# WIGGWIGG

Plateforme d'identité et de communications à connaissance nulle

# Livre blanc sur la sécurité de WIGGWIGG

**Version :** 1.0 (ébauche) · **Date :** 2026-06

**WIGGWIGG** est un produit de numéro de téléphone et d'identité axé sur la confidentialité. Ce document explique, en détail technique, comment on protège vos données : ce qu'on ne peut **pas** lire, ce qu'on **peut** lire et pourquoi, la cryptographie qu'on utilise, et où s'arrêtent nos garanties. Il s'adresse à un lecteur technique et sceptique.

Notre noyau cryptographique est ouvert sur github.com/WiggWiggDev/crypto (le paquet `@wiggwigg/crypto`) : les primitives de dérivation de clé, de scellement et de signature décrites ici sont vérifiables indépendamment de l'application fermée.

## 1. Résumé et position

- **À connaissance nulle par conception.** Votre mot de passe, vos entrées de coffre, vos identités, vos contacts et le **contenu** de vos messages sont chiffrés sur votre appareil avec des clés qu'on ne détient jamais. On ne peut pas les lire, et on ne peut pas être contraints de produire une clé qu'on n'a pas.
- **Noyau ouvert, pas « code source ouvert ».** On a ouvert le noyau cryptographique pour que vous puissiez vérifier *comment* les clés et les données sont protégées. Le reste de l'application (les surfaces opérationnelles et de lutte contre les abus) est fermé. On ne laisse jamais entendre le contraire.
- **Les limites.** Les numéros de téléphone, les adresses d'urgence (E911) et les métadonnées de routage sont lisibles par le serveur parce que le service téléphonique l'exige. On n'est pas chiffrés de bout en bout en transit (le réseau téléphonique est antérieur à cette possibilité). On n'a pas encore de confidentialité persistante ni de protection post-quantique. Ces limites sont énoncées aux sections 10 et 11.
- **Auto-attesté aujourd'hui, audité de l'extérieur plus tard.** Ce document s'appuie sur du code source ouvert et une politique publiée de divulgation des vulnérabilités. Un audit cryptographique et applicatif par un tiers est **prévu** ; on le publiera une fois terminé et on ne le revendiquera pas avant.

## 2. Modèle de menace et frontières de confiance

On conçoit contre un **serveur honnête mais curieux** : on suppose que notre infrastructure peut être compromise, assignée en justice, ou exploitée par quelqu'un qui ne devrait pas voir vos données. Sous cette hypothèse, vos données à connaissance nulle restent illisibles parce que les clés vivent sur votre appareil.

| Frontière | Ce qu'on suppose | Ce qu'elle peut voir |
|---|---|---|
| Votre appareil | De confiance ; la crypto côté client s'y exécute | Tout ce que vous stockez (il détient vos clés) |
| Votre mot de passe / phrase de récupération | Secret, connu de vous seul | Rien ne quitte l'appareil |
| Serveurs **WIGGWIGG** | Honnêtes mais curieux, peuvent être compromis | Données opérationnelles chiffrées côté serveur (section 4) ; jamais les données à connaissance nulle |
| Base de données | Protégée par la sécurité au niveau des lignes et des déclencheurs | Du texte chiffré seulement, pour les données à connaissance nulle |
| Opérateur télécom (Telnyx) | Un « tuyau bête » qui n'apprend que ce que le routage exige | Métadonnées d'appels/SMS et contenu **en transit** (comme tout service téléphonique) |
| Processeurs de paiement (Stripe, Blockonomics) | Données minimisées | L'essentiel de la facturation par carte, ou une adresse Bitcoin et un montant ; jamais votre coffre ni vos identités |

On ne défend **pas** contre un appareil compromis (un logiciel malveillant sur votre propre appareil, ou une extension de navigateur malveillante), ni contre les métadonnées inhérentes au fait de passer un appel ou un SMS sur le réseau public. Voir la section 10.

## 3. Architecture cryptographique

Toutes les primitives ci-dessous proviennent de bibliothèques auditées ( `@noble/curves` , `@noble/hashes` , `@scure/bip39` ). On n'écrit pas la nôtre.

- **Dérivation de clé :** votre mot de passe est étiré avec **Argon2id** (128 Mio de mémoire, 3 itérations). C'est à coût mémoire et résistant au forçage par GPU et ASIC. Les paramètres sont identiques au bit près entre le web et le mobile, pour que vos clés concordent sur tous vos appareils.
- **Indirection de la clé de coffre :** la *clé principale* dérivée du mot de passe enveloppe une *clé de coffre* distincte, propre au compte, et vos données sont chiffrées sous la clé de coffre. Changer votre mot de passe ré-enveloppe la clé de coffre seulement ; ça ne re-chiffre jamais vos données.
- **Chiffrement symétrique : AES-256-GCM** (authentifié) pour les entrées de coffre, les identités, les contacts, le contenu des messages et les champs opérationnels côté serveur.
- **Scellement par clé publique :** le contenu des messages entrants est chiffré avec AES-256-GCM sous une clé à usage unique, et cette clé est scellée à la clé publique **X25519** de votre appareil via X25519-ECDH, HKDF-SHA256 et XChaCha20-Poly1305 (un coffre scellé de style libsodium). On génère la paire de clés de l'expéditeur, on l'utilise une seule fois et on la détruit, donc on ne conserve **aucune clé** capable de lire vos messages (voir la section 5).

- **Authentification :** prouver que vous connaissez votre mot de passe sans le révéler (section 6).
- **Séparation des clés :** chaque clé dérivée (authentification, coffre, communications, récupération) provient de la clé principale par **HKDF avec séparation de domaine**, donc compromettre l'une n'en livre jamais une autre.
- **Agilité cryptographique :** chaque schéma porte une version, pour qu'on puisse faire pivoter une primitive sans casser les données stockées. C'est ainsi que la mise à niveau post-quantique de la section 11 arrive sans réécriture.

## 4. Classification des données

Ce qui est, ou n'est pas, à connaissance nulle :

| Niveau | Données | Qui peut les lire |
|---|---|---|
| **Connaissance nulle** | Identités, entrées de coffre, contacts, **contenu** des SMS/MMS/messagerie vocale (par défaut), préférences anti-hameçonnage | Votre appareil seulement |
| **Chiffré côté serveur** | Numéros de téléphone (routage), **adresses E911** (répartition d'urgence), identifiants SIP, métadonnées de facturation, coût et métadonnées d'opérateur des appels/SMS | Le serveur peut déchiffrer (requis pour fonctionner) |
| **Index aveugle** | Recherches d'égalité (p. ex. par numéro) sans stocker le texte en clair | Le serveur compare un HMAC à clé, pas la valeur |
| **Texte en clair** | Identifiant de compte (un nom d'utilisateur public), horodatages, clés primaires et étrangères, `is_admin` | Quiconque a accès à la base (non sensible) |

**Les exceptions :** les numéros de téléphone et les adresses E911 sont *lisibles par le serveur* parce que le réseau doit router vos appels et que les services d'urgence doivent vous localiser. Le contenu de la messagerie vocale est à connaissance nulle **sauf** si vous activez l'écoute par téléphone, qui conserve une copie lisible par le serveur pour que vous puissiez écouter votre messagerie en appelant. Ce sont des compromis assumés pour un service téléphonique fonctionnel, pas des trous dans la connaissance nulle.

Les champs chiffrés côté serveur sont liés à leur contexte (par exemple, une adresse E911 est cryptographiquement liée à son propriétaire) pour qu'un texte chiffré ne puisse pas être déplacé d'un enregistrement à l'autre. Étendre ce lien à chaque champ serveur, ainsi qu'une dérivation de clé par utilisateur, figure à notre feuille de route publiée.

## 5. Communications : « aveugle au serveur », pas à confidentialité persistante

Chaque SMS, MMS ou message vocal entrant est scellé à votre clé publique X25519, et on détruit notre paire de clés d'expéditeur dès que le message est stocké. Le résultat : **on ne conserve aucune clé capable de lire vos messages : ni au repos, ni plus tard, ni sous un mandat qui nous vise.** On peut être contraints de produire les octets chiffrés ; on ne peut pas être contraints de produire une clé qu'on n'a pas.

**Ce que ce n'est *pas* :** ce n'est **pas de la confidentialité persistante.** Vos messages sont scellés à la clé à long terme de votre appareil, soit la même clé que votre appareil utilise pour relire votre historique sur un nouvel

appareil. Si *votre* clé privée était compromise, votre historique stocké pourrait être déchiffré. On optimise pour l'**aveuglement au serveur** (un serveur ou un mandat ne peut pas lire votre contenu), qui est la propriété atteignable et utile pour un historique de messages stocké et relisible. La vraie confidentialité persistante entre en conflit avec un historique multi-appareils et n'est pas une chose qu'on revendique. Le durcissement post-quantique de ce sceau figure à notre feuille de route (section 11).

Le transit sur le réseau opérateur (SS7/SIP) est visible par les opérateurs comme tout service téléphonique. Pour un chiffrement de bout en bout *en transit*, utilisez Signal ou WhatsApp avec votre numéro **WIGGWIGG**.

## 6. Authentification

Votre mot de passe ne quitte jamais votre appareil. Aujourd'hui, votre appareil dérive la clé principale (Argon2id) et en calcule une **preuve HMAC-SHA256**. Le serveur ne stocke qu'un haché de cette preuve et le compare à la connexion. Le serveur ne voit jamais votre mot de passe, et une clé principale verrouillée par Argon2id rend la preuve coûteuse à falsifier.

On migre la connexion vers un schéma **défi-réponse Ed25519** où le serveur ne stocke qu'une clé publique et où vous prouvez la possession en signant un défi serveur à usage unique de 60 secondes, pour qu'une fuite du vérificateur en base ne puisse pas être rejouée pour se connecter. C'est construit et en cours de déploiement. On ne revendiquera pas la propriété plus forte « une fuite de base ne peut pas authentifier » tant que le vérificateur hérité n'est pas entièrement retiré.

**Authentification à plusieurs facteurs :** WebAuthn/clés d'accès (résistantes à l'hameçonnage) et TOTP. **Anti-hameçonnage :** avant que vous tapiez votre mot de passe complet, le site se prouve à vous avec un marqueur propre au compte (phrase, couleurs, avatar, audio optionnel) généré côté serveur à partir d'une graine, et une vérification partielle du mot de passe (premiers caractères, HMAC-SHA256) déclenche l'affichage.

## 7. Récupération de compte

La récupération utilise une phrase **BIP39** de 24 mots (256 bits d'entropie) que vous détenez et qu'on ne voit jamais. Votre appareil en dérive une clé de récupération. On ne stocke qu'un haché de la clé publique de récupération et une copie de votre clé principale enveloppée vers cette clé de récupération. La récupération prouve la possession par défi-réponse. Si vous perdez **à la fois** votre mot de passe et votre phrase de récupération, vos données à connaissance nulle sont irrécupérables par conception : on ne peut pas les déchiffrer pour vous. Une réinitialisation assistée par un administrateur peut rétablir l'*accès* à votre compte (numéros de téléphone, facturation) mais seulement avec votre confirmation explicite et distincte, et elle ne récupère pas les données à connaissance nulle.

## 8. Gestion et rotation des clés

Les clés côté serveur (la clé de champ opérationnel, la clé d'index aveugle, la graine de métadonnées par utilisateur) vivent dans un coffre de secrets géré sous forme d'**anneau versionné**, validé au démarrage. Chaque schéma (KDF, clé de coffre, sceau asymétrique, clé serveur) porte une colonne de version, permettant une rotation sans interruption : les nouvelles écritures utilisent la version courante, et les lecteurs s'aiguillent sur la

version stockée. Les colonnes porteuses de matériel cryptographique sont protégées par des déclencheurs de base de données qui refusent de les mettre à nul en silence.

## 9. Sécurité de l'application et de l'infrastructure

- La **sécurité au niveau des lignes** est imposée sur les tables locataires (avec `FORCE ROW LEVEL SECURITY`, une fonction de contexte obligatoire et un garde-fou d'intégration continue) pour qu'un utilisateur ne puisse jamais lire les lignes d'un autre.
- **Transport :** TLS partout ; en-têtes de sécurité (CSP/HSTS) ; CSRF défendue par des témoins SameSite et une liste blanche d'origine stricte.
- **Réseau :** isolation par VPC et NAT par VPC comme défense en profondeur délibérée.
- **Chiffrement au repos :** la base de données, la cache et le stockage objet sont chiffrés avec des clés gérées (AWS KMS), sous le chiffrement applicatif et à connaissance nulle ci-dessus.
- **Défense en profondeur :** un pare-feu applicatif géré protège la bordure publique, avec détection de menaces en continu, journalisation d'audit du plan de contrôle, journaux de flux VPC, et sauvegardes chiffrées et versionnées.
- **L'observabilité est anonyme par défaut :** aucune adresse IP ni agent utilisateur dans les journaux d'audit administrateur, et l'identité de l'utilisateur dans la télémétrie seulement sur consentement explicite. La rétention des métadonnées opérationnelles est de 90 jours.
- **Minimisation des données des fournisseurs :** l'opérateur n'apprend que ce que le routage exige ; le processeur de cartes (Stripe) reçoit des identifiants opaques et des libellés génériques, et le processeur Bitcoin (Blockonomics) ne voit qu'une adresse de paiement et un montant. On ne les interroge jamais à votre sujet.

## 10. Limites et non-objectifs

On les énonce clairement :

- **Appareil compromis.** Un logiciel malveillant ou une extension malveillante sur *votre* appareil peut lire vos données après que vous l'avez déverrouillé. Le chiffrement côté client ne peut pas défendre un client compromis.
- **Aucune confidentialité persistante** sur les messages stockés (section 5).
- **Aucune protection post-quantique** pour l'instant (section 11) : un adversaire quantique futur pourrait déchiffrer les données scellées en X25519 d'aujourd'hui (« récolter maintenant, déchiffrer plus tard »).
- **Les métadonnées du service téléphonique** (qui a appelé qui, quand, combien de temps) sont inhérentes au réseau public et visibles par les opérateurs. On les minimise et les conserve 90 jours, mais on ne peut pas les rendre privées.
- **Les données opérationnelles lisibles par le serveur** (numéros, E911, facturation) ne sont pas à connaissance nulle (section 4), car elles sont requises pour fonctionner.
- **L'authentification est en pleine migration** (section 6), donc la revendication la plus forte attend le retrait du vérificateur hérité.

## 11. Feuille de route

- **Post-quantique (la vedette) :** enveloppement de clé hybride **X25519 + ML-KEM-768** pour le sceau des messages et l'enveloppe de récupération, pour qu'un attaquant doive briser les **deux**. Ça répond directement au « récolter maintenant, déchiffrer plus tard » pour les données stockées à long terme.
- **Authentification :** compléter le déploiement du défi-réponse Ed25519 et retirer le vérificateur hérité, puis évaluer un PAKE asymétrique (OPAQUE).
- **Retirer RSA-2048** (chiffrement des billets de soutien administrateur) au profit du coffre scellé X25519.
- **Élargir la liaison au contexte** (AAD) et la dérivation de clé par utilisateur à tous les champs chiffrés côté serveur.
- **Audit tiers :** une revue cryptographique et applicative externe, prévue.

## 12. Vérification et divulgation

- **Vérifiez la crypto :** github.com/WiggWiggDev/crypto. Les primitives de KDF, de sceau, de signature et de récupération sont exactement celles décrites ici. Le code source ouvert prouve la *conception* et les primitives *côté client* ; à lui seul, il ne prouve pas que nos serveurs les exécutent sans modification, donc des versions client reproductibles et des empreintes de paquet publiées figurent à la feuille de route.
- **Signaler une vulnérabilité :** voir https://wiggwigg.ca/.well-known/security.txt. On offre un refuge de bonne foi pour la divulgation responsable et on répond rapidement.

*Ce document est versionné et daté. Si la réalité et ce document divergent un jour, c'est un bogue dans l'un des deux. Dites-le-nous.*

**WIGGWIGG**

Fait au Québec, Canada · securite@wiggwigg.ca