개념 · 2026년 5월 23일

← 이전 · 다음 →

24개의 단어: 암호학적 정체성이란 무엇인가

암호학적 정체성은 비밀번호가 아닙니다. 어떤 서버도 이를 저장하지 않으며 복구할 수도 없습니다. BIP39 메커니즘에 대한 교육적 설명, 왜 정확히 24개의 단어인지, 그리고 이를 소유한 사람에게 주어지는 실제적인 책임에 대해 알아봅니다.

쉽게 말해서: Gmail 비밀번호를 잊어버리면 Google이 재설정해 줍니다. 하지만 암호학적 정체성을 구성하는 24개의 단어를 잃어버리면 이를 요청할 대상이 아무도 없습니다. 절차가 엄격해서가 아니라, 반대편에 아무도 존재하지 않기 때문입니다. 이 차이가 모든 것을 결정합니다.

비밀번호와 정체성의 차이

기존 인터넷 모델에서 비밀번호는 사용자의 정체성 자체가 아닙니다. 그것은 일종의 '증명서'입니다. 사용자는 정체성(이름, 이메일, 고객 번호 등)을 가지고 있으며, 자신이 주장하는 본인임을 서버에 증명하기 위해 서버가 저장하고 있던 기록과 대조할 비밀번호를 제시합니다. 기록이 일치하면 서버는 세션을 허용합니다. 비밀번호를 잃어버려도 사용자는 여전히 같은 사용자입니다. 잃어버린 것은 증명서뿐이며, 등록된 주소로의 이메일이나 보안 질문과 같은 복구 절차를 통해 이를 되찾을 수 있습니다.

암호학적 정체성은 다른 방식으로 작동합니다. 이것은 누군가가 저장된 기록과 대조하는 자격 증명이 아니라, 그 *자체*로 완결된 수학적 비밀입니다. 그것이 종이 위에 있든, 기기 안에 있든, 심지어 타인의 서버에 있든 상관없습니다. 정체성은 누가 검증하느냐가 아니라 수학적 원리에 의해 존재합니다. 여기서 「SHA-256이란 과연 무엇인가」에서 보았던 것과 유사한 속성이 나타납니다. 소유권은 비밀을 보여줌으로써 증명되는 것이 아니라, 그것을 사용하여 '서명'함으로써 증명됩니다. 이렇게 생성된 서명은 비밀 자체를 알 필요 없이, 또한 제3자의 개입 없이, 비밀에서 수학적으로 유도된 공개 값을 사용하여 누구나 검증할 수 있습니다. 비밀을 가진 자가 곧 정체성이며, 이를 잃어버린 자는 더 이상 정체성이 아닙니다. 결론은 단호합니다. **정체성을 돌려달라고 요청할 대상은 어디에도 없습니다. 그런 존재는 없습니다. 왜냐하면 그들은 처음부터 그것을 가지고 있지 않았기 때문입니다.**

24개의 단어가 나타내는 것

암호학적 정체성은 보통 32바이트(256비트)의 수학적 비밀로 표현됩니다. 이는 기억하기 어렵고 실수 없이 옮겨 적기는 더 어려운 숫자입니다. 암호화 산업은 2013년에 BIP39라는 작고 우아한 표준을 통해 이 문제를 해결

했습니다. 256비트를 2048개의 공식 단어 목록에서 선택된 24개의 단어 시퀀스로 표현하는 방식입니다. 그 이면의 산술은 정교하게 맞아떨어집니다. 자세한 내용을 보려면 주석을 참고하십시오.

계산은 끝에서부터 시작됩니다. 비밀의 256비트에 8비트의 체크섬을 더해 총 264비트를 표현하고자 합니다. 이를 받아 적거나 불러주기 쉬운 24개의 단어로 나누면, 각 단어는 정확히 11비트의 정보를 담아야 합니다. 11비트는 2의 11제곱, 즉 2048가지의 가능성을 의미합니다. 그래서 공식 BIP39 어휘 목록이 정확히 그 크기인 것입니다. 목록이 문제에 맞춰 존재하는 것이지, 그 반대가 아닙니다.

이 계산은 단순한 장식이 아닙니다. 누군가 23개의 단어를 정확히 옮겨 적고 24번째 단어에서 실수한다면 체크섬이 이를 감지할 것입니다. 소프트웨어는 "이 시퀀스는 유효하지 않습니다"라고 알려줄 것입니다. 24개 단어를 모두 정확히 옮겨 적으면 소프트웨어는 모호함 없이 동일한 정체성을 도출해낼 것입니다. 단어 목록의 선택 또한 의도적입니다. BIP39 어휘들은 짧고, 서로 확연히 다르며, 발음 기호가 없고, 음성적 및 철자적 혼동을 최소화하도록 선택되었습니다. 인간이 실수 없이 기억하고, 쓰고, 불러줄 수 있도록 설계된 어휘입니다.

문구에서 키로

24개의 단어는 메시지에 서명하는 암호 키 자체가 아닙니다. 이는 PBKDF2라는 결정론적 프로세스를 통해 64바이트 시드(seed)로 변환되는 원래 엔트로피의 복구 가능한 표현입니다. 해당 시드로부터 사용자가 사용하는 구체적인 암호 키가 결정론적으로 파생됩니다. 즉, 서명을 위한 개인 키와 서명을 검증하기 위해 공개되는 해당 공개 키입니다. 서로 다른 시스템에서도 동일한 메커니즘이 사용됩니다. 암호화폐는 secp256k1 곡선을 사용하며, Signal 프로토콜과 많은 현대 시스템은 Curve25519 곡선 위의 Ed25519를 사용합니다. Ed25519와 같은 특정 곡선의 경우, BIP32 및 SLIP-0010 표준은 해당 64바이트 시드를 받아 결정론적으로 유효한 서명 키를 구성하는 32바이트를 파생시킵니다. 이는 다음 섹션의 코드 예제가 시작되는 것과 동일한 32바이트입니다.

이것은 암호화폐 지갑, 탈중앙화 신원 관리자, 영구 신원 부분에서의 Signal, 그리고 Solo2를 포함하여 업계 전체가 사용자에게 메커니즘을 제시하는 표준 방식입니다. 실무적으로 사용자는 시드나 파생 키를 직접 보지 않습니다. 사용자는 신원을 생성할 때 24개의 단어를 확인하고, 선택적으로 이를 종이에 적어 둡니다. 그 후 신원을 이전하고 싶을 때 단어들은 장치 간에 이동합니다. 새로운 애플리케이션에 단어를 입력하면 애플리케이션은 동일한 시드, 동일한 키, 동일한 신원을 파생시킵니다. 이것은 휴대가 가능하고 암호학적으로 견고하며 합리적인 범위 내에서 기억할 수 있는 메커니즘입니다.

키로 서명하는 방법(Zig를 통한 한 획)

Zig에서 24개의 단어로부터 파생된 32바이트 시드가 있으면 Ed25519를 사용한 메시지 서명은 단 몇 줄로 가능합니다.

```
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);
```

서명 작업은 '서명'이라 불리는 64바이트를 생성하며, 이는 해당 개인 키에서만 생성될 수 있습니다. 검증은 공개적입니다. 공개 키를 가진 사람은 누구나 서명이 메시지에 해당하는지 확인할 수 있습니다. 개인 키 없이는 누구도 해당 메시지에 대해 유효한 서명을 생성할 수 없으며, 공개 키가 있으면 서명이 유효한지 누구나 감지할 수 있습니다. 이러한 비대칭성 덕분에 서명자는 비밀을 공유하지 않고도 작성자임을 증명할 수 있습니다.

이전 예제는 설명서의 최소 버전입니다. Solo2의 실제 코드에서 이 체인은 두 개의 파일을 통과합니다. 하나는 사용자의 브라우저에서 실행되며 24개의 단어로부터 엔트로피를 재구성하는 JavaScript 파일이고, 다른 하나는 그 엔트로피를 받아 구체적인 암호화 키를 도출하는 *zcatcrypto* 라이브러리 내의 Zig 파일입니다. 브라우저 쪽부터 시작해보겠습니다.

```
// 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 };
}
```

해당 32바이트의 엔트로피는 동일한 단계에서 도출된 다른 32바이트와 함께 Ed25519 키 자체를 생성하는 Zig의 WebAssembly 모듈로 이동합니다. 최종 메모리 정리 기능이 포함된 전체 함수는 한 화면에 들어옵니다.

```
// 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;
}
```

주목할 만한 두 가지 세부 사항이 있습니다. 첫째: 동일한 시드(seed)는 항상 동일한 키 쌍을 생성합니다 — 바로 이것이 새 장치에서 24개의 단어를 입력하여 아이덴티티를 복구할 수 있게 해주는 것입니다. 둘째: 시드는 마지막 줄에서 메모리로부터 명시적으로 삭제됩니다. 그 시점을 지나면 함수 자체도 키를 재구성할 수 없으며, 사용자의 단어만이 유일한 소스가 됩니다.

작은 숫자로 확인하고 싶은 분들을 위해. 서명 방식은 손으로 계산할 수 있을 만큼 충분히 작은 숫자로 전체 과정을 훑어볼 수 있습니다. 산술 연산에 깊이 관여하고 싶지 않은 분들은 기사의 흐름을 놓치지 않고 이 블록을 건너뛸 수 있습니다. 메커니즘이 단계별로 어떻게 작동하는지 보고 싶은 분들은 여기에서 확인할 수 있습니다. 누구나 읽을 수 있는 **공개 규칙**: 소수 $p = 23$ (실제 Ed25519에서는 약 77자리 숫자이지만, 계산을 한 페이지에 담기 위해 23을 사용합니다), 이 그룹 내의 차수가 $q = 11$인 베이스 $g = 2$, 그리고 g를 사용한 모든 산술 연산은 *módulo p*로 수행되고 모든 지수는 *módulo q*로 축소된다는 관례입니다. **비공개 선택**, 단 하나이며 절대 공유되지 않는 것: 비밀 $x = 6$. 이것이 아이덴티티입니다.

1단계 — 아이덴티티의 공개 부분. 한 번 계산되어 공개적으로 게시됩니다.

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

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

아이덴티티의 공개 부분은 **18**입니다. 누구든지 이를 가져와 이 아이덴티티로 만든 서명을 확인하는 데 사용할 수 있습니다. 18만 보고 비밀 6을 복구할 수 있는 사람은 아무도 없습니다. 이것이 마지막에 다시 다룰 이산 로그 문제입니다.

2단계 — 메시지 서명. 아이덴티티 소유자는 메시지 $m = 7$에 서명하고자 합니다. 먼저 한 번만 사용되고 절대 공유되지 않을 새로운 난수 값 $k = 4$를 선택하는 것으로 시작합니다(실제 Ed25519에서 k는 재사용의 위험을 피하기 위해 메시지와 비밀로부터 결정론적으로 도출되지만, 그 역할은 바로 이것입니다). 그런 다음 세 개의 숫자를 계산합니다.

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

e = H(r, m) mod q = (16 + 7) mod 11 = 1

s = (k + x·e) mod q = (4 + 6·1) mod 11 = 10

서명은 쌍 **(r, s) = (16, 10)**입니다. 메시지와 함께 공개적으로 이동합니다. 누구든지 읽을 수 있습니다. 교육적 참고: 실제 Ed25519에서 함수 *H*는 암호학적으로 견고한 SHA-512입니다. 여기서는 독자가 해시를 계산할 필요 없이 단계를 따라갈 수 있도록 *e = (r + m) mod q*라는 단순화된 형태를 사용합니다. 알고리즘의 구조는 동일합니다.

3단계 — 서명 확인. 확인자는 공개 부분 *y* = 18, 메시지 *m* = 7, 서명 (*r*, *s*) = (16, 10)을 가지고 있습니다. 동일한 방식으로 *e*를 재구성하고 — *e* = (16 + 7) mod 11 = 1 — 이 등식이 성립하는지 확인합니다.

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

양쪽을 별도로 계산합니다.

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

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

양쪽 모두 **12**가 나옵니다. 서명이 유효합니다. 공개 부분 18을 가진 사람은 비밀이 6이었다는 사실을 전혀 모른 채 이 결론에 도달할 수 있습니다.

위조를 시도하는 제3자가 있다면 어떨까요? 에바는 채널을 통해 전달되는 모든 공개 정보를 보았습니다: *p* = 23, *g* = 2, *q* = 11, *y* = 18, *m* = 7, *r* = 16, *s* = 10. 이 아이덴티티의 이름으로 *다른* 메시지에 서명하려면 에바는 *x*를 알아야 합니다. 에바의 유일한 방법은 "어떤 지수 *x*에 대해 $2^x \bmod 23 = 18$이 성립하는가?"라고 자문하는 것입니다. *p* = 23이라면 에바는 0, 1, 2, 3, …을 시도하여 몇 초 만에 찾을 수 있습니다. 하지만 23을 실제 Ed25519 규모의 소수로 대체하면 가능한 지수의 공간은 관측 가능한 우주의 원자 수를 초과합니다. **오늘날 인류에게 알려진 알고리즘 중 그 공간을 수십억 년 이내에 훑어볼 수 있는 것은 존재하지 않습니다.** 이것은 이전 기사의 Diffie-Hellman을 뒷받침하는 것과 동일한 이산 로그 문제이며, 여기서는 서명 방식에 적용되었습니다.

우리가 방금 살펴본 것은 *정확히* Schnorr이며, Ed25519는 이를 타원 곡선에 맞게 조정한 변형입니다. 실제 Ed25519에서 모든 연산은 소수 모듈로 정수가 아닌 특정 곡선(Curve25519) 위의 점에 대해 수행되며, 함수 *H*는 위에서 사용한 장난감 같은 합계 대신 SHA-512가 됩니다. 두 가지 대체는 구현상의 조정입니다 — 무차별 대입 공격에 대한 암호학적 저항력을 얻고, *k*에 대한 추가적인 보안 속성을 얻기 위함입니다. 알고리즘 구조, 세 가지 연산, 비대칭성의 이유는 동일합니다.

여기서 잠시 멈추는 것이 적절합니다. 전체 체인이 언뜻 보기에 세 가지 원시 함수 중 하나인 해시와 혼동될 수 있기 때문입니다. 해시가 아닙니다. 해시는 압축을 수행하는 고유한 함수입니다 — 많은 바이트가 들어가고 짧은 자국이 나오며 거기서 길이 끝납니다. 암호학적 아이덴티티는 수학적으로 상호 보완적인 한 쌍입니다. 비밀은 보관되어 서명하고, 그에 대응하는 공개 부분은 게시되어 확인됩니다. 해시가 정보를 한 방향으로 축소하는 반면, 아이덴티티는 두 부분 사이에 비대칭성을 확립합니다. 해시는 무엇이 말해졌는지를 입증하고, 아이덴티티는 누가 그것을 말했는지를 입증합니다.

문구가 아닌 것

세 가지 흔한 오해를 바로잡을 필요가 있습니다. 첫째, 문구는 본래 의미의 비밀번호가 아닙니다. 서버에 저장된 지문과 비교되는 것이 아니라, 신원을 수학적으로 재구성하기 위해 사용자의 장치에 입력됩니다. 둘째, 문구는 복구되지 않습니다. 분실하면 누구에게도 요청할 수 없으며, 복제되면 신원도 복제됩니다. 셋째, 문구는 신원과 분리될 수 있는 인증 정보가 아닙니다. 문구 *자체*가 신원입니다. 이를 가진 사람은 추가 권한, 승인 프로세스, 복구 가능성 없이도 해당 신원으로서 행동할 수 있습니다.

이 세 번째 속성이 사안의 무게를 바꾸는 것입니다. 분실된 비밀번호는 행정적인 번거로움에 불과합니다. 그러나 분실된 암호화 신원은 신원 그 자체입니다. 제3자가 문구가 적힌 종이를 발견하는 것은 계정 도용의 위험이 아니라 신원 전체를 양도하는 것입니다. 누구도 귀하의 신원을 취소하거나 임의로 차단할 수 없다는 시스템의 약속은 귀하만이 귀하를 위해 누구도 복원할 수 없는 무언가의 유일한 관리자라는 책임과 불가분하게 연결되어 있습니다.

약속과 무게

암호화 신원 모델은 종종 *자기 주권*(영어로는 self-sovereign)이라는 수식어를 받습니다. 이 단어의 선택은 의도적이며 상태를 매우 정확하게 설명합니다. 사용자는 거의 중세적인 의미에서 자신의 신원에 대한 주권자입니다. 어떤 왕도, 발행자도, 중앙 당국도 이를 부여하지 않으며, 위 중 누구도 이를 회수할 수 없습니다. 그러나 중세 군주와 마찬가지로 사용자도 자신의 실수에 대한 온전한 책임을 집니다. 인장을 분실했을 때 대신 결정을 내려줄 섭정은 존재하지 않습니다.

제3자가 관리하는 신원과 자기 주권 신원 사이의 선택에는 유일한 보편적 정답이 없습니다. 중요하지 않은 포럼 계정의 경우 관리형 신원이 아마도 위험에 비례할 것입니다. 그러나 법적으로 구속력이 있는 문서에 서명하는 전문적인 신원, 자신의 저축을 지키는 경제적인 신원, 민감한 정보를 맡긴 고객과의 전문적인 커뮤니케이션 신원의 경우 사안이 달라집니다. 거기서 질문은 '편리한가?'가 아니라 '나 이외에 누가 나로서 행동할 권한을 가지고 있으며, 어떤 상황에서 그러한가?'로 바뀝니다.

실제 시스템에서 이 메커니즘이 나타나는 곳

BIP39은 2013년 Bitcoin 세계에서 탄생하여 암호화폐 생태계 전체로 빠르게 확산되었습니다. 오늘날 신뢰할 수 있는 모든 지갑은 소유자의 경제적 정체성에 대한 백업으로 12단어 또는 24단어 BIP39 문구를 수용합니다. 암호화폐 외에도 중개자 없이 저작권을 증명하는 암호화 쌍이라는 동일한 기본 개념은 구문이 다른 다른 시스템에서도 나타납니다. 시스템 관리자가 서버에 액세스하는 데 사용하는 SSH 키가 전형적인 사례입니다. 관리자가 자신의 머신에 보관하는 개인 키와 각 서버에 복사되는 공개 키가 있으며, 중앙 집중식 서비스에 필적하는 어떤 엔티티도 개입하지 않습니다. Signal 프로토콜은 기기에 영구적인 키 자료와 함께 Ed25519를 사용하며, 유럽의 eIDAS(적격 서명 부분)도 동일한 암호화 원칙에 기반하고 있습니다. 단, 키는 사용자가 아닌 적격 신뢰 서비스 제공자가 보관한다는 점이 다릅니다.

본 간행물의 출판 플랫폼인 Solo2는 각 사용자의 정체성으로 24단어 BIP39 문구를 사용합니다. 사용자는 계정을 생성할 때 단어들을 한 번 보게 됩니다. 이 단어들은 Solo2나 다른 누구의 서버에도 저장되지 않습니다. 사

용자가 이를 기록하고 보관하면 자신의 정체성을 영원히 유지할 수 있습니다. 이를 잃어버리면 모든 것을 잃게 됩니다. 이는 중간에 운영자가 없는 아키텍처의 일관된 결과입니다. 만약 Solo2가 정체성을 잃어버린 사용자에게 이를 되돌려 줄 수 있다면, Solo2에 압력을 가하는 누구에게나 이를 넘겨줄 수도 있기 때문입니다.

전문적인 독자를 위해

전문적인 맥락에서 자기주권형(autosoberana) 암호화 정체성 채택을 검토하는 분들을 위한 4가지 고려 사항:

1. 문구가 곧 정체성입니다. 물리적 보관(종이, 여러 장소에 사본 보관, 궁극적으로 장기 사용을 위한 금속 각인)은 분실 위험을 줄이지 않으면서 공격 표면을 늘리는 디지털 보관보다 더 많은 보장을 제공합니다.
2. 복구는 존재하지 않습니다. 언젠가 기본 사본을 잃어버릴 것이라고 가정하고 프로세스를 설계하는 것이 잃어버린 당일에 이를 발견하는 것보다 훨씬 현명합니다. 지리적으로 분리된 두 번째 사본이 있으면 거의 모든 시나리오를 해결할 수 있습니다.
3. eIDAS 적격 인증서와는 다릅니다. EU 내에서의 적격 서명(공정 증서, 행정 기관과의 특정 절차 등)의 경우, 법률에 따라 키를 보관하는 적격 제공자가 요구됩니다. 암호화 자기주권형 정체성은 전문적인 커뮤니케이션 및 증거 가치가 있는 문서 서명에 유용하지만, 법률이 요구하는 경우 적격 인증서를 자동으로 대체하지는 않습니다.
4. 정체성을 이전해야 하는 경우(상속, 직업적 승계, 활동 종료 등)에는 사후가 아닌 사전에 절차를 준비하는 것이 좋습니다. 봉인 왁스(lacre)로 봉인된 봉투를 이용한 공식 절차, 유언 집행자에 대한 지침, 공증인 사무소 예치 등은 자산의 암호화 특성과 완벽하게 호환되는 전통적인 방식입니다.

이 기사는 해시, 암호화, 정체성이라는 본 사이클을 시작한 세 가지 개념의 대미를 장식합니다. 이 세 가지 아이디어는 서로의 위에 쌓여 있습니다. 해시는 변경 불가능한 지문을 제공하고, 암호화는 신뢰할 수 있는 제3자 없이 기밀성을 제공하며, 정체성은 부여하는 제3자 없이 저작권을 제공합니다. 이 세 가지는 이데올로기적이지 않은 공통된 속성을 공유합니다. 즉, 전통적으로 운영자 측에 있던 기술적 역량을 서비스 관리자로부터 이용자에게 이전한다는 것입니다. 그와 함께 책임도 이전됩니다. 이 세 가지 중 어느 하나에 대해 정직하게 이야기하려면 나머지 두 가지에 대해서도 이야기해야 합니다.

참고 문헌 및 관련 자료

- Palatinus, M.; Rusnak, P.; Voisine, A.; Bowe, S. — *BIP-0039: Mnemonic code for generating deterministic keys*, 2013년 Bitcoin 개선 제안. 암호화 업계의 복구 문구에 대한 사실상의 표준.
- RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA), Ed25519 포함. IETF, 2017년 1월. 현대 업계의 상당 부분에서 사용되는 서명 방식의 규범적 사양.
- RFC 2898 — PKCS #5: Password-Based Cryptography Specification, 버전 2.0. IETF, 2000년 9월. 문구에서 시드(seed)로의 BIP39 파생에 사용되는 PBKDF2 알고리즘을 정의.
- 규정 (EU) 910/2014 (eIDAS) 및 규정 (EU) 2024/1183 (eIDAS 2)에 의한 발전 — 전자 정체성 및 적격 서명을 위한 유럽 프레임워크. 자기주권형과는 다른 제도이지만 개념적으로 동일한 암호화 프리미티브에 의해 뒷받침됨.
- Allen, C. — *The Path to Self-Sovereign Identity* (2016). 자기주권형 모델의 원칙과 약속에 관한 표준 텍스트. 이전 것이지만 현대 솔루션 군을 이해하는 데 유효함.

← 이전신뢰의 신호로서의 비즈니스 모델다음 →전문적 실무로서의 셀프 호스팅

최근 읽은 글

- 성찰 · 2026년 6월 29일 당신은 익명이 아닙니다
- 성찰 · 2026년 5월 27일 서명으로 해결할 수 없는 것
- 분석 · 2026년 5월 26일 진정한 개인정보 보호 vs 겉모습뿐인 보호: 스스로 던져야 할 질문

이 기사를 다운로드하여 필요한 곳에서 활용하십시오.

↓ 마크다운 ↓ 텍스트 형식 ↓ PDF

파일이 기기에 다운로드됩니다. 해당 위치에서 저장하거나 Solo2로 가져오거나 원하는 곳에 공유할 수 있습니다. Cuadernos는 전송 대상을 결정하지 않습니다.

봉인 · SHA-256 5a8dc2b3d6f36a6591c76f12ce94d02b845c33ca42a019446b5e5f307f95cec3

Cuadernos Lacre · Menzuri Gestión S.L.의 간행물 ·
저자: R.Eugenio · Solo2 팀 편집