Le 24 parole: cos'è un'identità crittografica

Un'identità crittografica non è una password: nessun server la conserva e non si può recuperare. Una spiegazione didattica del meccanismo BIP39, perché esattamente ventiquattro parole e quale peso reale ricade su chi le possiede.

Per intenderci: se dimentichi la password di Gmail, Google la reimposta per te. Se perdi le 24 parole che compongono un'identità crittografica, non c'è nessuno a cui chiederle. Non è che la procedura sia rigida — è che non esiste nessuno all'altro capo. Questa differenza è tutto.

La differenza tra una password e un'identità

Una password, nel modello classico di internet, non è l'identità dell'utente. È una ricevuta. L'utente ha un'identità — un nome, un'e-mail, un numero cliente — e, per dimostrare a un server di essere chi dice di essere, presenta una password che il server confronta con un'impronta memorizzata. Se le impronte coincidono, il server concede la sessione. Se la password viene persa, l'utente rimane lo stesso utente; ciò che perde è la ricevuta, ed esiste una procedura di recupero — un'e-mail all'indirizzo registrato, una domanda di sicurezza — per ripristinarla.

Un'identità crittografica funziona in modo diverso. Non è una credenziale che qualcuno confronta con un'impronta memorizzata; *è* un segreto matematico completo in sé. Non importa dove risieda — su un foglio, in un dispositivo, persino su un server altrui — l'identità esiste per la sua matematica, non per chi la convalida. Qui appare una proprietà simile a quella che abbiamo visto in «Cos'è veramente SHA-256»: la proprietà non si dimostra esibendo il segreto, ma usandolo per firmare. La firma così prodotta può essere verificata da chiunque con un valore pubblico che deriva matematicamente dal segreto stesso, senza necessità di conoscere il segreto e senza che un terzo intervenga nella verifica. Chi ha il segreto è l'identità; chi lo perde cessa di esserlo. La sentenza è categorica: **non c'è nessuno a cui chiedere di restituirti l'identità. Quel qualcuno non esiste, perché non la possedeva in primo luogo.**

Cosa rappresentano ventiquattro parole

L'identità crittografica è solitamente rappresentata da un segreto matematico di trentadue byte — duecentocinquantasei bit. Un numero difficile da ricordare e ancora più difficile da trascrivere senza errori. L'industria crittografica ha risolto questo problema nel 2013 con uno standard piccolo ed elegante chiamato BIP39: un modo per rappresentare quei duecentocinquantasei bit come una sequenza di ventiquattro parole tratte da un elenco ufficiale di duemilaquarantotto. L'aritmetica sottostante si incastra con eleganza; chi vuole vederla nel dettaglio la trova a margine.

Il calcolo inizia dalla fine. Vogliamo rappresentare i duecentocinquantasei bit del segreto aggiungendo otto bit di checksum: duecentosessantaquattro bit in totale. Se li dividiamo in ventiquattro parole — un numero gestibile da annotare e dettare senza perdite — ogni parola deve fornire esattamente undici bit di informazioni. E undici bit sono due elevato alla undicesima possibilità, ovvero duemilaquarantotto. Ecco perché il vocabolario ufficiale BIP39 ha esattamente quella dimensione: l'elenco esiste su misura per il problema, non il contrario.

Il calcolo non è decorativo. Se qualcuno trascrive ventitré parole correttamente e sbaglia la ventiquattresima, il checksum lo rileverà: il software gli dirà "questa sequenza non è valida". Se qualcuno trascrive tutte le ventiquattro correttamente, il software deriverà la stessa identità senza ambiguità. Anche la scelta dell'elenco di parole è deliberata: le parole del vocabolario BIP39 sono brevi, distinte tra loro, senza diacritici, scelte per ridurre al minimo le confusioni fonetiche e ortografiche. È un vocabolario progettato per essere ricordato, scritto e dettato dagli esseri umani senza perdite.

Dalla frase alla chiave

Le ventiquattro parole non sono la chiave crittografica che firma i messaggi. Sono una rappresentazione recuperabile dell'entropia originale che, attraverso un processo deterministico chiamato PBKDF2, viene trasformata in un seme (seed) di sessantaquattro byte. Da quel seme derivano, anch'esse in modo deterministico, le chiavi crittografiche concrete utilizzate dall'utente: una chiave privata per firmare e una corrispondente chiave pubblica che viene pubblicata per verificare le firme. Stesso meccanismo in sistemi diversi: le criptovalute usano la curva secp256k1; il protocollo Signal e molti sistemi moderni usano Ed25519 sulla curva Curve25519. Per una curva concreta come Ed25519, gli standard BIP32 e SLIP-0010 prendono quel seme di sessantaquattro byte e derivano deterministicamente i trentadue byte che costituiscono la chiave di firma effettiva — gli stessi trentadue byte con cui inizia l'esempio di codice nella sezione successiva.

Questo è il modo standard in cui l'intera industria presenta il meccanismo all'utente —portafogli di criptovalute, gestori di identità decentralizzata, Signal nella sua parte di identità persistente, Solo2 tra questi—: l'utente, in pratica, non vede mai il seme né le chiavi derivate. Vede le ventiquattro parole quando crea la sua identità e, opzionalmente, le annota su un foglio di carta. Le parole viaggiano poi tra i suoi dispositivi quando desidera migrare l'identità: le inserisce nella nuova applicazione, l'applicazione deriva lo stesso seme, le stesse chiavi, la stessa identità. È un meccanismo portatile, crittograficamente solido e, entro i limiti della ragionevolezza, memorizzabile.

Come firmare con la chiave (una pennellata di Zig)

In Zig, una volta ottenuto il seme di trentadue byte derivato dalle ventiquattro parole, firmare un messaggio con Ed25519 richiede solo poche righe:

```
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);
```

L'operazione di firma produce sessantaquattro byte —chiamati firma— che possono essere stati generati solo dalla corrispondente chiave privata. La verifica è pubblica: chiunque possieda la chiave pubblica può verificare che la firma corrisponda al messaggio. Senza la chiave privata, nessuno può produrre una firma valida per quel messaggio; con la chiave pubblica, tutti possono rilevare se una firma è valida. Questa asimmetria è ciò che permette al firmatario di dimostrare la paternità senza condividere il segreto.

L'esempio precedente è la versione minima da manuale. Nel codice reale di Solo2 la catena attraversa due file, uno in JavaScript che vive nel browser dell'utente e ricostruisce l'entropia a partire dalle ventiquattro parole, un

altro in Zig all'interno della libreria *zcatcrypto* che prende quell'entropia e deriva le chiavi crittografiche concrete. Cominciando dal lato del browser:

```
// 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 };
}
```

Quei trentadue byte di entropia, insieme ad altri trentadue derivati nello stesso passaggio, viaggiano verso il modulo WebAssembly di Zig che genera le chiavi Ed25519 vere e proprie. La funzione completa, con la sua pulizia finale della memoria, entra in una schermata:

```
// 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;
}
```

Due dettagli meritano di essere sottolineati. Il primo: un medesimo seme (seed) produce sempre la stessa coppia di chiavi — è esattamente questo che permette di recuperare l'identità inserendo le ventiquattro parole in un nuovo dispositivo. Il secondo: il seme viene cancellato esplicitamente dalla memoria nell'ultima riga. Oltre quel punto, nemmeno la funzione stessa potrebbe ricostruire le chiavi; le parole dell'utente sarebbero l'unica origine.

Per chi volesse verificarlo con numeri piccoli. Lo schema di firma può essere percorso interamente con cifre abbastanza ridotte da poter fare i conti a mano. Chi preferisce non entrare nell'aritmetica può saltare questo blocco senza perdere il filo dell'articolo; chi vuole vedere il meccanismo funzionare passo dopo passo lo troverà qui. **Le regole pubbliche**, che chiunque può leggere: un primo $p = 23$ (nel Ed25519 reale è di circa settantasette cifre; usiamo ventitré affinché i conti entrino in una pagina), una base $g = 2$ il cui ordine in questo gruppo è $q = 11$, e la convenzione che tutta l'aritmetica con g si fa *módulo p* e tutti gli esponenti si riducono *módulo q*. **La scelta privata**, una sola e mai condivisa: il segreto $x = 6$. Questa è l'identità.

Passaggio 1 — La parte pubblica dell'identità. Si calcola una volta e si pubblica apertamente.

$y = g^x \bmod p$

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

La parte pubblica dell'identità è **18**. Chiunque può prenderla e usarla per verificare firme effettuate con questa identità. Nessuno, osservando solo il 18, può recuperare il segreto 6: questo è il problema del logaritmo discreto al quale torneremo alla fine.

Passaggio 2 — Firmare un messaggio. Il possessore dell'identità vuole firmare il messaggio $m = 7$. Inizia scegliendo un nuovo valore casuale $k = 4$, che verrà usato una sola volta e non sarà mai condiviso (nel Ed25519 reale, k è derivato deterministicamente dal messaggio e dal segreto per evitare il pericolo di riutilizzo, ma il ruolo che gioca è esattamente questo). Successivamente calcola tre numeri:

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

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

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

La firma è la coppia **(r, s) = (16, 10)**. Viaggia in chiaro insieme al messaggio. Chiunque può leggerla. Nota didattica: nel Ed25519 reale la funzione H è SHA-512, crittograficamente robusta; qui usiamo la semplificazione $e = (r + m) \bmod q$ affinché il lettore possa percorrere i passaggi senza necessità di calcolare un hash. La struttura dell'algoritmo è la stessa.

Passaggio 3 — Verificare la firma. Il verificatore possiede la parte pubblica $y = 18$, il messaggio $m = 7$, e la firma $(r, s) = (16, 10)$. Ricostruisce e allo stesso modo — $e = (16 + 7) \bmod 11 = 1$ — e controlla se questa uguaglianza è soddisfatta:

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

Calcola i due lati separatamente:

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

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

I due lati danno **12**. La firma è valida. Chiunque con la parte pubblica 18 può arrivare a questa conclusione senza aver mai saputo che il segreto era 6.

E un terzo che tentasse di falsificare? Eva ha visto passare per il canale tutto ciò che è pubblico: $p = 23$, $g = 2$, $q = 11$, $y = 18$, $m = 7$, $r = 16$, $s = 10$. Per firmare un messaggio *diverso* a nome di questa identità, avrebbe bisogno di conoscere x. La sua unica via è chiedersi: «per quale esponente x si ottiene $2^x \bmod 23 = 18$?». Con $p = 23$ può provare 0, 1, 2, 3, … e trovarlo in pochi secondi. Ma sostituendo 23 con un primo dalle dimensioni reali di Ed25519, lo spazio degli esponenti possibili supera il numero di atomi dell'universo osservabile. **Non esiste ad oggi alcun algoritmo noto all'umanità che possa percorrere quello spazio in meno di miliardi di anni.** È lo stesso problema del logaritmo discreto che sostiene il Diffie-Hellman dell'articolo precedente, applicato qui allo schema di firma.

Ciò che abbiamo appena percorso è *esattamente* Schnorr, lo schema di firma di cui Ed25519 è una variante adattata a una curva ellittica. Nel Ed25519 reale, tutte le operazioni vengono eseguite sui punti di una curva concreta (Curve25519) invece che su numeri interi modulo un primo, e la funzione H è SHA-512 invece della somma giocattolo che abbiamo usato sopra. Le due sostituzioni sono regolazioni dell'implementazione — guadagnare resistenza crittografica alla forza bruta, guadagnare proprietà aggiuntive di sicurezza per k —. La struttura algoritmica, le tre operazioni, il perché dell'asimmetria, sono le stesse.

Conviene qui una breve sosta, perché l'intera catena può essere confusa a colpo d'occhio con un'altra primitiva del trio: l'hash. Non lo è. Un hash è una funzione unica che comprime — entrano molti byte, esce un'impronta corta, lì finisce il cammino —. Un'identità crittografica è una coppia matematica complementare: il segreto resta e firma; la sua controparte pubblica viene pubblicata e verifica. Dove l'hash collassa l'informazione in un unico senso, l'identità stabilisce un'asimmetria tra due metà. L'hash attesta cosa è stato detto; l'identità attesta chi lo ha detto.

Ciò che la frase non è

È opportuno chiarire tre equivoci frequenti. La frase non è una password in senso proprio: non viene confrontata con un'impronta memorizzata su un server; viene inserita nel dispositivo dell'utente per ricostruire matematicamente l'identità. La frase non si recupera: se viene persa, non c'è nessuno a cui chiederla; se viene duplicata, viene duplicata anche l'identità. La frase non è una credenziale separabile dall'identità: la frase *è* l'identità. Chi la possiede può agire come tale, senza ulteriori permessi, senza processi di autorizzazione, senza possibilità di recupero.

È questa terza proprietà che cambia il peso della questione. Una password persa è un inconveniente amministrativo. Un'identità crittografica persa è l'identità stessa. Un foglio con la frase trovato da terzi non è un rischio di furto dell'account: è la consegna dell'intera identità. La promessa del sistema — che nessuno possa revocare la tua identità o bloccarti arbitrariamente — è accompagnata inscindibilmente dalla responsabilità — che tu sia l'unico custode di qualcosa che nessuno può restituire per te.

La promessa e il peso

Il modello di identità crittografica riceve spesso l'appellativo di *auto-sovrana* —self-sovereign nella letteratura anglosassone—. La scelta del termine è deliberata e descrive con discreta esattezza la condizione. L'utente è sovrano sulla sua identità in un senso quasi medievale: non viene concessa da alcun re, alcun emittente, alcuna autorità centrale; né può essere revocata da nessuno dei precedenti. Ma anche, come il monarca medievale, l'utente sopporta l'intera conseguenza dei suoi errori: non c'è reggente che prenda decisioni al suo posto se perde il sigillo.

La scelta tra un'identità gestita da terzi e un'identità auto-sovrana non ha una risposta universale corretta. Per l'account di un forum irrilevante, l'identità gestita è probabilmente proporzionata al rischio. Per un'identità professionale che firma documenti legalmente vincolanti, per un'identità economica che custodisce i propri

risparmi, per un'identità di comunicazione professionale con clienti che hanno affidato informazioni sensibili, la questione cambia. Lì la domanda smette di essere «è comodo?» e diventa «chi, oltre a me, ha il potere di agire come me, e in quali circostanze?».

Dove appare questo meccanismo nei sistemi reali

Il BIP39 è nato nel mondo di Bitcoin nel 2013 e si è diffuso rapidamente nell'intero ecosistema delle criptovalute: qualsiasi portafoglio serio accetta oggi una frase BIP39 di dodici o ventiquattro parole come backup dell'identità economica del suo titolare. Al di fuori delle criptovalute, lo stesso concetto sottostante — una coppia crittografica che prova la paternità senza intermediari — appare in altri sistemi con sintassi diversa. Le chiavi SSH che un amministratore di sistema usa per accedere ai propri server sono un caso classico: una chiave privata che l'amministratore conserva sulla propria macchina e una pubblica che viene copiata su ogni server; non interviene alcuna entità paragonabile a un servizio centralizzato. Il protocollo Signal usa Ed25519 con materiale di chiave persistente sul dispositivo; l'eIDAS europeo, nella sua parte di firma qualificata, poggia sullo stesso principio crittografico, con la differenza che la chiave è custodita da un fornitore di servizi fiduciari qualificato invece che dall'utente.

Solo2, piattaforma editrice di questa pubblicazione, usa una frase BIP39 di ventiquattro parole come identità per ogni utente. L'utente, al momento della creazione del proprio account, vede le parole una sola volta. Non vengono memorizzate su alcun server di Solo2 né di nessun altro: se l'utente le annota e le custodisce, mantiene la propria identità per sempre. Se le perde, le perde. È la conseguenza coerente di un'architettura senza operatore intermedio: se Solo2 potesse restituire l'identità all'utente che l'ha persa, potrebbe anche darla a chiunque faccia pressione su Solo2 per ottenerla.

Per il lettore professionale

Quattro considerazioni per chi valuta l'adozione di un'identità crittografica autosovrana (autosoberana) in un contesto professionale :

1. La frase è l'identità. La custodia fisica — carta, diverse copie in luoghi differenti, eventualmente metallo inciso per un uso a lungo termine — offre più garanzie della custodia digitale, che aggiunge superficie di attacco senza ridurre il rischio di perdita.
2. Non c'è recupero. Progettare il processo assumendo che un giorno la copia primaria andrà persa è molto più saggio che scoprirlo il giorno in cui accade. Una seconda copia geograficamente separata risolve quasi tutti gli scenari.
3. Non è la stessa cosa di un certificato qualificato eIDAS. Per la firma qualificata nell'Unione — atti notarili, determinate procedure con l'Amministrazione — la legislazione richiede un fornitore qualificato che custodisca la chiave. L'identità crittografica autosovrana serve per la comunicazione professionale e la firma documentale con valore probatorio, ma non sostituisce automaticamente il certificato qualificato nei casi in cui la norma lo richiede.
4. Se l'identità deve essere trasferita — eredità, successione professionale, chiusura dell'attività — conviene preparare la procedura prima, non dopo. Procedure formalle con buste sigillate con cera lacca (lacre), istruzioni a un esecutore testamentario, deposito presso un notaio, sono accordi classici perfettamente compatibili con la natura crittografica dell'asset.

Questo articolo chiude il trio concettuale che ha aperto il ciclo — hash, cifratura, identità —. Le tre idee si costruiscono l'una sull'altra: l'hash fornisce l'impronta inalterabile, la cifratura fornisce la riservatezza senza terze parti fidate, l'identità fornisce la paternità senza terze parti concedenti. Tutte e tre condividono una proprietà che non è neppure ideologica: trasferiscono, dal gestore di un servizio a chi lo usa, capacità tecniche che tradizionalmente risiedevano nell'operatore. Con esse trasferiscono anche responsabilità. Parlare onestamente di una qualsiasi delle tre richiede di parlare anche delle altre due.

Fonti e letture aggiuntive

- Palatinus, M.; Rusnak, P.; Voisine, A.; Bowe, S. — *BIP-0039: Mnemonic code for generating deterministic keys*, proposta di miglioramento di Bitcoin del 2013. Standard de facto per le frasi di recupero nell'industria crittografica.
- RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA), incluso Ed25519. IETF, gennaio 2017. Specifica normativa dello schema di firma usato in gran parte dell'industria contemporanea.
- RFC 2898 — PKCS #5: Password-Based Cryptography Specification, versione 2.0. IETF, settembre 2000. Definisce l'algoritmo PBKDF2 usato nella derivazione BIP39 da frase a seme (seed).
- Regolamento (UE) 910/2014 (eIDAS) e la sua evoluzione con il Regolamento (UE) 2024/1183 (eIDAS 2) — quadro europeo per l'identità elettronica e la firma qualificata. Regime diverso da quello autosovrano, ma concettualmente supportato dagli stessi primitivi crittografici.
- Allen, C. — *The Path to Self-Sovereign Identity* (2016). Testo canonico sui principi e gli impegni del modello autosovrano, precedente ma rilevante per la comprensione della famiglia di soluzioni contemporanee.

← PrecedenteIl modello di business come segnale di fiduciaSuccessivo →Self-hosting come pratica professionale

Letture recenti

- Riflessione · 29 giugno 2026 Non sei anonimo
- Riflessione · 27 maggio 2026 Ciò che una firma non può risolvere
- Analisi · 26 maggio 2026 Privacy reale vs apparente: le domande da porsi

Porta questo articolo dove ne hai bisogno.

↓ Markdown ↓ Testo semplice ↓ PDF

Il file viene scaricato sul tuo dispositivo. Da lì puoi salvarlo, importarlo in Solo2 o condividerlo come preferisci. Cuadernos non decide la destinazione per te.

Sigillo di cera · SHA-256 8a05da67499d64c7280b008238bb4f12996c0481823ade0770fbdd8cf002b1b4

Cuadernos Lacre · Una pubblicazione di Menzuri Gestión S.L. ·
scritta da R.Eugenio · a cura del team di Solo2.