Le chiffrement de bout en bout, enfin expliqué

Ce que les fournisseurs disent quand ils disent E2EE, et ce qu'ils ne disent pas. Une explication didactique du mécanisme et de ses limites, sans l'enrobage publicitaire.

Soyons clairs : WhatsApp dit que vos messages sont chiffrés de bout en bout. C'est vrai — et ce n'est pas suffisant. Si la sauvegarde va sur iCloud ou Google Drive sans chiffrement supplémentaire, le chiffrement est rompu sur votre propre téléphone. La question opérationnelle n'est pas de savoir si c'est chiffré, mais où résident les clés.

Ce que chiffrer signifie, vraiment

Chiffrer un message, c'est le transformer en quelque chose qui ressemble à du bruit pour quiconque ne possède pas une certaine information appelée clé. L'opération est effectuée sur l'appareil de l'expéditeur et, avec la bonne clé, elle est annulée sur l'appareil du destinataire. Entre les deux, le message voyage comme une succession d'octets sans signification apparente. C'est l'idée simple. Le reste de l'article traite des nuances qui la transforment, selon le cas, en une véritable garantie ou en une étiquette marketing.

L'adjectif *de bout en bout* — en anglais *end-to-end*, abrégé E2EE — ajoute une précision. Le chiffrement n'est pas fait pour qu'un serveur intermédiaire puisse le lire et le livrer. Il est fait pour que seuls les deux bouts — l'appareil de l'expéditeur et l'appareil du destinataire — possèdent la clé. Tout serveur par lequel le message passe voit du bruit, pas le message. C'est la différence technique avec le chiffrement *en transit*, où le contenu voyage chiffré d'un serveur au suivant, mais chaque serveur par lequel il passe le déchiffre pour le réexpédier, récupérant temporairement le texte en clair.

Le paradoxe du secret partagé

Il y a un problème évident. Pour que deux personnes puissent chiffrer et déchiffrer des messages entre elles, elles ont toutes deux besoin de la même clé. Mais comment s'entendent-elles sur cette clé si tout ce qu'elles s'envoient, par définition, passe par un canal où quelqu'un pourrait écouter ? Convenir de la clé sur le canal même où elles l'utiliseront ensuite semble impossible : si l'attaquant l'entend lors de l'accord, il pourra déchiffrer tout ce qui suit. Pendant des décennies, la cryptographie classique a résolu cela par la manière forte : les clés étaient remises en personne, avant de commencer à être utilisées, lors de rencontres physiques. Les ambassadeurs transportaient des mallettes de clés cousues dans la doublure de leur manteau.

Dans le courrier électronique contemporain, cette solution n'est pas scalable. Si nous devions aller physiquement chez chaque personne avec qui nous avons l'intention de communiquer de manière chiffrée, nous ne pourrions parler à personne. La question posée il y a cinquante ans par la communauté cryptographique était celle-ci : est-il possible que deux personnes qui ne se connaissent pas et qui ne partagent qu'un canal public conviennent, sur ce même canal public, d'un secret que personne écoutant le canal ne puisse connaître ?

L'élégance de Diffie-Hellman

En 1976, deux mathématiciens nommés Whitfield Diffie et Martin Hellman ont démontré quelque chose d'apparemment impossible : que deux personnes, ne parlant que par un canal public — un canal où n'importe qui peut entendre tout ce qu'elles disent —, peuvent convenir d'un mot de passe secret sans qu'aucun auditeur ne puisse le découvrir. Cela ressemble à de la magie. Ce n'en est pas : c'est mathématique. L'échange de clés Diffie-Hellman, comme on l'appelle depuis lors, est la base de pratiquement toute la communication chiffrée d'internet, et un demi-siècle d'utilisation intensive et de contrôle académique mondial confirment sa solidité. Quiconque veut voir l'intuition visuelle ou les mathématiques peut continuer sa lecture. Quiconque préfère faire confiance au fait que cela fonctionne peut également continuer sans perdre le fil de l'article.

Pour quiconque veut l'imaginer en une image, il existe une analogie connue avec des couleurs. Imaginez qu'Alice et Bruno conviennent ouvertement d'une couleur de base — disons le jaune — à la vue d'Eva, qui les écoute. Chacun choisit en privé une seconde couleur secrète et mélange son secret avec le jaune. Alice obtient un orange particulier ; Bruno obtient un vert particulier. Ils échangent les résultats à la vue d'Eva. Maintenant chacun mélange la couleur reçue avec son propre secret, et tous deux arrivent à la même couleur finale, car l'ordre des mélanges n'importe pas. Eva a vu le jaune et les deux mélanges intermédiaires, mais pas les secrets ; sans l'un des secrets, elle ne peut arriver à la couleur finale. Les mathématiques réelles remplacent les couleurs par des exponentiations dans des groupes modulaires ou des courbes elliptiques, mais l'idée est la même : le secret partagé est construit en public sans que personne dans le canal ne puisse le reconstruire.

En arithmétique, pour qui préfère voir le mécanisme : Alice choisit un nombre secret a, Bruno choisit b. Ils échangent g^a et g^b en clair sur le canal. Alice calcule $(g^b)^a$ et Bruno calcule $(g^a)^b$; tous deux arrivent au même g^{ab}. Eva voit g, g^a et g^b passer par le canal, mais récupérer a depuis g^a — ce qu'on appelle le problème du logarithme discret — nécessite un temps de calcul astronomiquement supérieur à l'âge de l'univers quand g est choisi dans un groupe mathématique adéquat.

Pour ceux qui veulent le vérifier avec de petits nombres. L'échange Diffie-Hellman peut être parcouru en entier avec des chiffres assez petits pour faire les calculs à la main. Quiconque préfère ne pas se plonger dans l'arithmétique peut sauter ce bloc sans perdre le fil de l'article ; quiconque veut voir le mécanisme fonctionner étape par étape le trouvera ici. **Les règles publiques**, que n'importe qui peut lire : un nombre premier $p = 11$ (dans le vrai Diffie-Hellman, il compte environ trois cents chiffres ; nous utilisons onze pour que les calculs tiennent sur une page), une base $g = 2$, et la convention que toute l'arithmétique se fait *modulo p* — on calcule, on divise par p, et on garde le reste, comme une horloge à onze positions qui revient à zéro en dépassant le dix. **Les choix privés**, un chacun et jamais partagés : Alice choisit $a = 4$. Bruno choisit $b = 7$.

Étape 1. Alice calcule $2^4 = 16$, puis $16 \bmod 11 = \mathbf{5}$. Elle envoie le cinq. Eva le note.

Étape 2. Bruno calcule $2^7 = 128$, puis $128 \bmod 11 = \mathbf{7}$. Il envoie le sept. Eva le note également. Après les deux envois, le carnet d'Eva contient quatre données : $p = 11$, $g = 2$, A = 5, B = 7. Il lui manque le nombre partagé qu'Alice et Bruno sont sur le point de dériver — et qu'Eva ne pourra pas reconstruire.

Étape 3. Alice prend le sept que Bruno lui a envoyé et l'élève à son exposant privé $a = 4$. Pour éviter de manipuler $7^4 = 2401$, on calcule par parties en appliquant le modulo à chaque étape :

$7^2 = 49$

$49 \bmod 11 = 5$

$7^4 = (7^2)^2 = 5^2 = 25$

$25 \bmod 11 = 3$

Alice obtient le nombre **3**.

Étape 4. Bruno prend le cinq qu'Alice lui a envoyé et l'élève à son exposant privé $b = 7$. Encore par parties :

$5^2 = 25 \bmod 11 = 3$

$5^4 = (5^2)^2 = 3^2 = 9 \bmod 11 = 9$

$5^6 = 5^4 \times 5^2 = 9 \times 3 = 27 \bmod 11 = 5$

Finalement $5^7 = 5^6 \times 5 = 5 \times 5 = 25 \bmod 11 = \mathbf{3}$.

Bruno obtient également **3**.

Les deux sont arrivés au même nombre, 3, en travaillant en parallèle. Aucun n'a envoyé son exposant privé à aucun moment. Alice ne sait pas que $b = 7$; Bruno ne sait pas que $a = 4$. Chacun a utilisé la valeur publique que l'autre a envoyée combinée avec son propre exposant privé, et ils se sont rencontrés à la même destination. **Pourquoi arrivent-ils au même nombre ?** Ce que chacun a calculé : Alice, $(g^b)^a = 2^{7\times4} = 2^{28} \bmod 11$. Bruno, $(g^a)^b = 2^{4\times7} = 2^{28} \bmod 11$. C'est la même quantité car l'ordre de multiplication des exposants n'a pas d'importance ($7 \times 4 = 4 \times 7$). Chacun est arrivé par un chemin différent à la même destination.

Et Eva ? Elle a dans son carnet $p = 11$, $g = 2$, A = 5, B = 7, et voudrait le 3. Pour le calculer, elle aurait besoin de connaître a ou b — mais aucun des deux n'a voyagé sur le canal. Sa seule solution est de se demander : « pour quel exposant a a-t-on $2^a \bmod 11 = 5$? ». Avec un p si petit, elle peut essayer 0, 1, 2, 3, 4... et le trouver en moins d'une minute. Mais en remplaçant 11 par un nombre premier de trois cents chiffres, l'espace des exposants possibles contient plus d'éléments qu'il n'y a d'atomes dans l'univers observable. **Il n'existe aujourd'hui aucun algorithme connu de l'humanité capable de parcourir cet espace en moins de milliards d'années.** C'est le fameux *problème du logarithme discret* : facile en avant, informatiquement impossible en arrière. Et c'est la raison pour laquelle le chiffrement résiste même si Eva a suivi toute la conversation lettre par lettre.

Trois ingrédients simples — l'arithmétique sur une horloge, l'exponentiation, et la commutativité de la multiplication ($a \cdot b = b \cdot a$) — combinés produisent un protocole dont dépend la moitié de l'humanité chaque jour pour ses communications privées. Aucune des trois pièces, prise séparément, ne semble spéciale. Ce qui est décisif, c'est l'assemblage.

De Diffie-Hellman au protocole Signal

Le chiffrement de bout en bout qu'utilisent aujourd'hui les applications de messagerie professionnelle repose, presque sans exception, sur une version élégante et durcie de l'échange Diffie-Hellman. Le protocole Signal, conçu par Trevor Perrin et Moxie Marlinspike entre 2013 et 2016, est la référence. Il combine deux idées clés. La première, l'échange de clés en courbes elliptiques (X25519), qui produit le secret partagé initial entre deux appareils. La seconde, le « Double Ratchet » — double engrenage —, qui renouvelle les clés automatiquement avec chaque message, de sorte que compromettre l'appareil aujourd'hui ne permet pas de déchiffrer les messages passés, ni les messages futurs une fois que l'engrenage a tourné.

En Zig, l'échange X25519 qui produit le secret partagé entre deux appareils tient en six lignes, en utilisant la bibliothèque standard :

```
const std = @import("std");
const X25519 = std.crypto.dh.X25519;
```

```
// Alicia y Bruno generan cada uno un par (privada, pública).
const par_alicia = X25519.KeyPair.generate(io);
const par_bruno  = X25519.KeyPair.generate(io);

// Cada parte recibe la clave pública de la otra y deriva el mismo secreto.
const secreto_alicia = X25519.scalarmult(par_alicia.secret_key, par_bruno.public_key) catch unreachable;
const secreto_bruno  = X25519.scalarmult(par_bruno.secret_key,  par_alicia.public_key) catch unreachable;
// secreto_alicia == secreto_bruno  (32 bytes)
```

Ce qui se passe dans ces six lignes : Les clés publiques voyagent ouvertement. Les clés privées ne sortent jamais de l'appareil respectif. Chaque partie dérive, à partir de sa propre clé privée et de la clé publique de l'autre, un même secret de trente-deux octets que personne sur le canal ne peut récupérer. Ce secret sert ensuite de graine pour chiffrer les messages échangés. Le Double Ratchet du protocole Signal ajoute une rotation constante de ce matériel pour que le compromis d'un instant ne compromette pas le reste de la conversation.

Et que trouve-t-on exactement dans `std.crypto.dh.X25519` ? Pas de magie cachée. Ce sont deux fonctions courtes qui peuvent être lues en entier dans la bibliothèque standard de Zig elle-même. La première dérive la clé publique de la clé privée — le « g^a » de l'échange :

```
pub fn recoverPublicKey(secret_key: [secret_length]u8) IdentityElementError![public_length]u8 {
    const q = try Curve.basePoint.clampedMul(secret_key);
    return q.toBytes();
}
```

Dans le langage de l'article : la clé privée est « multipliée » — au sens elliptique, et non arithmétique élémentaire — par le point de base de la courbe Curve25519, et le résultat est sérialisé en trente-deux octets. L'opération `clampedMul` est la version durcie de cette multiplication scalaire : elle intègre les protections que la communauté cryptographique a ajoutées au fil des ans pour résister à des familles connues d'attaques. Deux lignes de corps de fonction.

La deuxième fonction combine votre clé privée avec la clé publique que l'autre partie vous envoie. C'est le « $(g^b)^a$ » de l'échange, qui produit le secret partagé de trente-deux octets qu'aucun de vous n'a jamais transmis :

```
pub fn scalarmult(secret_key: [secret_length]u8, public_key: [public_length]u8) IdentityElementError![shared_length]u8 {
    const q = try Curve.fromBytes(public_key).clampedMul(secret_key);
    return q.toBytes();
}
```

Deux autres lignes. La clé publique reçue est interprétée comme un point sur la courbe, et est « multipliée » par sa propre clé privée. En raison de la commutativité de l'opération sur la courbe — analogue à la commutativité de la multiplication des exposants que nous avons vue dans l'exemple numérique —, les deux parties se retrouvent avec le même point sérialisé : exactement le secret partagé dont parle l'article.

C'est tout. Ce qui dans une application ressemble à de la magie se résume en réalité à deux fonctions de trois lignes chacune. La complexité technique est concentrée dans une seule opération, `clampedMul`, qui est écrite plus loin dans la même bibliothèque standard, révisée pendant des décennies par la communauté cryptographique internationale, et disponible pour quiconque souhaite la lire lettre par lettre. Il n'y a pas de boîte noire, ni dans notre application ni dans la bibliothèque standard de Zig. Il y a du code open source qu'un humain peut comprendre, en choisissant le rythme auquel il veut s'y plonger.

Ce que le chiffrement de bout en bout protège

Ce que l'E2EE protège bien, en supposant une implémentation correcte, c'est le contenu du message en transit. Un serveur intermédiaire qui reçoit et réexpédie les données chiffrées verra une succession d'octets inintelligibles. Un attaquant ayant accès au câble, au routeur, au point d'accès wifi, verra la même chose. Un fournisseur de service qui conserve des copies du trafic ne pourra pas le lire a posteriori. Un Gouvernement qui ordonne à l'opérateur du service de livrer le contenu recevra les mêmes octets inintelligibles que ceux que le serveur avait au départ.

Ceci, en termes pratiques, est déjà beaucoup. C'est la différence entre écrire une lettre dans une enveloppe opaque et l'écrire sur une carte postale. Les deux arrivent. Une seule préserve le contenu face au facteur.

Ce que le chiffrement de bout en bout ne protège pas

Il convient de le savoir tout aussi bien. L'E2EE ne protège pas les métadonnées : le serveur sait toujours que l'utilisateur A envoie des données à l'utilisateur B, à quelle heure, avec quelle fréquence et d'où, même s'il ne sait pas ce qu'ils disent. Ces métadonnées, comme nous l'avons déjà argumenté dans *Chiffrer n'est pas être privé*, sont souvent plus révélatrices que le contenu. Savoir que quelqu'un a appelé un cabinet d'avocats spécialisé dans les divorces un vendredi à 22h00 pendant trente minutes raconte une histoire que le contenu de l'appel n'aurait jamais racontée. C'est la même situation que de voir une personne entrer et sortir plusieurs fois d'une clinique oncologique : il n'y a pas besoin d'entendre quoi que ce soit de ce qui se dit à l'intérieur pour imaginer ce qui se passe. Une seule métadonnée isolée peut ne rien signifier ; plusieurs métadonnées croisées dessinent quelque chose de trop proche de la vérité. L'E2EE ne protège pas les extrémités : si l'appareil du destinataire est compromis par un programme malveillant, le message est déchiffré normalement pour ce destinataire et le programme malveillant le lit. L'E2EE ne protège pas contre l'identité de l'interlocuteur en soi : si Alice croit parler à Bruno mais qu'un attaquant s'est interposé au début (un *man in the middle*) et que le protocole n'inclut pas de vérification indépendante, les deux parties finissent par parler à l'intrus en pensant qu'elles se parlent l'une à l'autre.

Il y a une quatrième chose qu'il convient de formuler sans ambiguïté. L'E2EE n'empêche pas un fournisseur qui affirme l'offrir de conserver, en plus, une copie du message non chiffré dans ses propres systèmes. L'affirmation « mes messages sont chiffrés de bout en bout » et l'affirmation «

le fournisseur ne conserve pas mon contenu » ne sont pas les mêmes. Une application peut respecter la première tout en enfreignant la seconde ; nous l'avons vu dans les titres de presse à plusieurs reprises depuis 2018. L'utilisateur, à moins que le code du client ne soit vérifiable, n'a aucun moyen technique de distinguer un cas de l'autre sans une investigation experte. Le cas le plus connu du grand public : WhatsApp chiffre les messages de bout en bout en transit, mais si l'utilisateur active la sauvegarde sur iCloud ou Google Drive sans chiffrement supplémentaire, cette copie est stockée de manière lisible dans l'infrastructure d'un tiers, et le chiffrement est rompu au bout de l'utilisateur lui-même.

La question que l'opérateur ne veut pas entendre

Une application qui affirme chiffrer de bout en bout peut, techniquement, faire l'une des trois choses suivantes concernant les clés :

1. **Les clés résident uniquement sur les appareils.** Elles sont générées et résident exclusivement sur les appareils des utilisateurs ; l'opérateur ne les connaît pas et ne les stocke pas. C'est le cas optimal.
2. **L'opérateur peut y accéder s'il le veut.** L'opérateur possède les clés des utilisateurs (ou peut les générer à sa guise) et les stocke dans ses bases de données. S'il le veut ou s'il y est contraint, il peut lire le contenu. C'est le cas de la majorité des services « cloud ».
3. **L'opérateur ne peut pas y accéder par conception, mais contrôle l'accès.** L'opérateur n'a pas les clés, mais a le contrôle de l'application qui les génère. S'il y est contraint, il peut envoyer une mise à jour malveillante qui capture les clés ou le contenu avant le chiffrement. C'est le cas de nombreux services E2EE commerciaux.

La question opérationnelle n'est donc pas de savoir si quelque chose est chiffré, mais qui a le contrôle de l'appareil et du logiciel qui gère les clés. Chez Solo2, les clés résident uniquement dans votre Coffre (IndexedDB chiffrée avec votre mot de passe) et le logiciel est un code source ouvert vérifiable.

Pour le lecteur professionnel

Le chiffrement de bout en bout est un outil de souveraineté numérique. Mais comme tout outil, son efficacité dépend de la main qui le manie et du sol sur lequel il repose.

1. Où sont générées les clés cryptographiques et où résident-elles physiquement ? Si l'opérateur peut y accéder (même temporairement, même sous couvert de récupération), l'E2EE est nominal.
2. Existe-t-il une vérification indépendante de l'interlocuteur (numéros de sécurité, codes QR, comparaison hors bande) qui empêche une attaque de l'homme du milieu pendant l'établissement de la conversation ?
3. Le code du client est-il auditable — ouvert, publié, reproductible — ou exige-t-il de faire confiance à la parole du fournisseur sur ce que le client fait réellement ?
4. Quelles métadonnées le service génère-t-il et conserve-t-il, et pour combien de temps ? Même si le contenu est opaque, les métadonnées peuvent reconstruire une bonne partie des informations sensibles.

Ces quatre questions ne demandent pas d'informations techniques avancées ; elles demandent des informations que tout opérateur honnête peut fournir dans sa documentation publique. La qualité et la précision de la réponse en disent aussi long sur le produit que la réponse elle-même.

Le chiffrement de bout en bout, bien fait, est l'une des constructions les plus fines que la cryptographie contemporaine ait livrées à la pratique quotidienne. L'idée originale — deux personnes peuvent convenir d'un secret sur un canal public — appartient à Whitfield Diffie et Martin Hellman, 1976 ; un demi-siècle plus tard, nous vivons toujours dans ses conséquences. Mais, comme pour toute promesse technique, sa valeur dépend de sa réalisation réelle, non de l'étiquette. La question du professionnel honnête n'est pas « est-ce chiffré ? », mais « qui a les clés ? ». Les réponses ont des conséquences différentes. Il convient de les connaître.

Sources et lectures complémentaires

- Diffie, W. ; Hellman, M. — *New Directions in Cryptography*, IEEE Transactions on Information Theory, novembre 1976. Article fondateur de la cryptographie à clé publique.
- Perrin, T. ; Marlinspike, M. — *The Double Ratchet Algorithm*, spécification publique d'Open Whisper Systems, révision de 2016. Base du protocole Signal et de ses dérivés industriels.
- RFC 7748 — Elliptic Curves for Security (IETF, janvier 2016). Spécification normative des courbes X25519 et X448 utilisées dans les échanges de clés modernes.
- Ferguson, N. ; Schneier, B. ; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Chapitres sur l'échange de clés et les protocoles de chiffrement authentifié.
- Règlement (UE) 2024/1183 sur le cadre européen d'identité numérique (eIDAS 2) — établit des cadres où la vérification indépendante de l'interlocuteur acquiert un soutien institutionnel, et où la distinction entre chiffrement nominal et chiffrement réel a des conséquences juridiques différentes.

← PrécédentKill switch et capture institutionnelleSuivant →Le modèle économique comme signal de confiance

Lectures recentes

- Analyse · 18 mai 2026 Confidentialité réelle vs apparente : les questions à se poser
- Analyse · 18 mai 2026 Self-hosting comme pratique professionnelle
- Concept · 18 mai 2026 Les 24 mots : qu'est-ce qu'une identité cryptographique

Emportez cet article où vous en avez besoin.

↓ Markdown ↓ Texte brut ↓ PDF

Le fichier est téléchargé sur votre appareil. À partir de là, vous pouvez l'enregistrer, l'importer dans Solo2 ou le partager comme vous le souhaitez. Cuadernos ne décide pas de la destination pour vous.

Cachet de cire · SHA-256 f24e46e8af7e9723c1ea2be18b6e302b8de4d7ccd1a8d13c1ea43352fa9e91ca

Cuadernos Lacre · Une publication de Menzuri Gestión S.L. ·
écrite par R.Eugenio · éditée par l'équipe de Solo2.