The 24 Words: What a Cryptographic Identity Is

A cryptographic identity is not a password: no server stores it and it cannot be recovered. A didactic explanation of the BIP39 mechanism, why exactly twenty-four words, and what real burden falls upon those who possess them.

To be clear: If you forget your Gmail password, Google resets it for you. If you lose the 24 words that make up a cryptographic identity, there is no one to ask for them. It is not that the procedure is strict — it is that there is no one at the other end. That difference is everything.

The Difference Between a Password and an Identity

A password, in the classic internet model, is not the user's identity. It is a voucher. The user has an identity —a name, an email address, a customer number— and, to prove to a server that they are who they say they are, they present a password that the server compares against a stored footprint. If the footprints match, the server grants the session. If the password is lost, the user remains the same user; what they lose is the voucher, and there is a recovery procedure —an email to the registered address, a security question— to restore it.

A cryptographic identity works differently. It is not a credential that someone compares against a stored footprint; it *is* a complete mathematical secret in itself. It doesn't matter where it resides —on paper, in a device, even on a foreign server—: the identity exists because of its mathematics, not because of who validates it. Here, a property appears similar to the one we saw in «What SHA-256 Really Is»: possession is not proven by exhibiting the secret, but by using it to sign. The signature thus produced can be checked by anyone with a public value that is mathematically derived from the secret itself, without needing to know the secret, and without a third party mediating the check. Whoever has the secret, is the identity; whoever loses it, ceases to be. The verdict is categorical: **there is no one to ask to return the identity to you. That person does not exist, because they did not have it in the first place.**

What Twenty-Four Words Represent

Cryptographic identity is usually represented by a thirty-two byte mathematical secret —two hundred fifty-six bits. A number that is hard to retain and even harder to transcribe without error. The cryptographic industry solved this problem in 2013 with a small and elegant standard called BIP39: a way to represent those two hundred fifty-six bits as a sequence of twenty-four words taken from an official list of two thousand forty-eight. The arithmetic behind it fits elegantly; those who want to see it in detail will find it in the margin.

The count starts from the end. We want to represent the two hundred fifty-six bits of the secret by adding eight bits of checksum: two hundred sixty-four bits in total. If we distribute them across twenty-four words —a manageable number to write down and dictate without loss— each word must contribute exactly eleven bits of information. And eleven bits are two raised to the power of eleven possibilities, that is, two thousand forty-eight. Hence the official BIP39 vocabulary has exactly that size: the list exists tailored to the problem, not the other way around.

The count is not decorative. If someone transcribes twenty-three words correctly and makes a mistake on the twenty-fourth, the checksum will detect it: the software will tell them "this sequence is invalid." If someone transcribes all twenty-four correctly, the software will derive the same identity unambiguously. The choice of word list is also deliberate: the words in the BIP39 vocabulary are short, distinct from each other, without diacritics, chosen to minimize phonetic and spelling confusions. It is a vocabulary designed to be remembered, written, and dictated by human beings without loss.

From the phrase to the key

The twenty-four words are not the cryptographic key that signs messages. They are a recoverable representation of the original entropy that, through a deterministic process called PBKDF2, is transformed into a sixty-four-byte seed. From that seed derive, also deterministically, the specific cryptographic keys that the user employs: a private key to sign and a corresponding public key that is published to verify the signatures. Same mechanism in different systems: cryptocurrencies use the secp256k1 curve; the Signal protocol and many modern systems use Ed25519 over the Curve25519 curve. For a specific curve like Ed25519, the BIP32 and SLIP-0010 standards take that sixty-four-byte seed and derive, deterministically, the thirty-two bytes that constitute the effective signing key — the same thirty-two bytes with which the code example in the next section starts.

This is the standard way the entire industry presents the mechanism to the user —cryptocurrency wallets, decentralized identity managers, Signal in its persistent identity part, Solo2 among them—: the user, in practice, never sees the seed or the derived keys. They see the twenty-four words when creating their identity and, optionally, write them down on a piece of paper. The words then travel between their devices when they want to migrate the identity: they enter them into the new application, the application derives the same seed, the same keys, the same identity. It is a portable, cryptographically sound and, within reasonable limits, rememberable mechanism.

How to sign with the key (a Zig brushstroke)

In Zig, once you have the thirty-two-byte seed derived from the twenty-four words, signing a message with Ed25519 fits into a few lines:

```
const std = @import("std");
const Ed25519 = std.crypto.sign.Ed25519;

// 'semilla' son los 32 bytes derivados de las 24 palabras.
const par = Ed25519.KeyPair.create(semilla);

// Firmar un mensaje con la clave privada:
const mensaje = "Este mensaje lo escribí yo.";
const firma = try par.sign(mensaje, null);

// Cualquiera con la clave pública del par puede verificar:
try Ed25519.Signature.verify(firma, mensaje, par.public_key);
```

The signing operation produces sixty-four bytes —called a signature— that could only have been generated from the corresponding private key. Verification is public: anyone with the public key can check that the signature corresponds to the message. Without the private key, no one can produce a valid signature for that message; with the public key, everyone can detect if a signature is valid. This asymmetry is what allows the signer to demonstrate authorship without sharing the secret.

The previous example is the minimal manual version. In the real Solo2 code, the chain passes through two files, one in JavaScript that lives in the user's browser and reconstructs the entropy from the twenty-four words,

another in Zig within the *zcatcrypto* library that takes that entropy and derives the specific cryptographic keys. Starting from the browser side:

```
// solo2/web-app/js/lib/bip39.js
async function mnemonicToEntropy(mnemonic, lang) {
    const validation = await validateMnemonic(mnemonic, lang);
    if (!validation.valid) {
        return { entropy: null, valid: false, error: validation.error };
    }
    const wordlist = WORDLISTS[lang || 'en'];
    const words = mnemonic.trim().split(/\s+/);

    // Cada palabra aporta 11 bits (su índice en la lista de 2048).
    let bits = '';
    for (let i = 0; i < words.length; i++) {
        bits += wordlist.indexOf(words[i]).toString(2).padStart(11, '0');
    }

    // 24 palabras = 264 bits. Los primeros 256 son la entropía.
    const entropyBytes = new Uint8Array(32);
    for (let j = 0; j < 32; j++) {
        entropyBytes[j] = parseInt(bits.slice(j * 8, (j + 1) * 8), 2);
    }
    return { entropy: entropyBytes, valid: true };
}
```

Those thirty-two bytes of entropy, along with another thirty-two derived in the same step, travel to the Zig WebAssembly module that generates the Ed25519 keys proper. The full function, with its final memory cleanup, fits on one screen:

```
// zcatcrypto/wasm/bindings/identity.zig
const Ed25519 = std.crypto.sign.Ed25519;
const X25519  = std.crypto.dh.X25519;

export fn identity_generate() ?*IdentityHandle {
    var seed: [64]u8 = undefined;
    if (!common.getRandomBytes(&seed)) return null;

    const handle = common.wasm_allocator.create(IdentityHandle) catch return null;

    // Bytes 0..31: semilla determinista del par Ed25519 (firma).
    const sign_kp = Ed25519.KeyPair.generateDeterministic(seed[0..32].*) catch {
        common.wasm_allocator.destroy(handle);
        return null;
    };
    handle.sign_secret = sign_kp.secret_key.toBytes();
    handle.sign_public = sign_kp.public_key.toBytes();

    // Bytes 32..63: secreto X25519 (para acordar claves de cifrado con el otro).
    handle.exchange_secret = seed[32..64].*;
    handle.exchange_public = X25519.recoverPublicKey(handle.exchange_secret) catch {
        common.wasm_allocator.destroy(handle);
        return null;
    };
```

```
    @memset(&seed, 0);  // Borra la semilla de la memoria.
    return handle;
}
```

Two details are worth noting. The first: a single seed always produces the same key pair — it is exactly this that allows identity recovery by entering the twenty-four words into a new device. The second: the seed is explicitly erased from memory in the last line. Past that point, not even the function itself could reconstruct the keys; the user's words would be the only source.

For those who want to check it with small numbers. The signature scheme can be traced in its entirety with figures small enough to do the math by hand. Those who prefer not to go into arithmetic can skip this block without losing the thread of the article; those who want to see the mechanism working step by step will find it here. **The public rules**, which anyone can read: a prime $p = 23$ (in real Ed25519 it is about seventy-seven digits long; we use twenty-three so the math fits on one page), a base $g = 2$ whose order in this group is $q = 11$, and the convention that all arithmetic with g is done *módulo p* and all exponents are reduced *módulo q*. **The private choice**, a single one and never shared: the secret $x = 6$. That is the identity.

Step 1 — The public part of the identity. It is calculated once and published openly.

$$y = g^x \bmod p$$

$$y = 2^6 \bmod 23 = 64 \bmod 23 = 18$$

The public part of the identity is **18**. Anyone can take it and use it to verify signatures made with this identity. No one, observing only the 18, can recover the secret 6: that is the discrete logarithm problem to which we will return at the end.

Step 2 — Signing a message. The identity holder wants to sign the message $m = 7$. He starts by choosing a new random value $k = 4$, which will be used only once and never shared (in real Ed25519, k is derived deterministically from the message and the secret to avoid the danger of reuse, but the role it plays is exactly this). He then calculates three numbers:

$$r = g^k \bmod p = 2^4 \bmod 23 = 16$$

$$e = H(r, m) \bmod q = (16 + 7) \bmod 11 = 1$$

$$s = (k + x \cdot e) \bmod q = (4 + 6 \cdot 1) \bmod 11 = 10$$

The signature is the pair **(r, s) = (16, 10)**. It travels openly along with the message. Anyone can read it. Didactic note: in real Ed25519 the function H is SHA-512, cryptographically robust; here we use the simplification $e = (r + m) \bmod q$ so the reader can follow the steps without needing to calculate a hash. The algorithm structure is the same.

Step 3 — Verifying the signature. The verifier has the public part $y = 18$, the message $m = 7$, and the signature $(r, s) = (16, 10)$. He reconstructs e the same way — $e = (16 + 7) \bmod 11 = 1$ — and checks if this equality holds:

$$g^s \bmod p \stackrel{?}{=} r \cdot y^e \bmod p$$

Calculate both sides separately:

Izquierda: $2^{10} \bmod 23 = 1024 \bmod 23 = 12$

Derecha: $16 \cdot 18^1 \bmod 23 = 288 \bmod 23 = 12$

Both sides give **12**. The signature is valid. Anyone with the public part 18 can reach this conclusion without ever knowing that the secret was 6.

What about a third party trying to forge? Eva has seen everything public pass through the channel: $p = 23$, $g = 2$, $q = 11$, $y = 18$, $m = 7$, $r = 16$, $s = 10$. To sign a *different* message on behalf of this identity, she would need to know x. Her only way is to ask herself: "for what exponent x does $2^x \bmod 23 = 18$ hold?". With $p = 23$ she can try 0, 1, 2, 3, … and find it in seconds. But when substituting 23 for a prime of Ed25519's real dimensions, the space of possible exponents exceeds the number of atoms in the observable universe. **There is today no algorithm known to humanity that can traverse that space in less than billions of years.** It is the same discrete logarithm problem that supports the Diffie-Hellman from the previous article, applied here to the signature scheme.

What we have just gone through is *exactly* Schnorr, the signature scheme of which Ed25519 is a variant adapted to an elliptic curve. In real Ed25519, all operations are done on the points of a specific curve (Curve25519) instead of on integers modulo a prime, and the function H is SHA-512 instead of the toy sum we used above. The two substitutions are implementation adjustments — gaining cryptographic resistance to brute force, gaining additional security properties for k. The algorithmic structure, the three operations, the reason for the asymmetry, are the same.

A brief pause is appropriate here, because the entire chain can be confused at a quick glance with another primitive from the trio: the hash. It is not. A hash is a unique function that compresses — many bytes enter, a short fingerprint comes out, the road ends there. A cryptographic identity is a complementary mathematical pair: the secret stays and signs; its public counterpart is published and verifies. Where the hash collapses information in a single direction, identity establishes an asymmetry between two halves. The hash testifies to what was said; identity testifies to who said it.

What the phrase is not

Three frequent misconceptions should be cleared up. The phrase is not a password in the proper sense: it is not compared against a fingerprint stored on a server; it is entered into the user's device to mathematically reconstruct the identity. The phrase is not recovered: if it is lost, there is no one to ask for it; if it is duplicated, the identity is also duplicated. The phrase is not a credential separable from the identity: the phrase *is* the identity. Whoever has it can act as it, without additional permission, without an authorization process, without possible recovery.

This third property is what changes the weight of the matter. A lost password is an administrative nuisance. A lost cryptographic identity is the identity. A paper with the phrase found by third parties is not a risk of account theft: it is the delivery of the entire identity. The promise of the system —that no one can revoke your identity or block you arbitrarily— is accompanied inseparably by the responsibility —that you are the sole custodian of something that no one can restore for you.

The promise and the weight

The model of cryptographic identity often receives the descriptor *self-sovereign*. The choice of word is deliberate and describes the condition with fair accuracy. The user is sovereign over their identity in an almost medieval sense: it is not granted by any king, any issuer, any central authority; nor can it be withdrawn by any of the former. But also, like the medieval monarch, the user bears the entire consequence of their mistakes: there is no regent to make decisions in their place if they lose the seal.

The choice between identity managed by a third party and self-sovereign identity does not have a single correct universal answer. For an irrelevant forum account, managed identity is probably proportional to the risk. For a professional identity that signs legally binding documents, for an economic identity that guards one's own savings, for a professional communication identity with clients who have entrusted sensitive information, the

matter changes. There the question stops being «is it convenient?» and becomes «who, besides me, has the power to act as me, and under what circumstances?».

Where this mechanism appears in real systems

BIP39 was born in the world of Bitcoin in 2013 and quickly spread to the entire cryptocurrency ecosystem: any serious wallet today accepts a twelve or twenty-four word BIP39 phrase as a backup of its holder's economic identity. Outside of cryptocurrencies, the same underlying concept — a cryptographic pair that proves authorship without an intermediary — appears in other systems with different syntax. The SSH keys that a system administrator uses to access their servers are a classic case: a private key that the administrator keeps on their machine and a public one that is copied to each server; no entity comparable to a centralized service intervenes. The Signal protocol uses Ed25519 with persistent key material on the device; the European eIDAS, in its qualified signature part, rests on the same cryptographic principle, with the difference that the key is kept by a qualified trust service provider instead of the user.

Solo2, the publishing platform of this publication, uses a twenty-four word BIP39 phrase as each user's identity. The user, upon creating their account, sees the words once. They are not stored on any Solo2 server or anyone else's: if the user writes them down and guards them, they keep their identity forever. If they lose them, they lose them. It is the coherent consequence of an architecture with no operator in the middle: if Solo2 could return the identity to the user who lost it, it could also give it to whoever pressures Solo2 to provide it.

For the professional reader

Four considerations for those evaluating adopting self-sovereign identity in a professional context:

1. The phrase is the identity. Physical custody — paper, several copies in different places, eventually engraved metal for long-term use — offers more guarantees than digital custody, which adds attack surface without reducing the risk of loss.
2. There is no recovery. Designing the process assuming that one day the primary copy will be lost is much better than discovering it on the day it is lost. A second geographically separate copy solves almost all scenarios.
3. It is not the same as an eIDAS qualified certificate. For qualified signatures in the Union — notarial deeds, certain procedures with the Administration — the legislation requires a qualified provider to hold the key. Self-sovereign cryptographic identity serves professional communication and documentary signing with evidentiary value, but does not automatically replace the qualified certificate in cases where the regulation requires it.
4. If the identity is to be transferred — inheritance, professional succession, closing of activity — it is advisable to prepare the procedure before, not after. Formal procedures with sealed (lacre) envelopes, instructions to an executor, deposit in a notary's office, are classic arrangements perfectly compatible with the cryptographic nature of the asset.

This article closes the conceptual trio that opened the cycle — hash, encryption, identity —. The three ideas build upon each other: hash gives the unalterable fingerprint, encryption gives confidentiality without a trusted third party, identity gives authorship without a granting third party. The three share a property that is not ideological either: they transfer, from the service manager to the user, technical capabilities that traditionally resided in the operator. They also transfer responsibilities with them. Speaking honestly about any of the three requires also speaking about the other two.

Sources and further reading

- Palatinus, M.; Rusnak, P.; Voisine, A.; Bowe, S. — *BIP-0039: Mnemonic code for generating deterministic keys*, Bitcoin improvement proposal of 2013. De facto standard for recovery phrases in the

crypto industry.
- RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA), including Ed25519. IETF, January 2017. Normative specification of the signature scheme used in much of the contemporary industry.
- RFC 2898 — PKCS #5: Password-Based Cryptography Specification, version 2.0. IETF, September 2000. Defines the PBKDF2 algorithm used in BIP39 phrase-to-seed derivation.
- Regulation (EU) 910/2014 (eIDAS) and its evolution by Regulation (EU) 2024/1183 (eIDAS 2) — European framework for electronic identity and qualified signature. Different regime from self-sovereign, but conceptually supported by the same cryptographic primitives.
- Allen, C. — *The Path to Self-Sovereign Identity* (2016). Canonical text on the principles and commitments of the self-sovereign model, prior but relevant for the understanding of the family of contemporary solutions.

← PreviousThe business model as a signal of trustNext →Self-hosting as a professional practice

Recent readings

- Reflection · June 29, 2026 You Are Not Anonymous
- Reflection · May 27, 2026 What a signature cannot fix
- Analysis · May 26, 2026 Real vs. apparent privacy: the questions to ask yourself

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 d03d77fd3cef8fa7d9c23da5d90e4e9747cd7aa8381c3e48a5ebe90f73b136e6

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