Las 24 palabras: qué es una identidad criptográfica

Una identidad criptográfica no es una contraseña: no la guarda ningún servidor y no se recupera. Una explicación didáctica del mecanismo BIP39, por qué exactamente veinticuatro palabras, y qué peso real recae sobre quien las posee.

Para entendernos: Si olvidas tu contraseña de Gmail, Google te la restablece. Si pierdes las 24 palabras que componen una identidad criptográfica, no hay a quién pedírselas. No es que el procedimiento sea estricto — es que no existe nadie en la otra punta. Esa diferencia es toda la diferencia.

La diferencia entre una contraseña y una identidad

Una contraseña, en el modelo clásico de internet, no es la identidad del usuario. Es un comprobante. El usuario tiene una identidad —un nombre, un correo electrónico, un número de cliente— y, para demostrar ante un servidor que es quien dice ser, presenta una contraseña que el servidor compara contra una huella que tenía almacenada. Si las huellas coinciden, el servidor concede la sesión. Si la contraseña se pierde, el usuario sigue siendo el mismo usuario; lo que pierde es el comprobante, y existe un procedimiento de recuperación —un correo a la dirección registrada, una pregunta de seguridad— para restituirlo.

Una identidad criptográfica funciona de otra manera. No es una credencial que alguien compare contra una huella almacenada; *es* un secreto matemático completo en sí mismo. Da igual dónde resida —en un papel, en un dispositivo, incluso en un servidor ajeno—: la identidad existe por su matemática, no por quién la valida. Aquí aparece una propiedad parecida a la que vimos en «Qué es realmente SHA-256»: la posesión no se demuestra exhibiendo el secreto, sino usándolo para firmar. La firma así producida cualquiera puede comprobarla con un valor público que se deriva matemáticamente del propio secreto, sin necesidad de conocer el secreto en sí, y sin que un tercero medie en la comprobación. Quien tiene el secreto, es la identidad; quien lo pierde, deja de serla. La sentencia es categórica: **no hay nadie a quien pedirle que te devuelva la identidad. No existe ese alguien, porque no la tenía en primer lugar.**

Lo que representan veinticuatro palabras

La identidad criptográfica se representa habitualmente con un secreto matemático de treinta y dos bytes —doscientos cincuenta y seis bits—. Un número difícil de retener y aún más difícil de transcribir sin error. La industria criptográfica resolvió este problema en 2013 con un estándar pequeño y elegante llamado BIP39: una forma de representar esos doscientos cincuenta y seis bits como una secuencia de veinticuatro palabras tomadas de una lista oficial de dos mil cuarenta y ocho. La aritmética detrás encaja con elegancia; quien quiera verla en detalle la encuentra al margen.

La cuenta empieza por el final. Queremos representar los doscientos cincuenta y seis bits del secreto añadiendo ocho bits de suma de comprobación: doscientos sesenta y cuatro bits en total. Si los repartimos en veinticuatro palabras —un número manejable para anotar y dictar sin pérdida—, cada palabra ha de aportar exactamente once bits de información. Y once bits son dos elevado a once posibilidades distintas, es decir, dos mil cuarenta y ocho. De ahí que el vocabulario oficial BIP39 tenga precisamente ese tamaño: la lista existe a medida del problema, no al revés.

La cuenta no es decorativa. Si alguien transcribe veintitrés palabras correctamente y se equivoca en la veinticuatro, la suma de comprobación lo detectará: el software le dirá «esta secuencia no es válida». Si alguien transcribe las veinticuatro correctas, el software derivará la misma identidad sin ambigüedad. La elección de la lista de palabras también es deliberada: las palabras del vocabulario BIP39 son cortas, distintas entre sí, sin diacríticos, escogidas para minimizar confusiones fonéticas y ortográficas. Es un vocabulario diseñado para ser recordado, escrito y dictado por seres humanos sin pérdida.

De la frase a la clave

Las veinticuatro palabras no son la clave criptográfica que firma mensajes. Son una representación recuperable de la entropía original que, mediante un proceso determinista llamado PBKDF2, se transforma en una semilla de sesenta y cuatro bytes. De esa semilla derivan, también de forma determinista, las claves criptográficas concretas que el usuario emplea: una clave privada para firmar y una clave pública correspondiente que se publica para verificar las firmas. Mismo mecanismo en sistemas distintos: las criptomonedas usan curva secp256k1; el protocolo Signal y muchos sistemas modernos usan Ed25519 sobre la curva Curve25519. Para una curva concreta como Ed25519, los estándares BIP32 y SLIP-0010 toman esa semilla de sesenta y cuatro bytes y derivan, de forma determinista, los treinta y dos bytes que constituyen la clave de firma efectiva — los mismos treinta y dos bytes con los que arranca el ejemplo de código de la próxima sección.

Esta es la forma estándar en que la industria entera presenta el mecanismo al usuario —monederos de criptomonedas, gestores de identidad descentralizada, Signal en su parte de identidad persistente, Solo2 entre ellos—: el usuario, en la práctica, nunca ve la semilla ni las claves derivadas. Ve las veinticuatro palabras al crear su identidad y, opcionalmente, las anota en un papel. Las palabras viajan luego entre sus dispositivos cuando quiere migrar la identidad: las introduce en la aplicación nueva, la aplicación deriva la misma semilla, las mismas claves, la misma identidad. Es un mecanismo portable, criptográficamente sólido y, dentro de los límites de lo razonable, recordable.

Cómo se firma con la clave (pincelada Zig)

En Zig, una vez se tiene la semilla de treinta y dos bytes derivada de las veinticuatro palabras, firmar un mensaje con Ed25519 cabe en pocas líneas:

```
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);
```

La operación de firmar produce sesenta y cuatro bytes —llamados firma— que solo pudieron generarse a partir de la clave privada correspondiente. La verificación es pública: cualquiera con la clave pública puede comprobar que la firma corresponde al mensaje. Sin la clave privada, nadie puede producir una firma válida para ese mensaje; con la clave pública, todos pueden detectar si una firma es válida. Esa asimetría es lo que permite que el firmante demuestre autoría sin compartir el secreto.

El ejemplo anterior es la versión mínima de manual. En el código real de Solo2 la cadena atraviesa dos archivos, uno en JavaScript que vive en el navegador del usuario y reconstruye la entropía a partir de las veinticuatro

palabras, otro en Zig dentro de la biblioteca *zcatcrypto* que toma esa entropía y deriva las claves criptográficas concretas. Empezando por el lado del navegador:

```
// 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 };
}
```

Esos treinta y dos bytes de entropía, junto con otros treinta y dos derivados en el mismo paso, viajan al módulo WebAssembly de Zig que genera las claves Ed25519 propiamente dichas. La función completa, con su limpieza de memoria final, cabe en una pantalla:

```
// 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;
}
```

Dos detalles vale la pena señalar. El primero: una misma semilla produce siempre el mismo par de claves —es exactamente eso lo que permite recuperar la identidad introduciendo las veinticuatro palabras en un dispositivo nuevo. El segundo: la semilla se borra explícitamente de la memoria en la última línea. Pasado ese punto, ni siquiera la propia función podría reconstruir las claves; las palabras del usuario serían el único origen.

Para quien quiera comprobarlo con números pequeños. El esquema de firma se puede recorrer entero con cifras lo bastante reducidas como para hacer las cuentas a mano. Quien prefiera no entrar en aritmética puede saltarse este bloque sin perder el hilo del artículo; quien quiera ver el mecanismo funcionando paso a paso lo encontrará aquí. **Las reglas públicas**, que cualquiera puede leer: un primo $p = 23$ (en Ed25519 real es de unas setenta y siete cifras; usamos veintitrés para que las cuentas quepan en una página), una base $g = 2$ cuyo orden en este grupo es $q = 11$, y la convención de que toda la aritmética con g se hace *módulo p* y todos los exponentes se reducen *módulo q*. **La elección privada**, una sola y jamás compartida: el secreto $x = 6$. Esa es la identidad.

Paso 1 — La parte pública de la identidad. Se calcula una vez y se publica abiertamente.

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

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

La parte pública de la identidad es **18**. Cualquiera puede tomarla y usarla para verificar firmas hechas con esta identidad. Nadie, observando solo el 18, puede recuperar el secreto 6: ese es el problema del logaritmo discreto al que volveremos al final.

Paso 2 — Firmar un mensaje. El poseedor de la identidad quiere firmar el mensaje $m = 7$. Empieza eligiendo un valor aleatorio nuevo $k = 4$, que se usará una sola vez y no se compartirá jamás (en Ed25519 real, k se deriva determinísticamente del mensaje y del secreto para evitar el peligro de reutilizarlo, pero el papel que juega es exactamente este). Después calcula tres números:

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

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

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

La firma es el par **(r, s) = (16, 10)**. Viaja en abierto junto al mensaje. Cualquiera puede leerla. Nota didáctica: en Ed25519 real la función H es SHA-512, criptográficamente robusta; aquí usamos la simplificación $e = (r + m) \bmod q$ para que el lector pueda recorrer los pasos sin necesidad de calcular un hash. La estructura del algoritmo es la misma.

Paso 3 — Verificar la firma. El verificador tiene la parte pública $y = 18$, el mensaje $m = 7$, y la firma $(r, s) = (16, 10)$. Reconstruye e de la misma forma —$e = (16 + 7) \bmod 11 = 1$— y comprueba si esta igualdad se cumple:

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

Calcula los dos lados por separado:

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

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

Los dos lados dan **12**. La firma es válida. Cualquiera con la parte pública 18 puede llegar a esta conclusión sin haber sabido nunca que el secreto era 6.

¿Y un tercero que intentara falsificar? Eva ha visto pasar por el canal todo lo público: $p = 23$, $g = 2$, $q = 11$, $y = 18$, $m = 7$, $r = 16$, $s = 10$. Para firmar un mensaje *distinto* en nombre de esta identidad, necesitaría conocer x. Su única vía es preguntarse: «¿para qué exponente x se cumple $2^x \bmod 23 = 18$?». Con $p = 23$ puede probar 0, 1, 2, 3, … y encontrarlo en segundos. Pero al sustituir 23 por un primo de las dimensiones reales de Ed25519, el espacio de exponentes posibles supera el número de átomos del universo observable. **No existe a día de hoy ningún algoritmo conocido por la humanidad que pueda recorrer ese espacio en menos de miles de millones de años.** Es el mismo problema del logaritmo discreto que sustenta el Diffie-Hellman del artículo anterior, aplicado aquí al esquema de firma.

Esto que acabamos de recorrer es *exactamente* Schnorr, el esquema de firma del que Ed25519 es una variante adaptada a una curva elíptica. En Ed25519 real, todas las operaciones se hacen sobre los puntos de una curva concreta (Curve25519) en vez de sobre números enteros módulo un primo, y la función H es SHA-512 en vez de la suma de juguete que usamos arriba. Las dos sustituciones son ajustes de implementación —ganar resistencia criptográfica a la fuerza bruta, ganar propiedades adicionales de seguridad para k—. La estructura algorítmica, las tres operaciones, el porqué de la asimetría, son las mismas.

Conviene aquí un alto breve, porque la cadena entera puede confundirse de un vistazo rápido con otra primitiva del trío: el hash. No lo es. Un hash es una función única que comprime —entran muchos bytes, sale una huella corta, ahí acaba el camino—. Una identidad criptográfica es una pareja matemática complementaria: el secreto se queda y firma; su contraparte pública se publica y verifica. Donde el hash colapsa información en un sentido único, la identidad establece una asimetría entre dos mitades. El hash atestigua qué se dijo; la identidad atestigua quién lo dijo.

Lo que la frase no es

Tres equivocaciones frecuentes conviene despejar. La frase no es una contraseña en sentido propio: no se compara contra una huella almacenada en un servidor; se introduce en el dispositivo del usuario para reconstruir matemáticamente la identidad. La frase no se recupera: si se pierde, no hay nadie a quien pedírsela; si se duplica, se duplica también la identidad. La frase no es una credencial separable de la identidad: la frase *es* la identidad. Quien la tiene puede actuar como ella, sin permiso adicional, sin proceso de autorización, sin recuperación posible.

Esta tercera propiedad es la que cambia el peso del asunto. Una contraseña perdida es una molestia administrativa. Una identidad criptográfica perdida es la identidad. Un papel con la frase encontrado por terceros no es un riesgo de robo de cuenta: es la entrega de la identidad entera. La promesa del sistema —que nadie pueda revocarte tu identidad ni bloquearte arbitrariamente— viene acompañada inseparablemente de la responsabilidad — que tú eres el único custodio de algo que nadie puede restituir por ti.

La promesa y el peso

El modelo de identidad criptográfica suele recibir el calificativo de *autosoberano* —self-sovereign en la literatura anglosajona—. La elección de palabra es deliberada y describe con bastante exactitud la condición. El usuario es soberano sobre su identidad en un sentido casi medieval: no la concede ningún rey, ningún emisor, ninguna autoridad central; tampoco la pueden retirar ninguno de los anteriores. Pero también, como el monarca medieval, el usuario carga con la consecuencia entera de sus errores: no hay regente que tome decisiones en su lugar si pierde el sello.

La elección entre identidad gestionada por un tercero y identidad autosoberana no tiene una respuesta universal correcta. Para la cuenta de un foro irrelevante, la identidad gestionada es probablemente proporcional al riesgo. Para una identidad profesional que firma documentos jurídicamente vinculantes, para una identidad económica

que custodia ahorros propios, para una identidad de comunicación profesional con clientes que han confiado información sensible, la cuestión cambia. Allí la pregunta deja de ser «¿es cómodo?» y se vuelve «¿quién, además de yo, tiene el poder de actuar como yo, y bajo qué circunstancias?».

Dónde aparece este mecanismo en sistemas reales

El BIP39 nació en el mundo de Bitcoin en 2013 y se extendió rápidamente al ecosistema entero de criptomonedas: cualquier monedero serio acepta hoy una frase BIP39 de doce o veinticuatro palabras como respaldo de la identidad económica de su poseedor. Fuera de las criptomonedas, el mismo concepto subyacente —par criptográfico que prueba autoría sin intermediario— aparece en otros sistemas con sintaxis distinta. Las claves SSH que un administrador de sistemas usa para acceder a sus servidores son un caso clásico: una clave privada que el administrador guarda en su máquina y una pública que se copia en cada servidor; nadie comparable a un servicio centralizado interviene. El protocolo Signal usa Ed25519 con material de clave persistente en el dispositivo; las eIDAS europeas, en su parte de firma cualificada, descansan sobre el mismo principio criptográfico, con la diferencia de que la clave la custodia un proveedor de servicios de confianza cualificado en lugar del usuario.

Solo2, plataforma editora de esta publicación, usa una frase BIP39 de veinticuatro palabras como identidad de cada usuario. El usuario, al crear su cuenta, ve las palabras una vez. No se almacenan en ningún servidor de Solo2 ni de nadie: si el usuario las anota y las custodia, mantiene su identidad para siempre. Si las pierde, las pierde. Es la consecuencia coherente con la arquitectura de no haber operador en medio: si Solo2 pudiera devolverle la identidad al usuario que la perdió, podría también dárselo a quien presione a Solo2 para que se lo dé.

Para el lector profesional

Cuatro consideraciones para quien evalúa adoptar identidad criptográfica en un contexto profesional:

1. La frase es la identidad. Custodia física —papel, varias copias en lugares distintos, eventualmente metal grabado para uso de largo plazo— ofrece más garantías que la custodia digital, que añade superficie de ataque sin reducir el riesgo de pérdida.
2. No hay recuperación. Diseñar el proceso asumiendo que un día se pierde la copia primaria conviene mucho más que descubrirlo el día que se pierde. Una segunda copia geográficamente separada resuelve casi todos los escenarios.
3. No es lo mismo que un certificado cualificado eIDAS. Para firma cualificada en la Unión —escrituras notariales, ciertos trámites con la Administración— la legislación exige un proveedor cualificado que custodia la clave. La identidad criptográfica autosoberana sirve para la comunicación profesional y la firma documental con valor probatorio, pero no sustituye automáticamente al certificado cualificado en los casos donde la norma lo exige.
4. Si la identidad va a transferirse —herencia, sucesión profesional, cierre de actividad— conviene preparar el procedimiento antes, no después. Procedimientos formales con sobres lacrados, instrucciones a un albacea, depósito en notaría, son arreglos clásicos perfectamente compatibles con la naturaleza criptográfica del activo.

Este artículo cierra el trío conceptual que abrió el ciclo —hash, cifrado, identidad—. Las tres ideas se construyen unas sobre otras: el hash da la huella inalterable, el cifrado da la confidencialidad sin tercero de confianza, la identidad da la autoría sin tercero de concesión. Las tres comparten una propiedad que tampoco es ideológica: trasladan, de quien gestiona un servicio a quien lo usa, capacidades técnicas que tradicionalmente residían en el operador. Trasladan con ellas también responsabilidades. Hablar con honestidad de cualquiera de las tres exige hablar también de las otras dos.

Fuentes y lectura adicional

- Palatinus, M.; Rusnak, P.; Voisine, A.; Bowe, S. — *BIP-0039: Mnemonic code for generating deterministic keys*, propuesta de mejora de Bitcoin de 2013. Estándar de facto para frases de recuperación en la industria criptográfica.
- RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA), incluyendo Ed25519. IETF, enero de 2017. Especificación normativa del esquema de firma usado en buena parte de la industria contemporánea.
- RFC 2898 — PKCS #5: Password-Based Cryptography Specification, versión 2.0. IETF, septiembre de 2000. Define el algoritmo PBKDF2 usado en la derivación BIP39 de frase a semilla.
- Reglamento (UE) 910/2014 (eIDAS) y su evolución por el Reglamento (UE) 2024/1183 (eIDAS 2) — marco europeo de identidad electrónica y firma cualificada. Régimen distinto del autosoberano, pero conceptualmente apoyado en los mismos primitivos criptográficos.
- Allen, C. — *The Path to Self-Sovereign Identity* (2016). Texto canónico sobre los principios y compromisos del modelo autosoberano, anterior pero relevante para la comprensión de la familia de soluciones contemporáneas.

← AnteriorEl modelo de negocio como señal de confianzaSiguiente →Self-hosting como práctica profesional

Lecturas recientes

- Reflexión · 29 de junio de 2026 No eres anónimo
- Reflexión · 27 de mayo de 2026 Lo que una firma no puede arreglar
- Análisis · 26 de mayo de 2026 Privacidad real vs aparente: las preguntas que conviene hacerse

Llévate este artículo donde lo necesites.

↓ Markdown ↓ Texto plano ↓ PDF

El archivo se descarga a tu dispositivo. Desde ahí puedes guardarlo, importarlo a Solo2, o compartirlo donde quieras. Cuadernos no decide el destino por ti.

Sello de lacre · SHA-256 6c77df00c5626eff391cb4cf97f103734167621a345413b8e7e088def58f17d6