الكلمات الـ 24: ما هي الهوية التشفيرية

الهوية التشفيرية ليست كلمة مرور: لا يحفظها أي خادم ولا يمكن استعادتها. شرح تعليمي لآلية BIP39، ولماذا بالضبط أربع وعشرون كلمة، وما هو الثقل الحقيقي الذي يقع على عاتق من يملكها.

لنفهم الأمر: إذا نسيت كلمة مرور Gmail، ستقوم جوجل بإعادة تعيينها لك. أما إذا فقدت الكلمات الـ 24 التي تشكل هوية تشفيرية، فلا يوجد أحد تطلبها منه. ليس لأن الإجراء صارم، بل لأنه لا يوجد أحد على الطرف الآخر. هذا الفرق هو كل الفرق.

الفرق بين كلمة المرور والهوية

كلمة المرور، في نموذج الإنترنت الكلاسيكي، ليست هوية المستخدم. بل هي قسيمة إثبات. يملك المستخدم هوية —اسماً، بريداً إلكترونياً، رقماً للعميل— ولإثبات هويته أمام الخادم، يقدم كلمة مرور يقارنها الخادم ببصمة مخزنة لديه. إذا تطابقت البصمات، يمنح الخادم الجلسة. وإذا فقدت كلمة المرور، يظل المستخدم هو نفس المستخدم؛ ما يفقده هو القسيمة، وهناك إجراء للاستعادة —رسالة إلى العنوان المسجل، سؤال أمان— لإعادتها.

تعمل الهوية التشفيرية بطريقة أخرى. فهي ليست اعتماداً يقارنه شخص ما ببصمة مخزنة؛ بل هي سر رياضي كامل في حد ذاته. لا يهم أين يوجد —على ورقة، في جهاز، أو حتى في خادم خارجي—: الهوية موجودة برياضياتها، وليس بمن يصادق عليها. تظهر هنا خاصية مشابهة لتلك التي رأيناها في «ما هو SHA-256 حقاً»: الحيازة لا تُثبت بعرض السر، بل باستخدامه للتوقيع. التوقيع الناتج يمكن لأي شخص التحقق منه بقيمة عامة مشتقة رياضياً من السر نفسه، دون الحاجة لمعرفة السر، ودون تدخل طرف ثالث. من يملك السر، هو الهوية؛ ومن يفقده، يتوقف عن كونها. الحكم قاطع: **لا يوجد أحد تطلب منه إعادة الهوية إليك. هذا الشخص غير موجود، لأنه لم يكن يملكها في المقام الأول.**

ما تمثله أربع وعشرون كلمة

تتمثل الهوية التشفيرية عادةً بسر رياضي من اثنين وثلاثين بايت —مائتان وستة وخمسون بت—. وهو رقم يصعب حفظه وأصعب في نسخه دون خطأ. حلت صناعة التشفير هذه المشكلة في عام 2013 بمعيار صغير وأنيق يسمى BIP39: طريقة لتمثيل تلك المائتين وستة وخمسين بت كسلسلة من أربع وعشرين كلمة مأخوذة من قائمة رسمية تضم ألفين وثمان وأربعين كلمة. الحسابات الكامنة وراء ذلك تتطابق بأناقة؛ ومن يريد رؤيتها بالتفصيل سيجدها في الهامش.

يبدأ العد من النهاية. نريد تمثيل مائتين وستة وخمسين بت من السر مع إضافة ثمانية بتات كجمع تدقيقي: مائتان وأربعة وستون بت في المجموع. إذا وزعناها على أربع وعشرين كلمة —وهو رقم يسهل تدوينه وإملاؤه دون فقدان—، يجب أن توفر كل كلمة بالضبط أحد عشر بت من المعلومات. وأحد عشر بت تعني اثنين مرفوعة للقوة أحد عشر من الاحتمالات، أي ألفين وثمان وأربعين. ومن هنا جاء حجم مفردات BIP39 الرسمية بهذا الحجم: القائمة وُجدت لتناسب المشكلة، وليس العكس.

العد ليس تجميلياً. إذا نسخ أحدهم ثلاثاً وعشرين كلمة بشكل صحيح وأخطأ في الرابعة والعشرين، سيكشف الجمع التدقيقي ذلك: سيخبره البرنامج «هذا التسلسل غير صالح». وإذا نسخ الأربعة والعشرين كلمة صحيحة، سيشتق البرنامج نفس الهوية دون غموض. اختيار قائمة الكلمات مدروس أيضاً: كلمات مفردات BIP39 قصيرة، متميزة عن

بعضها البعض، بدون علامات تشكيل، ومختارة لتقليل الالتباس الصوتي والإملائي. إنه قاموس مصمم ليتم تذكره وكتابته وإملاؤه من قبل البشر دون فقدان.

من العبارة إلى المفتاح

الكلمات الأربع والعشرون ليست المفتاح التشفيري الذي يوقع الرسائل. إنها تمثيل قابل للاسترداد للإنتروبيا الأصلية التي، من خلال عملية حتمية تسمى PBKDF2، تتحول إلى بذرة مكونة من أربعة وستين بايت. من تلك البذرة تشتق، أيضاً بشكل حتمي، المفاتيح التشفيرية المحددة التي يستخدمها المستخدم: مفتاح خاص للتوقيع ومفتاح عام مقابل يتم نشره للتحقق من التوقيعات. نفس الآلية في أنظمة مختلفة: تستخدم العملات المشفرة منحنى secp256k1؛ بينما يستخدم بروتوكول Signal والعديد من الأنظمة الحديثة Ed25519 على منحنى Curve25519. بالنسبة لمنحنى معين مثل Ed25519، تأخذ معايير BIP32 و SLIP-0010 تلك البذرة المكونة من أربعة وستين بايت وتشتق، بشكل حتمي، الاثنين وثلاثين بايت التي تشكل مفتاح التوقيع الفعلي — نفس الاثنين وثلاثين بايت التي يبدأ بها مثال الكود في القسم التالي.

هذه هي الطريقة القياسية التي تعرض بها الصناعة بأكملها الآلية للمستخدم —محافظ العملات المشفرة، ومديرو الهوية اللامركزية، و Signal في جزء الهوية الدائمة، و Solo2 من بينها—: المستخدم، في الممارسة العملية، لا يرى البذرة ولا المفاتيح المشتقة أبداً. يرى الكلمات الأربع والعشرين عند إنشاء هويته، واختيارياً، يدونها على ورقة. تنتقل الكلمات بعد ذلك بين أجهزته عندما يريد نقل الهوية: يدخلها في التطبيق الجديد، فيشتق التطبيق نفس البذرة، ونفس المفاتيح، ونفس الهوية. إنها آلية محمولة، ومتينة تشفيرياً، وقابلة للتذكر ضمن حدود المعقول.

كيفية التوقيع بالمفتاح (لمسة Zig)

في Zig، بمجرد حصولك على البذرة المكونة من اثنين وثلاثين بايت والمشتقة من الكلمات الأربع والعشرين، فإن توقيع رسالة باستخدام Ed25519 يتناسب مع أسطر قليلة:

```
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);
```

تنتج عملية التوقيع أربعة وستين بايت —تسمى توقيعاً— لا يمكن إنتاجها إلا من المفتاح الخاص المقابل. التحقق عام: يمكن لأي شخص لديه المفتاح العام التأكد من أن التوقيع يتوافق مع الرسالة. بدون المفتاح الخاص، لا يمكن لأحد إنتاج توقيع صالح لتلك الرسالة؛ ومع المفتاح العام، يمكن للجميع اكتشاف ما إذا كان التوقيع صالحاً. هذا عدم التماثل هو ما يسمح للموقع بإثبات التأليف دون مشاركة السر.

المثال السابق هو النسخة الدنيا من الدليل. في كود Solo2 الحقيقي، تعبر السلسلة ملفين، أحدهما في JavaScript يعيش في متصفح المستخدم ويعيد بناء الإنتروبيا من الكلمات الأربع والعشرين، والآخر في Zig داخل مكتبة *zcatcrypto* التي تأخذ تلك الإنتروبيا وتشتق المفاتيح التشفيرية المحددة. بدءاً من جانب المتصفح:

```
// 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')
                                                                                {

                                           .palabras = 264 bits. Los primeros 256 son la entropía 24 //
                                                 ;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 }
                                                                              {
```

تلك الاثنان وثلاثون بايت من الإنتروبيا، جنباً إلى جنب مع اثنين وثلاثين بايت أخرى مشتقة في نفس الخطوة، تنتقل إلى وحدة WebAssembly الخاصة بـ Zig التي تولد مفاتيح Ed25519 الفعلية. الدالة الكاملة، مع تنظيف الذاكرة النهائي، تتسع في شاشة واحدة:

```
                                                    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
                                                                              {
```

هناك تفصيلان يستحقان الإشارة. الأول: نفس البذرة تنتج دائماً نفس زوج المفاتيح — وهذا بالضبط ما يسمح باستعادة الهوية عن طريق إدخال الكلمات الأربع والعشرين في جهاز جديد. الثاني: يتم مسح البذرة صراحة من

الذاكرة في السطر الأخير. بعد تلك النقطة، لا يمكن حتى للدالة نفسها إعادة بناء المفاتيح؛ كلمات المستخدم هي المصدر الوحيد.

لمن يريد التحقق من ذلك بأرقام صغيرة. يمكن استعراض مخطط التوقيع بالكامل بأرقام صغيرة بما يكفي لإجراء الحسابات يدوياً. من يفضل عدم الخوض في الحساب يمكنه تجاوز هذه الفقرة دون فقدان سياق المقال؛ من يريد رؤية الآلية تعمل خطوة بخطوة سيجدها هنا. **القواعد العامة**، التي يمكن لأي شخص قراءتها: عدد أولي $p = 23$ (في Ed25519 الحقيقي يتكون من حوالي سبعة وسبعين رقماً؛ نستخدم ثلاثة وعشرين لتناسب الحسابات صفحة واحدة)، أساس $g = 2$ رتبته في هذه المجموعة هي $q = 11$، والاتفاق هو أن جميع العمليات الحسابية مع g تتم *módulo* p وجميع الأسس يتم تقليلها *módulo* q. **الاختيار الخاص**، واحد فقط ولا يتم مشاركته أبداً: السر $x = 6$. هذه هي الهوية.

الخطوة 1 — الجزء العام من الهوية. يتم حسابه مرة واحدة ويتم نشره علناً.

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

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

الجزء العام من الهوية هو **18**. يمكن لأي شخص أخذه واستخدامه للتحقق من التوقيعات المصنوعة بهذه الهوية. لا أحد، من خلال مراقبة الرقم 18 فقط، يمكنه استعادة السر 6: هذه هي مشكلة اللوغاريتم المنفصل التي سنعود إليها في النهاية.

الخطوة 2 — توقيع الرسالة. يريد صاحب الهوية توقيع الرسالة $m = 7$. يبدأ باختيار قيمة عشوائية جديدة $k = 4$، والتي ستستخدم مرة واحدة فقط ولن يتم مشاركتها أبداً (في Ed25519 الحقيقي، يتم اشتقاق k بشكل حتمي من الرسالة والسر لتجنب خطر إعادة استخدامه، ولكن الدور الذي يلعبه هو هذا بالضبط). بعد ذلك يحسب ثلاثة أرقام:

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

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

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

التوقيع هو الزوج **(r, s) = (16, 10)**. ينتقل علناً مع الرسالة. يمكن لأي شخص قراءته. ملاحظة تعليمية: في Ed25519 الحقيقي، الدالة H هي SHA-512، وهي قوية تشفيرياً؛ هنا نستخدم التبسيط $e = (r + m) \bmod q$ ليتمكن القارئ من تتبع الخطوات دون الحاجة لحساب هاش. هيكل الخوارزمية هو نفسه.

الخطوة 3 — التحقق من التوقيع. لدى المتحقق الجزء العام $y = 18$، والرسالة $m = 7$، والتوقيع $(r, s) = (16, 10)$. يعيد بناء e بنفس الطريقة — $e = (16 + 7) \bmod 11 = 1$ — ويتحقق مما إذا كانت هذه المساواة تتحقق:

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

يحسب الجانبين بشكل منفصل:

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

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

الجانبان يعطيان **12**. التوقيع صالح. يمكن لأي شخص لديه الجزء العام 18 الوصول إلى هذا الاستنتاج دون أن يعرف أبداً أن السر كان 6.

ماذا عن طرف ثالث يحاول التزوير؟ رأت إيفا كل ما هو عام يمر عبر القناة: $p = 23$, $g = 2$, $q = 11$, $y = 18$, $m = 7$, $r = 16$, $s = 10$. لتوقيع رسالة *مختلفة* باسم هذه الهوية، ستحتاج إلى معرفة x. طريقها الوحيد هو أن تسأل نفسها: «لأي أس x يتحقق $2^x \bmod 23 = 18$؟». مع $p = 23$ يمكنها تجربة 0، 1، 2، 3، ... والعثور عليه في ثوانٍ. ولكن عند استبدال 23 بعدد أولي من الأبعاد الحقيقية لـ Ed25519، فإن مساحة الأسس الممكنة تتجاوز عدد ذرات الكون المرئي. **لا يوجد اليوم أي خوارزمية معروفة للبشرية يمكنها استعراض تلك المساحة في أقل من مليارات السنين.** إنها نفس مشكلة اللوغاريتم المنفصل التي تدعم Diffie-Hellman في المقال السابق، مطبقة هنا على مخطط التوقيع.

ما قمنا باستعراضه للتو هو *بالضبط* Schnorr، مخطط التوقيع الذي يعد Ed25519 متغيراً منه متكيفاً مع منحنى إهليلجي. في Ed25519 الحقيقي، تتم جميع العمليات على نقاط منحنى محدد (Curve25519) بدلاً من الأعداد الصحيحة موديولو عدد أولي، والدالة H هي SHA-512 بدلاً من الجمع البسيط الذي استخدمناه أعلاه. الاستبدالان هما تعديلات في التنفيذ —لكسب مقاومة تشفيرية للقوة الغاشمة، وكسب خصائص أمان إضافية لـ k—. الهيكل الخوارزمي، والعمليات الثلاث، وسبب عدم التماثل، هي نفسها.

من المناسب هنا التوقف قليلاً، لأن السلسلة بأكملها يمكن أن تختلط من نظرة سريعة مع بدائية أخرى من الثلاثي: الهاش. هي ليست كذلك. الهاش هو دالة فريدة تضغط —تدخل بايتات كثيرة، وتخرج بصمة قصيرة، وهناك ينتهي الطريق—. الهوية التشفيرية هي زوج رياضي متكامل: السر يبقى ويوقع؛ ونظيره العام ينشر ويتحقق. حيث يقوم الهاش بدمج المعلومات في اتجاه واحد، تؤسس الهوية عدم تماثل بين نصفين. الهاش يشهد على ما قيل؛ الهوية تشهد على من قاله.

ما ليست عليه العبارة

ينبغي توضيح ثلاثة أخطاء شائعة. العبارة ليست كلمة مرور بالمعنى الصحيح: لا تتم مقارنتها ببصمة مخزنة على خادم؛ بل يتم إدخالها في جهاز المستخدم لإعادة بناء الهوية رياضياً. العبارة لا تُسترد: إذا فُقدت، فلا يوجد أحد تطلبها منه؛ وإذا نُسخت، تُنسخ الهوية أيضاً. العبارة ليست وثيقة اعتماد منفصلة عن الهوية: العبارة هي الهوية. من يملكها يمكنه التصرف بصفتها، دون إذن إضافي، ودون عملية تفويض، ودون استرداد ممكن.

هذه الخاصية الثالثة هي التي تغير ثقل المسألة. كلمة المرور المفقودة هي إزعاج إداري. أما الهوية التشفيرية المفقودة فهي الهوية نفسها. الورقة التي تحتوي على العبارة والتي يجدها أطراف ثالثة ليست خطراً لسرقة الحساب: بل هي تسليم للهوية بأكملها. إن وعد النظام —ألا يتمكن أحد من إلغاء هويتك أو حظرك تعسفياً— يأتي مصحوباً بشكل لا ينفصم بالمسؤولية — بأنك الحارس الوحيد لشيء لا يمكن لأحد استعادته لك.

الوعد والثقل

غالباً ما يوصف نموذج الهوية التشفيرية بأنه *ذاتي السيادة* —self-sovereign في الأدبيات الأنجلوسكسونية—. اختيار الكلمة متعمد ويصف الحالة بدقة تامة. المستخدم سيد على هويته بمعنى شبه قروسطي: لا يمنحه إياها أي ملك، ولا أي جهة إصدار، ولا أي سلطة مركزية؛ ولا يمكن لأي مما سبق سحبها منه. ولكنه أيضاً، مثل العاهل القروسطي، يتحمل العواقب الكاملة لأخطائه: لا يوجد وصي يتخذ القرارات مكانه إذا فقد الختم.

الاختيار بين الهوية التي يديرها طرف ثالث والهوية ذاتية السيادة ليس له إجابة عالمية صحيحة واحدة. بالنسبة لحساب منتدى غير مهم، من المحتمل أن تكون الهوية المدارة متناسبة مع المخاطر. أما بالنسبة لهوية مهنية توقع وثائق ملزمة قانوناً، أو لهوية اقتصادية تحرس المدخرات الخاصة، أو لهوية تواصل مهني مع عملاء اؤتمنوا على معلومات حساسة، فإن المسألة تتغير. هناك يتوقف السؤال عن كونه «هل هو مريح؟» ويصبح «من، غيري، لديه القدرة على التصرف بصفتي، وتحت أي ظروف؟».

أين تظهر هذه الآلية في الأنظمة الحقيقية

وُلد BIP39 في عالم Bitcoin في عام 2013 وانتشر بسرعة إلى نظام العملات المشفرة بأكمله: أي محفظة جادة اليوم تقبل عبارة BIP39 المكونة من اثنتي عشرة أو أربع وعشرين كلمة كدعم للهوية الاقتصادية لصاحبها. خارج العملات المشفرة، يظهر نفس المفهوم الأساسي — الزوج التشفيري الذي يثبت التأليف بدون وسيط — في أنظمة أخرى بصيغ مختلفة. مفاتيح SSH التي يستخدمها مسؤول النظام للوصول إلى خوادمه هي حالة كلاسيكية: مفتاح خاص يحفظه المسؤول على جهازه ومفتاح عام يتم نسخه إلى كل خادم؛ لا يتدخل أي طرف مشابه لخدمة مركزية. يستخدم بروتوكول Signal نظام Ed25519 مع مادة مفتاح دائمة على الجهاز؛ أما eIDAS الأوروبية، في جزء التوقيع المؤهل، فتعتمد على نفس المبدأ التشفيري، مع فارق أن المفتاح يحفظه مزود خدمة ثقة مؤهل بدلاً من المستخدم.

تستخدم Solo2، المنصة الناشرة لهذا الإصدار، عبارة BIP39 المكونة من أربع وعشرين كلمة كـ هوية لكل مستخدم. يرى المستخدم الكلمات مرة واحدة عند إنشاء حسابه. لا يتم تخزينها في أي خادم لـ Solo2 أو أي طرف آخر: إذا قام

المستخدم بتدوينها وحفظها، فإنه يحافظ على هويته للأبد. إذا فقدها، فقدها. هذه هي النتيجة المنطقية لهندسة معمارية لا يوجد فيها مشغل وسيط: إذا كانت Solo2 قادرة على إعادة الهوية للمستخدم الذي فقدها، فستكون قادرة أيضاً على منحها لمن يضغط على Solo2 للحصول عليها.

للقارئ المهني

أربعة اعتبارات لمن يقيم اعتماد هوية تشفيرية سيادية ذاتية (autosoberana) في سياق مهني:

1. العبارة هي الهوية. الحفظ المادي — الورق، نسخ متعددة في أماكن مختلفة، أو حتى المعدن المحفور للاستخدام طويل الأمد — يوفر ضمانات أكثر من الحفظ الرقمي، الذي يزيد من مساحة الهجوم دون تقليل خطر الفقدان.
2. لا يوجد استرداد. تصميم العملية بافتراض ضياع النسخة الأولية يوماً ما هو أفضل بكثير من اكتشاف ذلك يوم ضياعها. نسخة ثانية مفصولة جغرافياً تحل معظم السيناريوهات.
3. الأمر ليس هو نفسه شهادة eIDAS المؤهلة. بالنسبة للتوقيع المؤهل في الاتحاد الأوروبي — العقود الموثقة، بعض الإجراءات مع الإدارة — يتطلب التشريع مزوداً مؤهلاً يحفظ المفتاح. الهوية التشفيرية السيادية الذاتية (autosoberana) تخدم التواصل المهني والتوقيع المستندي ذو القيمة الإثباتية، لكنها لا تحل محل الشهادة المؤهلة تلقائياً في الحالات التي يتطلب فيها القانون ذلك.
4. إذا كانت الهوية ستنتقل — ميراث، تعاقب مهني، إغلاق نشاط — فمن الأفضل إعداد الإجراء مسبقاً، وليس لاحقاً. الإجراءات الرسمية بظروف مختومة بالشمع (lacre)، تعليمات لمنفذ وصية، إيداع لدى كاتب عدل، هي ترتيبات كلاسيكية متوافقة تماماً مع الطبيعة التشفيرية للأصل.

تغلق هذه المقالة الثلاثية المفاهيمية التي افتتحت الدورة — hash، التشفير، الهوية —. الأفكار الثلاث تبنى فوق بعضها البعض: الـ hash يعطي البصمة غير القابلة للتغيير، التشفير يعطي السرية بدون طرف ثالث موثوق، والهوية تعطي التأليف بدون طرف ثالث مانح. تشترك الثلاثة في خاصية ليست أيديولوجية أيضاً: فهي تنقل القدرات التقنية التي كانت تقيم تقليدياً لدى المشغل، من مدير الخدمة إلى مستخدمها. وتنقل معها أيضاً المسؤوليات. التحدث بصدق عن أي من الثلاثة يتطلب التحدث أيضاً عن الاثنين الآخرين.

المصادر ومزيد من القراءة

- 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.

← السابقنموذج العمل كإشارة ثقةالتالي → الاستضافة الذاتية كممارسة مهنية

قراءات حديثة

- تأمل · 29 يونيو 2026 أنت لست مجهول الهوية
- تأمل · 27 مايو 2026 ما لا يمكن لتوقيع أن يصلحه
- تحليل · 26 مايو 2026 الخصوصية الحقيقية مقابل الظاهرية: الأسئلة التي ينبغي طرحها

خذ هذا المقال معك أينما احتجت إليه.

↓ ماركداون ↓ نص عادي ↓ PDF

سيتم تنزيل الملف على جهازك. من هناك يمكنك حفظه، أو استيراده إلى Solo2 أو مشاركته أينما تريد. Cuadernos لا تقرر الوجهة نيابة عنك.

ختم شمعي · SHA-256 7c18f77f3f36df559431df1391191b1e2ccedbdb4613701365702997a0eed6c2

Cuadernos Lacre · منشور لشركة Menzuri Gestión S.L. ·
بقلم R.Eugenio · تحرير فريق Solo2.