective device. Each party derives, from its private and the public of the other, the same secret of thirty-two bytes that no one in the channel can recover. That secret later serves as a seed to encrypt the exchanged messages. The Double Ratchet of the Signal protocol adds a constant rotation of that material so that the compromise of an instant does not compromise the rest of the conversation.

And what exactly is inside `std.crypto.dh.X25519`? No hidden magic. They are two short functions that can be read in their entirety in Zig's own standard library. The first derives the public key from the private one — the «g^a» of the exchange:

```
pub fn recoverPublicKey(secret_key: [secret_length]u8) IdentityElementError![public_length]u8 {
    const q = try Curve.basePoint.clampedMul(secret_key);
    return q.toBytes();
}
```

In the language of the article: the private key is «multiplied» — in the elliptic sense, not the elementary arithmetic one — by the base point of the Curve25519 curve, and the result is serialized into thirty-two bytes. The `clampedMul` operation is the hardened version of that scalar multiplication: it incorporates the safeguards that the cryptographic community added over the years to resist known families of attacks. Two lines of function body.

The second function combines your private key with the public key that the other party sends you. It is the «$(g^b)^a$» of the exchange, which produces the thirty-two byte shared secret that neither of you ever transmitted:

```
pub fn scalarmult(secret_key: [secret_length]u8, public_key: [public_length]u8) IdentityElementError![shared_length]u8 {
    const q = try Curve.fromBytes(public_key).clampedMul(secret_key);
    return q.toBytes();
}
```

Two more lines. The received public key is interpreted as a point on the curve, and «multiplied» by one's own private key. By the commutativity of the curve operation — analogous to the commutativity of the multiplication of exponents that we saw in the numerical example — both parties end up with the same serialized point: exactly the shared secret the article talks about.

That is all. What in an application looks like magic is, in reality, two functions of three lines each. The technical complexity is concentrated in a single operation, `clampedMul`, which is written further down in the same standard library, reviewed for decades by the international cryptographic community, and available to anyone who wants to read it letter by letter. There is no black box either in our application or in Zig's standard library. There is open source code that a human can understand, choosing the pace at which they want to delve into it.

What end-to-end encryption protects

What E2EE protects well, assuming a correct implementation, is the content of the message in transit. An intermediate server that receives and forwards the encrypted data will see a succession of unintelligible bytes. An attacker with access to the cable, the router, the wifi access point, will see the same. A service provider that keeps copies of the traffic will not be able to read it later. A Government that orders the service operator to deliver the content will receive the same unintelligible bytes that the server had in the first place.

This, in practical terms, is a lot. It is the difference between writing a letter inside an opaque envelope and writing it on a postcard. Both arrive. Only one preserves the content from the postman.

What end-to-end encryption does not protect

It's worth knowing it just as well. E2EE does not protect metadata: the server still knows that user A sends data to user B, at what time, with what frequency and from where, although it does not know what they say. This metadata, as we have already argued in *To encrypt is not to be private*, is often more revealing than the content. Knowing that someone called a law firm specialized in divorces on a Friday at 22:00 for thirty minutes tells a story that the content of the call never told. It is the same situation as seeing a person enter and leave several times an oncology clinic: you don't need to hear anything of what is spoken inside to imagine what is happening. A single isolated metadata may mean nothing; several cross-referenced draw something too similar to the truth. E2EE does not protect the ends: if the device of the receiver is compromised by a malicious program, the message is decrypted normally for that receiver and the malicious program reads it. E2EE does not protect against the identity of the interlocutor in itself: if Alice believes she is talking to Bruno but an attacker has interposed at the beginning (a *man in the middle*) and the protocol does not include independent verification, the two parties end up talking to the intruder thinking they are talking to each other.

There is a fourth thing that's worth formulating without ambiguity. E2EE does not prevent a provider that claims to offer it from also keeping a copy of the unencrypted message in its own systems. The statement «my messages are end-to-end encrypted» and the statement «the provider does not keep my content» are not the same. An application can fulfill the first while breaching the second; we have seen it in press headlines repeatedly since 2018. The user, unless the client's code is verifiable, has no technical way to distinguish one case from the other without expert investigation. The most known case in the general public: WhatsApp encrypts messages end-to-end in transit, but if the user activates the backup in iCloud or Google Drive without additional encryption, that copy is stored readable in a third party's infrastructure, and the encryption is broken at the end of the user themselves.

The question the operator does not want to hear

An application that claims to encrypt end-to-end can, technically, do one of three things regarding the keys:

1. **The keys reside only on the devices.** They are generated and reside exclusively on the users' devices; the operator does not know them or store them. It is the optimal case.
2. **The operator can access if they want.** The operator has the users' keys (or can generate them at will) and stores them in their databases. If they want or are forced to, they can read the content. This is the case for most 'cloud' services.
3. **The operator cannot access by design, but controls access.** The operator does not have the keys, but has control over the application that generates them. If forced, they can send a malicious update that captures the keys or content before encrypting. This is the case for many commercial E2EE services.

The operational question, therefore, is not whether something is encrypted, but who has control of the device and the software that manages the keys. In Solo2, the keys reside solely in your Vault (IndexedDB encrypted with your password) and the software is verifiable open source.

For the professional reader

End-to-end encryption is a tool for digital sovereignty. But like every tool, its effectiveness depends on the hand that wields it and the ground on which it rests.

1. Where are the cryptographic keys generated and where do they physically reside? If the operator can access them (even temporarily, even under the guise of recovery), the E2EE is nominal.
2. Is there independent verification of the interlocutor (security numbers, QR codes, out-of-band comparison) that prevents a man-in-the-middle attack during the establishment of the conversation?
3. Is the client code auditable — open, published, reproducible — or does it require trusting the provider's word on what the client actually does?
4. What metadata does the service generate and keep, and for how long? Even if the content is opaque, metadata can reconstruct a good part of the sensitive information.

These four questions do not ask for advanced technical information; they ask for information that any honest operator can answer in their public documentation. The quality and precision of the answer says as much about the product as the answer itself.

End-to-end encryption, done right, is one of the finest constructions that contemporary cryptography has delivered to daily practice. The original idea — two people can agree on a secret over a public channel — belongs to Whitfield Diffie and Martin Hellman, 1976; half a century later we continue to live in its consequence. But, as with any technical promise, its value depends on real fulfillment, not on the label. The honest professional's question is not "is it encrypted?", but "who has the keys?". The answers have different consequences. It's worth knowing them.

Sources and further reading

- Diffie, W.; Hellman, M. — *New Directions in Cryptography*, IEEE Transactions on Information Theory, November 1976. Foundational article of public-key cryptography.
- Perrin, T.; Marlinspike, M. — *The Double Ratchet Algorithm*, public specification by Open Whisper Systems, 2016 revision. Basis of the Signal protocol and its industrial derivatives.
- RFC 7748 — Elliptic Curves for Security (IETF, January 2016). Normative specification of the X25519 and X448 curves used in modern key exchanges.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Chapters on key exchange and authenticated encryption protocols.
- Regulation (EU) 2024/1183 on the European digital identity framework (eIDAS 2) — establishes frameworks where independent verification of the interlocutor acquires institutional support, and where the distinction between nominal and real encryption has different legal consequences.

← PreviousKill switch and institutional captureNext →The business model as a signal of trust

Recent readings

- Analysis · May 18, 2026 Real vs. apparent privacy: the questions to ask yourself
- Analysis · May 18, 2026 Self-hosting as a professional practice
- Concept · May 18, 2026 The 24 words: what a cryptographic identity is

Take this article wherever you need it.

↓ Markdown ↓ Plain text ↓ PDF

The file is downloaded to your device. From there, you can save it, import it into Solo2, or share it as you wish. Cuadernos does not decide the destination for you.

Wax seal · SHA-256 b763ce2ff9a1faa121b5380f0a2fe387718c48742e6b096d015e201607c9f66c

Cuadernos Lacre · A publication of Menzuri Gestión S.L. ·
written by R.Eugenio · edited by the team of Solo2.

This website does not use cookies and does not load third-party resources. It uses a self-hosted anonymous visitor counter (Umami, on our European server) and the minimum JavaScript necessary for your light/dark theme preference. No trackers, no profiling, no data sharing. If you want to follow us: RSS.