コンセプト・2026年5月23日

← 前へ·次へ →

24の言葉：暗号学的アイデンティティとは何か

暗号学的アイデンティティはパスワードではありません。いかなるサーバーもそれを保存せず、復元も不可能です。BIP39メカニズムの解説、なぜ正確に24単語なのか、そしてそれを保持する者に課せられる真の重みについて。

分かりやすく言えば：Gmailのパスワードを忘れた場合、Googleが再設定してくれます。しかし、暗号学的アイデンティティを構成する24の言葉を失った場合、それを頼める相手は誰もいません。手続きが厳格なのではなく、相手側に誰も存在しないのです。この違いこそが、決定的な違いです。

パスワードとアイデンティティの違い

従来のインターネットモデルにおけるパスワードは、ユーザーのアイデンティティそのものではありません。それは「証明書」です。ユーザーはアイデンティティ（名前、メールアドレス、顧客番号など）を持っており、自分が本人であることをサーバーに証明するために、パスワードを提示します。サーバーはそれを保存されているデータと比較し、一致すればセッションを許可します。パスワードを紛失しても、ユーザーはユーザーのままです。失うのは証明書だけであり、登録されたアドレスへのメールや秘密の質問といった復元手続きが存在します。

暗号学的アイデンティティは、それとは異なる仕組みで機能します。それは誰かが保存データと比較する資格情報ではなく、それ*自体*が完結した数学的な秘密です。それが紙に書かれていようと、デバイス内にあろうと、あるいは他人のサーバーにあろうと関係ありません。アイデンティティは数学によって存在するのであり、誰が検証するかによって存在するのではないのです。ここで、「SHA-256とは一体何なのか」で見たのと同様の特性が現れます。所有は秘密を提示することによってではなく、それを使って「署名」することによって証明されます。生成された署名は、秘密自体を知る必要なく、また第三者の仲介なしに、秘密から数学的に導き出された公開値を使用して誰でも検証できます。秘密を持つ者がアイデンティティであり、失う者はアイデンティティではなくなります。結論は断定的です。**アイデンティティを返してくれと頼める相手はどこにもいません。そんな人は存在しないのです。なぜなら、彼らは最初からそれを持っていなかったからです。**

24の言葉が表すもの

暗号学的アイデンティティは通常、32バイト（256ビット）の数学的秘密で表されます。これは覚えるのが難しく、間違いなく書き写すのはさらに困難な数字です。暗号業界は2013年、BIP39という小さくエレガントな標準によってこの問題を解決しました。これは、256ビットを2048単語の公式リストから選ばれた24の言葉のシーケンスとして表現する方法です。その背後にある算術は見事に合致しています。詳細を知りたい方は、補足説明をご覧ください。

計算は最後から始まります。秘密の256ビットに8ビットのチェックサムを加え、計264ビットを表現したいと考えます。これを、書き留めたり読み上げたりするのに扱いやすい24の言葉に分けると、1単語あたり正確に11ビットの情報を持つ必要があります。そして11ビットは2の11乗、つまり2048通りの可能性があります。だからこそ、公式のBIP39語彙リストはこのサイズなのです。リストが問題に合わせて存在するのであり、その逆ではありません。

この計算は単なる飾りではありません。もし誰かが23単語を正しく書き写し、24単語目で間違えた場合、チェックサムがそれを検出し、ソフトウェアは「このシーケンスは無効です」と告げます。24単語すべてを正しく書き写せば、ソフトウェアは曖昧さなく同じアイデンティティを導き出します。単語リストの選択も意図的です。BIP39の単語は短く、互いに明確に異なり、アクセント記号もなく、音声的・綴り上の混乱を最小限に抑えるよう選ばれています。これは、人間が間違いなく記憶し、書き、読み上げられるように設計された語彙なのです。

フレーズから鍵へ

24個の単語は、メッセージに署名する暗号鍵そのものではありません。これらは元のエントロピーを復元可能な形で表現したものであり、PBKDF2と呼ばれる決定論的なプロセスを通じて、64バイトのシード（種）に変換されます。そのシードから、同じく決定論的に、ユーザーが使用する具体的な暗号鍵が導出されます。署名用の秘密鍵と、署名を検証するために公開される対応する公開鍵です。異なるシステムでも同じメカニズムが使われています。暗号資産（仮想通貨）ではsecp256k1曲線が使われ、Signalプロトコルや多くの現代的なシステムではCurve25519曲線上のEd25519が使われています。Ed25519のような特定の曲線の場合、BIP32やSLIP-0010といった標準規格が、その64バイトのシードを受け取り、決定論的に有効な署名鍵を構成する32バイトを導出します。これは、次のセクションのコード例が始まるのと同じ32バイトです。

これは業界全体がユーザーにメカニズムを提示する標準的な方法です。暗号資産ウォレット、分散型アイデンティティ管理、恒久的なアイデンティティ部分におけるSignal、そしてSolo2もその一つです。実務上、ユーザーがシードや導出された鍵を直接目にすることはありません。ユーザーはアイデンティティを作成する際に24個の単語を確認し、オプションでそれを紙に書き留めます。その後、アイデンティティを移行したいときに、それらの単語はデバイス間を移動します。新しいアプリに単語を入力すると、アプリは同じシード、同じ鍵、同じアイデンティティを導出します。これはポータブルで暗号学的に堅牢であり、合理的な範囲で記憶可能なメカニズムです。

鍵で署名する方法（Zigによる一筆）

Zigでは、24個の単語から導出された32バイトのシードがあれば、Ed25519によるメッセージへの署名はわずか数行で記述できます。

```
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);
```

署名操作は「署名」と呼ばれる64バイトを生成します。これは対応する秘密鍵からしか生成できません。検証は公開されています。公開鍵を持つ人なら誰でも、署名がメッセージに対応しているかを確認できます。秘密鍵がなければ、そのメッセージに対して有効な署名を作成することは誰にもできません。公開鍵があれば、署名が有効かどうかを誰もが検知できます。この非対称性こそが、秘密を共有することなく、署名者が作成者であることを証明できる理由です。

前の例はマニュアルの最小構成版です。Solo2の実際のコードでは、このチェーンは2つのファイルを通過します。1つはユーザーのブラウザ上で動作し24語の単語からエントロピーを再構築するJavaScriptファイル、もう1つは *zcatcrypto* ライブラリ内のZigファイルで、そのエントロピーを受け取って具体的な暗号鍵を導出します。ブラウザ側から見ていきましょう：

```
// 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 };
}
```

その32バイトのエントロピーは、同じステップで導出された別の32バイトと共に、Ed25519鍵そのものを生成するZigのWebAssemblyモジュールへと送られます。最終的なメモリクリーニングを含む関数全体は、1つの画面に収まります：

```
// 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;
}
```

注目すべき点が2つあります。1つ目は、同じシード（seed）からは常に同じ鍵ペアが生成されるということです。まさにこれによって、新しいデバイスで24語の単語を入力することでアイデンティティを回復できるのです。2つ目は、シードが関数の最後の行でメモリから明示的に消去されることです。その時点を過ぎると、関数自体でさえ鍵を再構築することはできません。ユーザーの単語だけが唯一のソースとなります。

小さな数字で確認したい人のために。 署名スキームは、手計算ができるほど十分に小さな数字を使って全体を辿ることができます。算術に深入りしたくない人は、記事の文脈を失うことなくこのブロックを読み飛ばすことができます。メカニズムがステップバイステップでどのように機能するかを見たい人は、ここで確認できます。**公開ルール**（誰でも読むことができます）：素数 $p = 23$（実際のEd25519では約77桁ですが、計算を1ページに収めるために23を使用します）、このグループ内での位数が $q = 11$ であるベース $g = 2$、そして g を使ったすべての算術は *módulo* p で行われ、すべての指数は *módulo* q で簡約されるという規約です。**非公開の選択**（ただ1つであり、決して共有されません）：秘密 $x = 6$。これがアイデンティティです。

ステップ1 — アイデンティティの公開部分。 一度だけ計算され、公に公開されます。

$y = g^x \bmod p$

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

アイデンティティの公開部分は **18** です。誰でもこれを取得し、このアイデンティティで作られた署名を検証するために使用できます。18だけを見て秘密の6を復元できる人は誰もいません。これが、最後に改めて説明する離散対数問題です。

ステップ2 — メッセージに署名する。 アイデンティティの所有者はメッセージ $m = 7$ に署名したいと考えます。まず、一度だけ使用され、決して共有されない新しい乱数値 $k = 4$ を選択します（実際のEd25519では、再利用の危険を避けるために k はメッセージと秘密から決定論的に導出されますが、果たす役割はまさにこれです）。次に、3つの数値を計算します：

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

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

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

署名はペア **(r, s) = (16, 10)** です。メッセージと共に公開状態で送られます。誰でも読むことができます。教育的注意：実際のEd25519では関数 H は暗号的に堅牢なSHA-512ですが、ここでは読者がハッシュを計算せずに手順を辿れるように、簡略化した $e = (r + m) \bmod q$ を使用しています。アルゴリズムの構造は同じです。

ステップ3 — 署名を検証する。 検証者は公開部分 $y = 18$、メッセージ $m = 7$、および署名 $(r, s) = (16, 10)$ を持っています。同じ方法で e を再構築し（$e = (16 + 7) \bmod 11 = 1$）、この等式が成り立つかどうかを確認します：

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

両辺を個別に計算します：

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

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

両辺とも **12** になります。署名は有効です。公開部分の18を持っている人なら誰でも、秘密が6であったことを知らなくても、この結論に達することができます。

偽造を試みる第三者がいたらどうなるでしょうか？ エヴァは、チャネルを通過するすべての公開情報を見ました： $p = 23$, $g = 2$, $q = 11$, $y = 18$, $m = 7$, $r = 16$, $s = 10$。このアイデンティティを名乗って *別の* メッセージに署名するには、 x を知る必要があります。彼女の唯一の道は、「どの指数 x に対して $2^x \bmod 23 = 18$ が成り立つか？」と自問することです。 $p = 23$ であれば、0, 1, 2, 3, … と試していけば、

数秒で見つけることができます。しかし、23を実際のEd25519の規模の素数に置き換えると、可能な指数の空間は観測可能な宇宙の原子の数を超えます。**人類が知る限り、その空間を数十億年以内に走査できるアルゴリズムは今日存在しません。** これは前の記事のDiffie-Hellmanを支えているのと同じ離散対数問題であり、ここでは署名スキームに適用されています。

今辿ってきたものは *まさに* Schnorrであり、Ed25519はそのバリエーションを楕円曲線に適応させたものです。実際のEd25519では、すべての演算は素数モジュロの整数ではなく、特定の曲線（Curve25519）上の点に対して行われ、関数 H は上記で使用したおもちゃのような合計ではなくSHA-512になります。2つの置き換えは実装上の調整です（総当たり攻撃に対する暗号的な耐性の獲得、 k に対する追加のセキュリティ特性の獲得）。アルゴリズムの構造、3つの演算、そしてなぜ非対称なのかという理由は同じです。

ここで少し立ち止まる必要があります。チェーン全体をざっと見ただけでは、三羽烏の別のプリミティブであるハッシュと混同される可能性があるからです。ハッシュではありません。ハッシュは圧縮を行うユニークな関数です（多くのバイトが入力され、短い足跡が出力され、そこで道は終わります）。暗号アイデンティティは、数学的に補完し合うペアです。秘密は手元に残って署名し、それに対応する公開部分は公開されて検証されます。ハッシュが情報を一方向に崩壊させるのに対し、アイデンティティは2つの半分の間に非対称性を確立します。ハッシュは何が言われたかを証明し、アイデンティティは誰が言ったかを証明します。

フレーズではないもの

よくある3つの誤解を解いておく必要があります。第一に、フレーズは本来の意味でのパスワードではありません。サーバーに保存された指紋と比較されるのではなく、ユーザーのデバイスに入力されてアイデンティティを数学的に再構築するために使われます。第二に、フレーズは「復旧」できません。紛失しても、発行元に問い合わせることはできません。複製されれば、アイデンティティも複製されます。第三に、フレーズはアイデンティティから切り離せる認証情報ではありません。フレーズ*こそ*がアイデンティティなのです。それを持つ者は、追加の許可も、認証プロセスも、復旧の可能性もなく、そのアイデンティティとして振る舞うことができます。

この3番目の性質こそが、事の重みを変えるものです。紛失したパスワードは事務的な手間に過ぎません。しかし、紛失した暗号アイデンティティは、アイデンティティそのものの喪失です。フレーズが書かれた紙を第三者に見つかることは、アカウント盗難のリスクではなく、アイデンティティ全体の譲渡を意味します。「誰もあなたのアイデンティティを失効させたり、恣意的にブロックしたりできない」というシステムの約束は、「誰もあなたの代わりに復元できないものの唯一の保管者はあなたである」という責任と不可分なのです。

約束と重み

暗号アイデンティティのモデルは、しばしば*自己主権型*（英語ではself-sovereign）と呼ばれます。この言葉の選択は意図的なものであり、その状態を極めて正確に表しています。ユーザーは、ほぼ中世

的な意味で自分のアイデンティティの主権者です。いかなる王も、発行者も、中央当局もそれを授けることはなく、またそれを取り消すこともできません。しかし、中世の君主と同様に、ユーザーは自分の過ちの結果をすべて背負います。印章を紛失した際、代わりに決定を下してくれる摂政は存在しません。

第三者が管理するアイデンティティと自己主権型アイデンティティのどちらを選ぶべきか、という問いに唯一の正解はありません。重要でないフォーラムのアカウントであれば、管理されたアイデンティティはおそらくリスクに見合っています。しかし、法的に拘束力のある文書に署名する専門的なアイデンティティや、個人の貯蓄を守る経済的なアイデンティティ、機密情報を預けているクライアントとの専門的なコミュニケーションのアイデンティティとなれば、話は別です。そこでは「便利かどうか」という問いは消え、「自分の代わりに振る舞う権限を自分以外に誰が持っているのか、そしてそれはどのような状況においてか」という問いに変わります。

実際のシステムでこのメカニズムが使われている場所

BIP39は2013年にBitcoinの世界で誕生し、暗号通貨エコシステム全体に急速に広がりました。現在、信頼できるウォレットであれば、所有者の経済的アイデンティティのバックアップとして、12語または24語のBIP39フレーズを受け入れます。暗号通貨以外でも、仲介者なしで著作者を証明する暗号ペアという同様の基本概念は、異なる構文を持つ他のシステムにも現れています。システム管理者がサーバーにアクセスするために使用するSSHキーはその典型的な例です。管理者が自分のマシンに保管する秘密鍵と、各サーバーにコピーされる公開鍵があり、中央集権的なサービスに相当するものは介在しません。Signalプロトコルはデバイス上の永続的な鍵素材にEd25519を使用し、欧州のeIDAS（適格署名部分）も同様の暗号原理に基づいています。ただし、鍵はユーザーではなく適格信頼サービスプロバイダーによって保管されるという点が異なります。

本誌の出版プラットフォームであるSolo2は、各ユーザーのアイデンティティとして24語のBIP39フレーズを使用しています。ユーザーはアカウント作成時に一度だけその言葉を目にします。それらはSolo2のサーバーや他の誰のサーバーにも保存されません。ユーザーがそれらをメモして保管すれば、自身のアイデンティティを永遠に保持できます。紛失すれば、それまでです。これは中間にオペレーターが存在しないアーキテクチャの論理的な帰結です。もしSolo2が紛失したユーザーにアイデンティティを返せるとしたら、Solo2に圧力をかける誰かにそれを渡すこともできてしまうからです。

専門的な読者のために

専門的な文脈で自己主権型（autosoberana）アイデンティティの採用を検討している方への4つの考慮事項：

1. フレーズこそがアイデンティティです。物理的な保管（紙、異なる場所への複数のコピー、最終的には長期使用のための金属への刻印）は、紛失のリスクを減らさずに攻撃対象領域を増やすデジタル保管よりも、多くの保証を提供します。

2. リカバリは存在しません。いつかプライマリコピーを紛失することを前提にプロセスを設計することは、紛失した日にそれに気づくよりもはるかに賢明です。地理的に離れた場所に2つ目のコピーがあれば、ほとんどのシナリオを解決できます。
3. eIDAS適格証明書とは異なります。EU内での適格署名（公正証書、行政機関との特定の続きなど）については、法律により鍵を保管する適格プロバイダーが求められます。暗号による自己主権型アイデンティティは、職業上のコミュニケーションや証拠価値のある文書署名に役立ちますが、法律が義務付けている場合に適格証明書を自動的に代替するものではありません。
4. アイデンティティを譲渡する場合（相続、職業上の承継、活動の終了など）は、事後ではなく事前に手順を準備しておくのが賢明です。封蝋（lacre）された封筒による正式な手続き、遺言執行者への指示、公証人役場への預託などは、資産の暗号的性質と完全に互換性のある古典的な手配です。

この記事は、ハッシュ、暗号化、アイデンティティという、このサイクルを開始した概念のトリオを締めくくるものです。これら3つのアイデアは互いに積み重なっています。ハッシュは不変の指紋を与え、暗号化は信頼できる第三者なしで機密性を与え、アイデンティティは付与する第三者なしで著作者を与えます。これら3つは、イデオロギー的ではない共通の特性を共有しています。それは、伝統的にオペレーター側にあった技術的能力を、サービス管理者から利用者に移転させるということです。それとともに責任も移転されます。これら3つのいずれかについて誠実に語るには、他の2つについても語る必要があります。

参考文献および関連資料

- Palatinus, M.; Rusnak, P.; Voisine, A.; Bowe, S. — *BIP-0039: Mnemonic code for generating deterministic keys*, 2013年のBitcoin改善提案。暗号業界におけるリカバリフレーズの事実上の標準。
- RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA), Ed25519を含む。IETF, 2017年1月。現代の業界の大部分で使用されている署名スキームの規範的仕様。
- RFC 2898 — PKCS #5: Password-Based Cryptography Specification, バージョン 2.0。IETF, 2000年9月。フレーズからシードへのBIP39派生で使用されるPBKDF2アルゴリズムを定義。
- 規則 (EU) 910/2014 (eIDAS) および規則 (EU) 2024/1183 (eIDAS 2) によるその進化 — 電子アイデンティティおよび適格署名のための欧州の枠組み。自己主権型とは異なる制度だが、概念的には同じ暗号プリミティブに支えられている。
- Allen, C. — *The Path to Self-Sovereign Identity* (2016)。自己主権型モデルの原則とコミットメントに関する標準的なテキスト。以前のものだが、現代のソリューション群を理解する上で重要。

← 前へ信頼のシグナルとしてのビジネスモデル次へ →専門的実践としてのセルフホスティング

最近の記事

- 考察・2026年6月29日 あなたは匿名ではない

- 考察・2026年5月27日 署名では解決できないこと
- 分析・2026年5月26日 真のプライバシー vs 表向きのプライバシー：問い直すべきこと

この記事をダウンロードして、必要な場所で活用してください。

↓ Markdown ↓ テキスト形式 ↓ PDF

ファイルはお使いのデバイスにダウンロードされます。そこから保存、Solo2 へのインポート、または任意の場所での共有が可能です。Cuadernos が送信先を決定することはありません。

封蝋・SHA-256 2c10b701fa22a10a6541bc936a1e4d94ed8e7e97537892e00db1cc03c9ba15dd

Cuadernos Lacre・Menzuri Gestión S.L. による刊行物・
著者：R.Eugenio・編集：Solo2 チーム