Condorcet.Vote — Livre blanc du protocole

Conception cryptographique, registre public, format de la blockchain et principes d'implémentation — version 1 du protocole

The Condorcet.Vote project

Julien Boudry (julien.boudry@condorcet.io)

15 novembre 2026 — révision 1.2 (brouillon)

Résumé

Condorcet.Vote est une plateforme web d'organisation d'élections préférentielles (méthodes de Condorcet) et à choix unique sur des listes électorales nominatives fermées. Ce livre blanc explique le fonctionnement de son protocole de vote et ce qu'il garantit.

Trois idées portent la conception :

1. Chaque bulletin est signé dans le navigateur du votant par une signature en anneau chaînable (une variante bLSAG à portée d'élection sur Ristretto255) : le bulletin est prouvablement légitime, anonyme au sein de son anneau, et les doubles votes sont publiquement manifestes — le tout sans compte votant.
2. Chaque élection publie son registre complet — anneaux, émargement pseudonyme et tous les bulletins signés — sous la forme d'une suite de blocs JSON en ajout strict, chaînés par empreinte et signés par la plateforme, que quiconque peut mettre en miroir, re-vérifier avec des outils standard et re-dépouiller.
3. La plateforme elle-même est un tiers de confiance déclaré : digne de confiance pendant le dépôt des votes, auditable de façon indépendante ensuite.

Le document énonce précisément ce qui est protégé et ce qui ne l'est pas, détaille le format des blocs et ce que leur immuabilité signifie — et ne signifie pas —, et présente les arbitrages entre force cryptographique et facilité d'utilisation comme des décisions d'ingénierie factuelles.

Table des matières

1 Introduction **3**
- 1.1 Objet et périmètre . . . 3
- 1.2 Vue d'ensemble du système . . . 3

2 Objectifs de sécurité et modèle de confiance **4**
- 2.1 Garanties . . . 4
- 2.2 Le modèle de tiers de confiance . . . 4
- 2.3 Non-objectifs explicites . . . 4

3 Fondations cryptographiques **5**
- 3.1 Architecture d'implémentation : un cœur Rust unique, exécuté des deux côtés . 5
- 3.2 Inventaire des primitives . . . 5
- 3.3 La signature des bulletins : une variante bLSAG à portée d'élection . . . 6
- 3.4 Preuve de possession (Opération D) . . . 8
- 3.5 Encodages canoniques des chaînes de caractères . . . 9

4 Le protocole de vote **9**
4.1 Cycle de vie d'une élection . . . 9
4.2 Provisionnement des clés : trois modes coexistants . . . 9
4.3 Anneaux . . . 11
4.4 Distribution des clés (mode généré par le serveur) . . . 13
4.5 Le bulletin : une charge utile signée, exacte à l'octet . . . 13
4.6 Dépôt du vote et prévention du double vote . . . 14
4.7 Reçu de participation et vérifiabilité individuelle . . . 15
4.8 Options d'élection aux conséquences explicites sur la vie privée . . . 15

5 Le registre public et sa blockchain **16**
5.1 Deux faces délibérément non reliables . . . 16
5.2 Stockage des bulletins : exact à l'octet et en ajout strict . . . 16
5.3 Règles anti-chronologie . . . 16
5.4 La chaîne : enveloppe et chaînage . . . 17
5.5 Types de blocs . . . 17
5.6 Sérialisation : signer le fichier, pas un modèle canonique . . . 20
5.7 Scellement et ancrage : comment un bloc est produit . . . 21
5.8 Signature de plateforme et gestion des clés . . . 22
5.9 Ancrage temporel externe : OpenTimestamps . . . 22
5.10 Ce que « immuable » veut dire ici — et ce qu'il ne veut pas dire . . . 23
5.11 Ce que quiconque peut vérifier . . . 23
5.12 Reproductibilité du dépouillement . . . 24

6 Modèle de menace **24**
6.1 Menaces couvertes . . . 24
6.2 Limites acceptées — documentées, non dissimulées . . . 25

7 Garde-fous d'implémentation **26**
7.1 Protéger la fenêtre de signature . . . 26
7.2 Minimisation des données et hygiène de chronologie . . . 26
7.3 La base de données comme arbitre unique . . . 27
7.4 Isolation de la publication . . . 27
7.5 Discipline d'ingénierie . . . 27

8 Arbitrages de conception : cryptographie avancée contre facilité d'utilisation **27**
8.1 La vérifiabilité par la publication, non par une cryptographie de dépouillement avancée . . . 27
8.2 Des reçus plutôt que l'absence de reçus . . . 28
8.3 Clés générées par le serveur par défaut, BYO-only comme mode de confiance minimale . . . 28
8.4 Un vote sans compte, par liens de capacité envoyés par e-mail . . . 28
8.5 Une taille d'anneau plafonnée : un plafond, une taille de lancement par élection 28
8.6 Le cache de redistribution . . . 29
8.7 Une plateforme centrale aux preuves mirrorables, pas un registre décentralisé . 29

9 Limites connues et travaux prévus **29**

10 Références **30**

Ce document est la traduction française du document original rédigé en anglais (« Condorcet.Vote — Protocol Whitepaper »).

Statut de ce document. Ce livre blanc est une *source secondaire* : un résumé consolidé des spécifications normatives du projet et de son implémentation (le code source de la plateforme et la bibliothèque `cryptoVote`). Il est descriptif, non normatif. Si quoi que ce soit dans ce document contredit les spécifications ou le code, les spécifications et le code prévalent. Il décrit la **version 1 du protocole**.

1 Introduction

1.1 Objet et périmètre

Ce document explique le **protocole de vote** de Condorcet.Vote et les principes d'implémentation qui le soutiennent : les mécanismes cryptographiques (avec précision), la façon dont les bulletins sont stockés, la façon dont le registre public est formaté, chaîné, signé et vérifié, les garanties qui en découlent, et les menaces qui ne sont explicitement *pas* couvertes. Il s'adresse aux organisateurs à l'aise avec la technique, aux relecteurs en sécurité et aux auditeurs indépendants.

C'est un document de protocole et d'implémentation, pas un audit de sécurité général de la plateforme : la gestion des comptes organisateurs, le durcissement de l'application web et l'exploitation de l'infrastructure sont hors de son périmètre. Seuls les garde-fous d'implémentation qui portent directement une garantie du protocole — comme l'intégrité de l'environnement de signature côté client ou l'effacement de la chronologie au niveau du stockage — sont décrits, et au niveau des principes.

1.2 Vue d'ensemble du système

Condorcet.Vote organise des élections sur des listes électorales nominatives fermées : un organisateur définit des candidats et des votants ; les votants reçoivent un lien d'accès personnel (*VoteLink*) par e-mail et votent depuis leur navigateur sans créer de compte. Les bulletins sont des expressions ordonnées dépouillées par des méthodes de Condorcet (ou à choix unique, scrutin uninominal), calculées par le moteur open-source CondorcetPHP.

Le protocole repose sur trois piliers :

1. **L'anonymat des bulletins par signatures en anneau chaînables.** Chaque bulletin est signé dans le navigateur du votant avec une clé secrète à usage unique dont la clé publique est cachée parmi des centaines d'autres dans un *anneau*. La signature prouve que le bulletin provient d'un membre légitime de l'anneau sans révéler lequel.
2. **La vérifiabilité publique par la publication.** Chaque élection publie un *registre public* — sa définition gelée, tous les anneaux, un registre d'émargement pseudonyme, et chaque bulletin avec sa signature — distribué sous la forme d'une *blockchain* d'artefacts JSON en ajout strict, chaînés par empreinte et signés par la plateforme, que quiconque peut télécharger, mettre en miroir et re-vérifier avec des outils standard. Quiconque peut recalculer le dépouillement.
3. **Un modèle de tiers de confiance déclaré.** La plateforme est digne de confiance au moment du vote (elle voit nécessairement un bulletin arriver sur une session authentifiée) et n'est *pas* digne de confiance ensuite : tout ce qu'elle a fait est vérifiable à partir du seul registre publié.

2 Objectifs de sécurité et modèle de confiance

2.1 Garanties

Le protocole fournit, simultanément et sans compte votant :

— **La confidentialité des bulletins.** Aucun membre du public, aucun organisateur, aucun lecteur a posteriori d'un export logique de la base de données ne peut relier un bulletin stocké à un votant. L'ensemble d'anonymat cryptographique d'un bulletin est son anneau ; l'ensemble d'anonymat effectif est constitué des membres *participants* de l'anneau (voir §4.3 et §6).

— **La vérifiabilité publique.** Quiconque peut vérifier, à partir du seul registre publié : que chaque bulletin porte une signature en anneau valide, qu'aucune clé n'a signé deux fois, que les bulletins ne sont pas plus nombreux que les votants enregistrés, et que le résultat publié est le dépouillement déterministe des bulletins publiés (voir §5.11).

— **La résistance au double vote.** Un bulletin effectif par votant, imposé transactionnellement par le registre d'émargement et attesté publiquement par les *images de clé* cryptographiques (voir §4.6).

2.2 Le modèle de tiers de confiance

La plateforme est un **tiers de confiance dont la confiance est bornée dans le temps** :

— **Digne de confiance au moment du vote.** La plateforme sert la page de vote, délivre les clés secrètes dans son mode de provisionnement par défaut, et reçoit chaque bulletin signé sur la session VoteLink authentifiée du votant. Pendant cette fenêtre, elle sait transitoirement qui vote et, dans le mode par défaut, quelle clé elle a remise à qui. Cette exposition est une décision de conception délibérée et documentée, pas un oubli (§8).

— **Non digne de confiance ensuite.** Chaque affirmation de la plateforme est matérialisée dans le registre public et vérifiable de façon indépendante. Les deux faces du registre — émargement et bulletins — sont publiées de sorte qu'elles ne puissent pas être jointes en un lien votant-bulletin.

Le niveau de confiance accordé au moment du vote est lui-même un **curseur au niveau de l'élection** : une élection peut être configurée en *BYO-only* (§4.2), auquel cas la plateforme ne détient jamais le secret de signature d'aucun votant et la fenêtre de confiance se réduit à la seule réception des bulletins.

2.3 Non-objectifs explicites

Les points suivants sont consciemment **hors périmètre** pour la version 1 du protocole et sont traités en détail en §6 et §8 :

— **Résistance à la coercition / absence de reçu (receipt-freeness).** Les votants obtiennent des reçus prouvables ; les bulletins publiés sont des classements en clair. Le protocole vise les élections privées et associatives, pas les scrutins publics sensibles à la coercition.

— **Décentralisation.** La chaîne est en ajout strict et chaînée par empreinte, mais produite par un opérateur unique ; l'immuabilité repose sur la mise en miroir indépendante (§5.10), complétée par un ancrage temporel par bloc dans Bitcoin via OpenTimestamps (§5.9) — une preuve d'existence, pas un consensus distribué.

— **Fonctionnement sans confiance (« trustless »).** La conception ne cherche pas à retirer la plateforme de l'équation de confiance au moment du vote (pas de réseaux de mélange, pas de dépouillement homomorphe ou à divulgation nulle de connaissance).

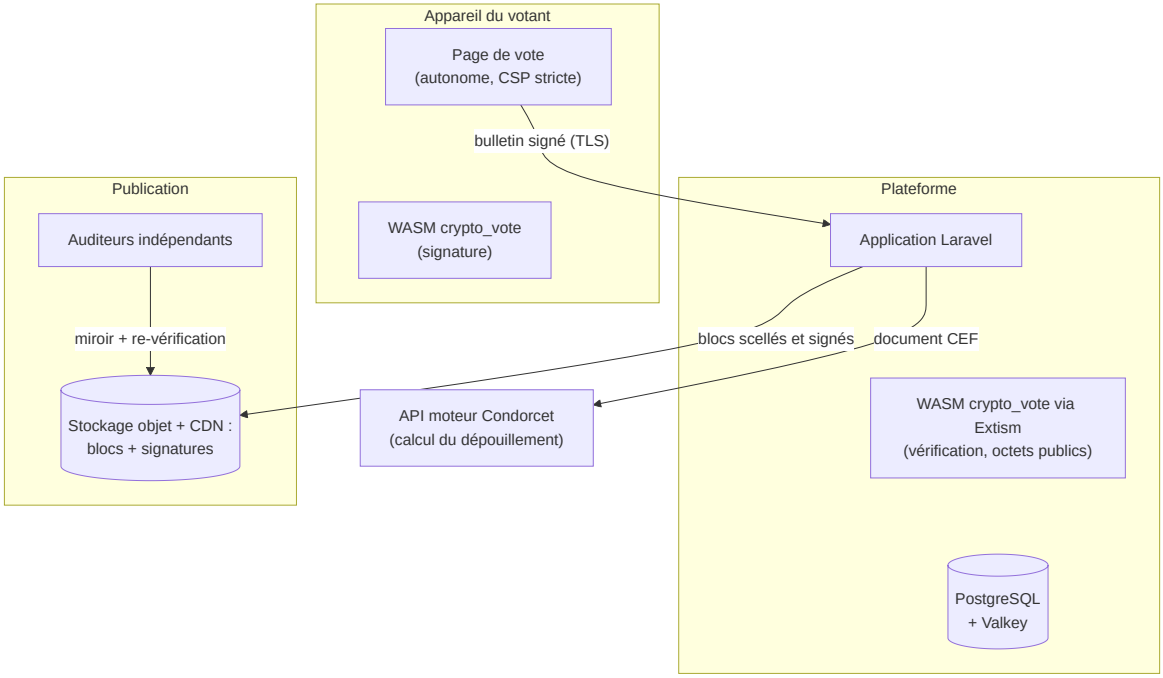
3 Fondations cryptographiques

3.1 Architecture d'implémentation : un cœur Rust unique, exécuté des deux côtés

Toute la cryptographie des bulletins est implémentée une seule fois, dans un crate Rust open-source dédié — `crypto_vote` (dépôt *cryptoVote*, AGPL-3.0-or-later) — et compilée en WebAssembly pour les deux environnements d'exécution. La même logique, à l'octet près, s'exécute donc dans le navigateur et sur le serveur ; la plateforme ne réimplémente jamais une primitive en PHP ou en JavaScript.

— **Appareil du votant — signature.** Le paquet npm `@condorcet.vote/crypto-vote` (un module ES `wasm-bindgen`) s'exécute dans le navigateur, sur une page de vote autonome servie sous la CSP la plus stricte de la plateforme.
— **Plateforme — vérification.** Le paquet Composer `condorcet-vote/crypto-vote-php` enveloppe un build Extism (`wasm32-wasip1`) du même cœur et l'exécute in-process sur le serveur ; il ne manipule que des octets publics. Il n'existe aucun micro-service cryptographique séparé.

Le crate expose un oracle délibérément minimal de quatre opérations : `generate_identity`, `sign_vote`, `verify_vote`, et la preuve de possession à l'initiative du votant (§3.4). Il ne sait rien des élections, des bases de données ou du réseau ; la comptabilité anti-double-vote appartient à la plateforme.



3.2 Inventaire des primitives

Usage	Primitive	Paramètres
Groupe	Ristretto255 (groupe d'ordre premier sur Curve25519)	ordre premier $\ell \approx 2^{252}$; `curve25519-dalek`
Signature des bulletins	Signature en anneau chaînable — variante bLSAG à portée d'élection	Fiat–Shamir, signatures de $32(1+n)$ octets pour un anneau de taille n
Hachage vers scalaire / vers point	BLAKE2b-512	réduction large 512 bits mod ℓ ; application univoque Ristretto
Image de clé (étiquette de chaînabilité)	$I = x \cdot H_p(\text{domaine} \parallel \text{len}(e) \parallel e \parallel P)$	à portée d'élection ; 32 octets
Preuve de possession (Opération D)	DLEQ Chaum–Pedersen, non interactive	preuve de 64 octets ; nonce frais de 32 octets
Somme de contrôle des chaînes d'artefacts	BLAKE3, tronquée à 4 octets	garde anti-faute de frappe uniquement — jamais une primitive de sécurité
Empreinte des blocs de la chaîne	SHA3-512 sur les octets stockés du bloc	hexadécimal minuscule, 128 caractères
Empreinte e-mail d'émargement (opt-in)	SHA3-256(id d'élection ‖ e-mail normalisé)	pseudonymisation délibérément faible, voir §4.8
Signature des blocs de la chaîne	OpenPGP v6 (RFC 9580), Ed25519, détachée	produite par la CLI Sequoia `sq`
Jeton VoteLink au repos	Empreinte SHA3-256	le jeton en clair n'existe que dans l'URL envoyée par e-mail
Secrets stockés au repos	AES-256-GCM (chiffrement par clé applicative)	clés secrètes de vote, avant distribution
Aléa	CSPRNG du système	famille `getrandom(2)` ; Web Crypto `getRandomValues` dans les navigateurs ; WASI `random_get`

Notation utilisée ci-dessous : G est le point de base de Ristretto255 ; ℓ l'ordre du groupe ; H_s le hachage vers un scalaire (BLAKE2b-512 puis réduction large) ; H_p le hachage vers un élément du groupe (BLAKE2b-512 puis l'application univoque de Ristretto) ; enc$(\cdot)$ l'encodage canonique compressé de 32 octets d'un point ; $\text{len}_8(\cdot)$ un préfixe de longueur de 64 bits gros-boutiste ; $\parallel$ la concaténation d'octets.

3.3 La signature des bulletins : une variante bLSAG à portée d'élection

Le schéma est une **signature de groupe anonyme spontanée chaînable (LSAG)** dans la forme *back-linkable* (bLSAG) documentée dans la littérature Monero, dont les équations d'anneau suivent Liu–Wei–Wong [LWW04] tel que présenté dans *Zero to Monero* [ZtM20].

Elle est de type Schnorr : le maillon du signataire dans l'anneau utilise la réponse de Schnorr $s_\pi = \alpha - c_\pi x$, chaînée autour de tous les membres de l'anneau de sorte que la signature se vérifie pour l'anneau entier sans révéler quel membre a signé.

Génération de clé. Une identité de vote est un scalaire secret uniformément aléatoire $x \in [1, \ell)$ de clé publique $P = xG$. Il n'y a ni « clamping » ni schéma de dérivation de clé ; le secret est généré par le CSPRNG du système (dans le navigateur du votant pour les clés auto-enrôlées, sur la plateforme pour les clés générées par le serveur).

Base de chaînabilité à portée d'élection. Pour une clé publique P et un identifiant d'élection e (l'UUID de l'élection, normalisé NFC), le schéma définit un point de base par membre :

$$B_P \;=\; H_p(\texttt{"crypto_vote:blsag:key-image-point"} \,\|\, \mathrm{len}_8(e) \,\|\, e \,\|\, \mathrm{enc}(P))$$

L'**image de clé** du signataire est $I = x \cdot B_P$. Le bLSAG standard utilise $I = x \cdot H_p(P)$; incorporer l'identifiant d'élection dans la pré-image du hachage vers un point est la *seule* déviation de ce schéma par rapport à la construction standard. Son but : la même clé secrète produit la *même* image de clé au sein d'une élection (chaînabilité intra-élection — les doubles votes avec la même clé sont publiquement manifestes) mais des images de clé *non reliables* entre élections différentes (sous l'hypothèse décisionnelle de Diffie–Hellman dans le groupe Ristretto255, les images de clé d'une même clé dans deux élections ne peuvent pas être corrélées sans x).

Liaison du message. Le message signé n'est pas le bulletin brut mais le blob lié à l'élection

$$m \;=\; \mathrm{len}_8(e) \,\|\, e \,\|\, v$$

où v est constitué des octets de la charge utile du bulletin (§4.5), traités comme opaques et jamais normalisés. Le préfixe de longueur élimine l'ambiguïté de concaténation entre e et v. Seul e est normalisé NFC (sans effet pour les UUID ASCII utilisés en pratique).

Canonicalisation de l'anneau. Un anneau est l'ensemble des $n \geq 2$ clés publiques autorisées, incluant celle du signataire. Le signataire et le vérificateur trient indépendamment l'anneau lexicographiquement par encodage compressé de 32 octets, de sorte que l'ordre de transmission est sans importance ; les membres dupliqués sont rejetés, tout comme les encodages de point non canoniques et l'élément neutre.

Signature (signataire à l'indice π après canonicalisation) :

1. Tirer $\alpha \xleftarrow{\$} \mathbb{Z}_\ell$ et des réponses leurres $s_i \xleftarrow{\$} \mathbb{Z}_\ell$ pour tout $i \neq \pi$.
2. Amorcer la chaîne de défis : $c_{\pi+1} = H_s(D_c \,\|\, \mathrm{len}_8(m) \,\|\, m \,\|\, \mathrm{enc}(\alpha G) \,\|\, \mathrm{enc}(\alpha B_{P_\pi}))$ avec la chaîne de domaine $D_c = \texttt{"crypto_vote:blsag:challenge"}$.
3. Pour chaque membre suivant $i = \pi+1, \pi+2, \ldots$ (indices mod n), calculer $L_i = s_i G + c_i P_i$, $R_i = s_i B_{P_i} + c_i I$, et $c_{i+1} = H_s(D_c \,\|\, \mathrm{len}_8(m) \,\|\, m \,\|\, \mathrm{enc}(L_i) \,\|\, \mathrm{enc}(R_i))$, en s'arrêtant une fois c_π produit.
4. Fermer l'anneau : $s_\pi = \alpha - c_\pi x \bmod \ell$.

La signature est $\sigma = (c_0, s_0, \ldots, s_{n-1})$ accompagnée de l'image de clé I — $32(1+n)$ octets plus 32 octets d'image de clé.

Vérification. Étant donné $(\sigma, I, \text{anneau}, e, m)$: recalculer la chaîne de défis depuis c_0 sur les n membres avec les mêmes équations et accepter si et seulement si elle se referme sur c_0. Notons

que chaque étape de vérification recalcule la base par membre B_{P_i}, de sorte que l'identifiant d'élection est lié à chaque maillon de la chaîne deux fois : par le message m et par les bases de chaînabilité. La vérification est une fonction totale : des entrées malformées (longueurs incorrectes, encodages non canoniques, anneau discordant) renvoient *faux* plutôt que de lever des erreurs.

Propriétés de sécurité :

- *Anonymat :* le vérificateur apprend seulement qu'*un* membre de l'anneau a signé ; la distribution des valeurs (c, s_i) est identique pour chaque signataire possible.
- *Chaînabilité :* deux signatures par la même clé secrète dans la même élection portent nécessairement la même image de clé I.
- *Non-calomnie / infalsifiabilité :* produire une signature valide ou une image de clé en collision sans le secret d'un membre de l'anneau se réduit au problème du logarithme discret dans Ristretto255 (arguments LSAG standard).
- *Non-reliabilité inter-élections :* ajoutée par la variante, argumentée sous DDH comme ci-dessus.

Notes d'implémentation. L'arithmétique des scalaires et des points est à temps constant (`curve25519-dalek`) ; le tri de l'anneau et la recherche d'indice n'opèrent que sur des données publiques. Les scalaires secrets sont mis à zéro à leur destruction ; la frontière navigateur transmet le secret dans un `Uint8Array` mutable que l'appelant efface. Le crate embarque des vecteurs de test figés qui verrouillent les chaînes de domaine, le format de liaison et les octets de signature/image de clé contre toute dérive silencieuse du protocole, ainsi que des balayages de malléabilité bit à bit, des cibles de fuzzing sur tous les parseurs et un `cargo audit` en CI.

3.4 Preuve de possession (Opération D)

Une opération optionnelle, **à l'initiative du votant**, permet au détenteur d'une clé secrète de prouver à tout tiers qu'une image de clé donnée — donc un bulletin publié donné — est la sienne, *sans révéler la clé secrète*. C'est une **preuve Chaum–Pedersen d'égalité de logarithmes discrets** non interactive [CP92] : la connaissance d'un scalaire x tel que $P = xG$ et $I = xB$, où B est la même base à portée d'élection que ci-dessus.

- Le vérificateur fournit un nonce frais de 32 octets η (fraîcheur, pas vérification désignée).
- Prouveur : $r \xleftarrow{\$} \mathbb{Z}_\ell$; $R_1 = rG$; $R_2 = rB$; $c = H_s(D_o \parallel \text{len}_8(e) \parallel e \parallel \text{len}_8(\eta) \parallel \eta \parallel \text{enc}(P) \parallel \text{enc}(B) \parallel \text{enc}(I) \parallel \text{enc}(R_1) \parallel \text{enc}(R_2))$; $s = r + cx$. La preuve est (c, s) — 64 octets. $D_o =$ `"crypto_vote:ownership:challenge"` est distinct du domaine de la signature en anneau, ce qui empêche tout rejeu inter-protocole.
- Vérificateur : recalculer $R'_1 = sG - cP$, $R'_2 = sB - cI$, re-dériver le défi et comparer.

Deux propriétés sont énoncées sans détour parce qu'elles importent aux votants : la preuve est **transférable** (quiconque détient le nonce et la preuve peut convaincre n'importe qui d'autre — ce n'est pas une preuve à vérificateur désigné), et son usage **révèle comment le prouveur a voté** (elle relie le prouveur à un bulletin publié en clair). Elle n'ajoute aucune exposition à la coercition que le reçu (§4.7) n'avait pas déjà créée ; elle standardise une capacité existante pour des usages légitimes comme le vote mandaté ou par procuration. Elle est servie par une page d'outils publique, s'exécute entièrement côté client, et échappe à tout contrôle de l'organisateur.

3.5 Encodages canoniques des chaînes de caractères

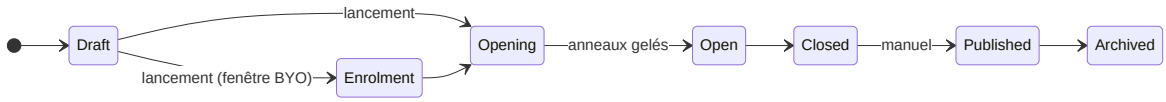
Chaque artefact cryptographique traversant une interface, une API ou une frontière de fichier utilise une forme canonique préfixée et munie d'une somme de contrôle :

`<tag>_<corps hexadécimal minuscule>_<8 caractères hexa de somme de contrôle>`

avec les étiquettes `pk` (clé publique), `sk` (clé secrète), `ki` (image de clé), `blsag` (signature en anneau), `own` (preuve de possession), `nonce` (nonce de preuve). La somme de contrôle est constituée des 4 premiers octets de `BLAKE3("crypto_vote/prefixed-checksum/v1" || tag || 0x00 || payload)`, de sorte qu'une valeur mal saisie — ou un `pk_` collé là où un `ki_` était attendu — est rejetée immédiatement. La somme de contrôle est uniquement une garde d'intégrité et de confusion de type ; l'authenticité provient toujours de la preuve bLSAG. Les frontaux WASM, Extism et CLI acceptent exclusivement cette forme et rejettent l'hexadécimal nu.

4 Le protocole de vote

4.1 Cycle de vie d'une élection



Une élection peut être *annulée* depuis tout état pré-terminal ; l'annulation retire les vues publiques mais ne peut pas rétracter les blocs de chaîne déjà mis en miroir (§5.5). L'état `Opening` est une construction de provisionnement transitoire, tout-ou-rien : les clés côté serveur sont frappées, mises en commun avec les clés publiques importées et enrôlées, mélangées, partitionnées en anneaux, et gelées. La transition vers `Open` est conditionnée à la confirmation du gel de chaque anneau — une construction échouée ne produit jamais d'élection à moitié ouverte. À `Closed`, chaque secret encore stocké est purgé. Pendant `Open`, la liste d'une élection autonome qui autorise les clés serveur peut encore **grandir** : des votants peuvent être ajoutés — jamais modifiés ni retirés — chaque ajout étant définitif, provisionné par le serveur par construction (un anneau gelé ne peut accueillir une clé apportée, raison pour laquelle une élection BYO-only refuse les ajouts), livré son lien immédiatement, et absorbé par la marge des anneaux et les anneaux d'urgence de §4.3.

4.2 Provisionnement des clés : trois modes coexistants

Mode	Clé secrète détenue par	Ce qu'il apporte
Générée par le serveur (défaut)	La plateforme (chiffrée au repos), puis le votant après une distribution explicite	Vote sans friction ; exige de faire confiance à la plateforme au moment de la distribution

Mode	Clé secrète détenue par	Ce qu'il apporte
Importée par l'organisateur	Le votant (la plateforme ne voit jamais le secret)	La plateforme n'a jamais connu le secret et ne peut jamais signer pour ce votant. L'organisateur a fourni la clé publique et peut — ou non — détenir aussi le secret ; le protocole ne peut le prouver ni dans un sens ni dans l'autre
Enrôlée par le votant (apportez votre clé)	Le votant (générée dans son propre navigateur)	Le plus fort : la plateforme n'a jamais connu le secret et ne peut jamais signer pour ce votant, et le secret n'a jamais existé hors de l'appareil du votant — l'organisateur ne l'a jamais détenu non plus

Les trois modes coexistent au sein d'une même élection — une **élection par défaut est mixte par construction** : chaque votant est, individuellement, provisionné par le serveur, importé par l'organisateur ou auto-enrôlé, et le mode généré par le serveur est simplement le repli pour les votants qui n'apportent pas de clé. Le mode d'un votant est gelé à la construction `Opening`, sans repli inter-mode (une clé auto-gérée perdue signifie pas de vote, exactement comme un absent — la clé publique inutilisée reste dans l'anneau comme un rembourrage inoffensif). Les clés publiques sont uniques par liste électorale, et les clés en attente sont chiffrées au repos avec une empreinte SHA3-256 en clair comme ancre d'unicité. Le mode n'est pas une comptabilité privée : il est **publié par votant** dans le registre d'émargement final (§5.5), de sorte que les votants pour lesquels la plateforme a un jour détenu une clé sont eux-mêmes un fait public et auditable.

4.2.1 Élections BYO-only : retirer à la plateforme la garde des clés

L'enrôlement obligatoire (*BYO-only*) est un **mode d'élection de premier rang**, pas une option annexe : un unique réglage effectué au stade du brouillon rend la fenêtre d'enrôlement obligatoire, et chaque votant doit alors apporter une clé importée ou en enrôler une — le navigateur génère la paire de clés localement (WASM) et remet au votant un fichier de clé à conserver. **Aucune clé côté serveur n'est jamais générée** : la plateforme ne détient jamais le secret de signature d'aucun votant.

Le mode est trivial à activer pour l'organisateur et plus exigeant pour les votants — chacun est responsable de la garde d'une clé, et une clé perdue signifie pas de vote, exactement comme un absent. En échange, il *élimine structurellement* plusieurs des arbitrages que le mode par défaut accepte :

Propriété	Élection par défaut (mixte ; clés serveur autorisées)	Élection BYO-only
La plateforme détient des secrets de vote	Pour les votants provisionnés par le serveur — chiffrés, effacés à la distribution	Jamais
La plateforme pourrait signer à la place d'un votant	Possible pour les seules clés générées par le serveur	Structurellement impossible
Précalcul des images de clé par une plateforme malveillante (§6.2)	Possible pour les seules clés générées par le serveur	Structurellement impossible
Machinerie de distribution — plafond de 3 clés, réclamations (§4.4)	Requise	Absente : rien à distribuer
Liaison transitoire votant-clé en cache (§8.6)	Existe pour les votants provisionnés par le serveur, jusqu'à 180 minutes	N'existe jamais
Composition des anneaux	Une taille de lancement par élection (100 à 600 clés) rembourrée de clés serveur ; anneaux d'urgence au plafond de 600 ; affectation probabiliste	Effectif connu ; partition déterministe et équilibrée ; plafond seulement
Risque d'anneau clairsemé (§6.2)	Atténué de façon probabiliste (anneaux d'urgence, plancher d'anonymat)	Structurellement minimisé par la partition équilibrée
Charge pour le votant	Optionnelle — une clé est délivrée à la demande sauf si le votant en apporte une	Le votant génère, conserve et présente sa propre clé ; la perte signifie pas de vote
Exposition votant-bulletin à la soumission	La plateforme la voit transitoirement	Inchangée — inhérente à la soumission authentifiée

Ce que BYO-only ne change **pas** est tout aussi explicite : la plateforme voit toujours chaque bulletin arriver sur une session authentifiée (dernière ligne du tableau), les reçus restent prouvables, et le modèle de publication est identique — les garanties côté public (anneaux, images de clé à portée d'élection, bornes du registre) sont les mêmes dans les deux modes.

4.3 Anneaux

— **Un plafond par élection, matérialisé.** Le plafond de taille d'anneau (`max_ring_size`, actuellement 600) est copié sur l'élection à la construction `Opening` ; aucun anneau de cette élection ne le dépasse jamais, et chaque anneau créé ensuite lit la valeur matérialisée, de sorte qu'un changement ultérieur de la constante du code n'affecte jamais une élection en cours. À $n = 600$, la liste des clés publiques d'un anneau pèse environ 19 Ko, une signature environ 19 Ko, et signer ou vérifier prend grossièrement 0,5–0,6 s de

temps mono-cœur — une enveloppe de coût délibérée (§8.5) ; au plancher de 100 clés, chaque chiffre est environ six fois plus petit.

— **Provisionnement au lancement.** Pour une liste électorale de N votants, l'élection ouvre avec $R_0 = \max(1, \lceil 1{,}5 \cdot N / \text{max_ring_size} \rceil)$ anneaux : 1,5 clé par votant modélise la demande attendue (la plupart des votants réclament une seule clé ; la participation est inférieure à 100 %), pas le pire cas du plafond de 3 distributions.

— **Une taille de lancement par élection, sous le plafond.** Tous les anneaux de lancement partagent une taille unique dérivée de la liste gelée — N_s votants sans clé importée ni enrôlée, N_b clés importées ou enrôlées :

$$\text{taille_lancement} \;=\; \text{clamp}\Big(\Big\lceil \tfrac{3 \cdot 2 \cdot N_s + N_b}{R_0} \Big\rceil,\; 100,\; \text{max_ring_size}\Big)$$

$3 \cdot N_s$ est la demande de redistribution au pire cas (chaque votant provisionné par le serveur brûlant ses trois essais), le facteur 2 laisse de la place aux votants ajoutés en cours d'élection (une marge réservée au lancement : le provisionnement en cours d'élection se dimensionne sur la demande qu'il voit et n'anticipe jamais une vague suivante), et chaque clé connue occupe une place. Cinquante votants serveur et cinq clés enrôlées ouvrent sur un anneau de 305 clés ; trois votants ouvrent sur le plancher de 100 ; à partir de 100 votants serveur, l'anneau fait les 600 pleins. Dès que la cible tient sous le plafond, la capacité couvre le pire cas : à liste gelée, une telle élection ne peut jamais avoir besoin d'un anneau d'urgence, quoi que ses votants brûlent.

— **Le plancher est de la marge, pas de l'anonymat.** Les clés de rembourrage n'ajoutent jamais d'anonymat (§4.3, dernier point) : une élection à trois votants a un ensemble d'anonymat de trois, quoi que contienne son anneau. Le plancher de 100 clés garde de la place dans un petit anneau de lancement pour que des votants ajoutés plus tard *diluent* ses premiers participants — l'affectation pondérée par déficit continue de lui envoyer une partie des nouvelles distributions — au lieu de déborder directement dans un anneau d'urgence.

— **Affectation.** Au moment de la distribution, un anneau est choisi parmi les anneaux actifs disposant de clés libres, avec une probabilité proportionnelle au déficit de distribution de chaque anneau, plancher uniforme compris. Avec k anneaux actifs, des déficits $I_i = D_{max} - D_i$ et $S = \sum_j I_j$:

$$P_i \;=\; 0{,}25 \cdot \tfrac{1}{k} \;+\; 0{,}75 \cdot \tfrac{I_i}{S} \qquad (S > 0;\ \text{uniforme si } S = 0)$$

Un anneau fraîchement créé capte donc l'essentiel des distributions suivantes et franchit organiquement son seuil d'anonymat utile, puis redescend vers l'uniforme.

— **Anneaux d'urgence : toujours au plafond, autant qu'il en faut d'un coup.** Dès que les clés libres passeraient sous la demande restante projetée (votants provisionnés par le serveur n'ayant pas voté fois un facteur 1,3) — délibérément tôt, parce qu'un anneau clairsemé coûte cher et qu'un anneau supplémentaire est bon marché — un job en file d'attente calcule combien d'anneaux pleins de `max_ring_size` clés le déficit exige, borné par le plafond dur de 3 clés par votant sur la liste *courante*, et les crée tous dans une seule transaction. Une urgence est un événement anormal (une liste qui grossit en cours d'élection), elle prend donc la marge maximale : une élection à trois votants qui reçoit soudain 2 000 votants obtient cinq anneaux de 600 clés en une seule passe, pas un filet de petits anneaux dont chacun serait un bloc `rings`, un signal de postériorité et

un petit ensemble d'anonymat. La distribution ne produit jamais d'anneau partiel ou sous-dimensionné.

— **Plancher d'anonymat.** L'ensemble d'anonymat effectif d'un anneau est constitué de ses membres *participants*, pas de son nombre de clés. La conception vise un minimum de 25 participants par anneau ; cette cible est une justification de conception (elle a façonné la formule d'affectation et le déclencheur d'urgence précoce), pas un paramètre d'exécution, et la clôture d'une élection n'est jamais bloquée par elle. Les clés de rembourrage ajoutent de la marge contre les attaques d'assèchement d'anneau ; elles n'ajoutent aucun anonymat.

4.4 Distribution des clés (mode généré par le serveur)

— Les secrets pré-générés sont stockés chiffrés (AES-256-GCM sous la clé applicative) et sont **supprimés au moment de la distribution** : la réclamation d'une clé libre aléatoire (une sélection `FOR UPDATE SKIP LOCKED`) et la mise à néant de son chiffré stocké se font en une seule instruction, dans une transaction courte qui impose aussi le plafond par votant via une écriture gardée, insensible aux courses. Aucune correspondance durable votant-clé n'est jamais stockée.

— La distribution exige une action explicite du votant, est plafonnée à **3 clés par votant** par défaut (des rallonges accordées par l'organisateur sont possibles et publiquement visibles dans le registre), et le secret est délivré une seule fois, sur TLS.

— Par commodité de re-délivrance, le secret distribué vit au plus **180 minutes** dans une entrée de cache chiffrée, indexée par votant (Valkey), renvoyée à l'identique si le votant recharge la page de vote avant d'avoir voté ; l'entrée est supprimée immédiatement quand un bulletin est accepté. Pendant sa durée de vie, ce cache *est* une liaison transitoire votant-clé — un arbitrage d'utilisabilité explicitement accepté au sein du modèle de confiance au moment du vote (§8.6).

— La clôture de l'élection met en file une purge de chaque secret encore stocké ; la garantie « aucune clé délivrée après la clôture » repose sur la barrière d'état, pas sur le déclenchement de la purge.

4.5 Le bulletin : une charge utile signée, exacte à l'octet

Le bulletin est un document JSON **construit exactement une fois dans le navigateur du votant**, signé à l'octet près, transmis inchangé et stocké tel quel (§5.2). Les octets stockés sont le bulletin canonique — « ce que vous signez est ce qui est stocké est ce qui est vérifié ».

```
{
  "v": 1,
  "election": "0c2e7e2a-5df3-4c0e-9d2b-3f6a8f0c9e21",
  "ballot": "Annecy > Bourget = Leman",
  "weight": 1
}
```

— `v` — la version de protocole de l'élection (actuellement 1).

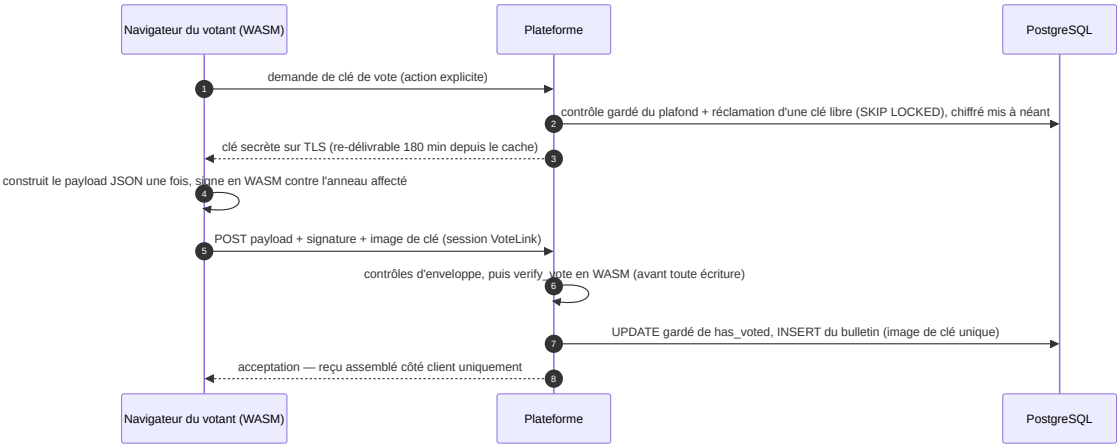
— `election` — l'UUID de l'élection ; également le paramètre de signature `election_id`.

— `ballot` — l'expression du votant sous forme de chaîne UTF-8 plate en syntaxe de classement CEF (étiquettes de candidats avec `>` et `=`), ou la sentinelle `/EMPTY_RANKING/` pour un blanc explicite.

— `weight` — toujours présent, entier ≥ 1 ; le poids de bulletin faisant autorité pour le votant, validé contre la liste électorale à la soumission.
— `voter` — présent uniquement dans les élections non anonymes (§4.8) : l'identifiant de vote public du votant.

Le serveur valide l'**enveloppe** (validité et platitude du JSON, plafond de 8192 octets, concordance de version et d'élection, autorisation du poids, concordance du votant en mode non anonyme) mais ne valide délibérément jamais le contenu de `ballot` : une expression malformée est classée de façon déterministe comme bulletin nul au dépouillement plutôt que rejetée — la règle de classification est publiée et reproductible (§5.12). Aucune canonicalisation JSON n'est imposée ; la signature couvre les octets exacts, et le chemin de sérialisation vers le dépouillement rejette chaque caractère réservé qui pourrait introduire en contrebande des poids ou des votes supplémentaires dans le document CEF.

4.6 Dépôt du vote et prévention du double vote



Les contrôles côté serveur s'exécutent dans cet ordre : l'élection est `Open` ; l'enregistrement de participation du votant indique encore `has_voted = false` ; l'enveloppe est valide ; l'image de clé est inconnue (une contrainte d'unicité en base sur (élection, image de clé) est le filet de sécurité insensible aux courses) ; la signature en anneau se vérifie contre l'anneau désigné par la soumission. L'acceptation bascule alors `has_voted` par une mise à jour conditionnelle gardée et insère la ligne de bulletin — UUID aléatoire, octets de la charge utile tels quels, signature, image de clé, identifiant d'anneau, **aucun horodatage** — en une seule transaction.

Deux mécanismes se partagent délibérément la charge du double vote :

— Le **registre d'émargement est la garde principale** : un votant peut détenir plusieurs clés valides (plusieurs distributions, ou une distribution plus une clé enrôlée), et les images de clé ne peuvent pas relier des bulletins signés avec des clés *différentes*. Seul le drapeau `has_voted` porté par le votant ferme ce cas, et c'est pourquoi la soumission des bulletins est couplée à la session VoteLink et ne peut pas être une API de dépôt anonyme.
— L'**image de clé est la preuve publique, mathématique** : au sein du registre, l'unicité des images de clé prouve qu'aucune *clé* n'a été comptée deux fois — un contrôle que tout tiers peut exécuter sans faire confiance à la plateforme (§5.11).

Un bulletin soumis est définitif : pas de re-vote, pas de modification.

4.7 Reçu de participation et vérifiabilité individuelle

Une fois le bulletin accepté, le navigateur — et lui seul — assemble un reçu contenant l'UUID de l'élection, l'identifiant d'anneau, l'identifiant de vote, la paire de clés du votant, et la signature et l'image de clé du bulletin. Il est affiché une seule fois avec un téléchargement de fichier optionnel, jamais envoyé par e-mail, jamais stocké côté serveur, et impossible à réémettre (la plateforme ne détient plus le secret). Le fichier téléchargé est un unique objet JSON UTF-8 (`.condorcet_receipt`), chaque valeur cryptographique en forme canonique préfixée et la charge utile signée transportée **telle quelle** :

```
{
  "version": "condorcet-receipt/1",
  "election": "0c2e7e2a-5df3-4c0e-9d2b-3f6a8f0c9e21",
  "ring": "<identifiant d'anneau>",
  "voter_id": "<l'identifiant de vote du votant>",
  "public_key": "pk_...",
  "secret_key": "sk_...",
  "ballot": {
    "signed_payload": "<la chaîne JSON exacte qui a été signée, à l'octet près>",
    "signature": "blsag_...",
    "key_image": "ki_..."
  }
}
```

Avec la clé secrète, le votant peut recalculer son image de clé, localiser *son* bulletin dans le registre publié et en comparer les octets — vérifiabilité individuelle, y compris la re-vérification hors ligne de la signature. La page de vote ne garde le secret en mémoire que jusqu'à la navigation et offre une action permanente « masquer les détails » pour les appareils partagés.

La conséquence délibérée : les reçus sont *prouvables* (voir l'Opération D), donc le protocole n'offre **aucune résistance à la coercition** — une décision acceptée, discutée en §8.2.

4.8 Options d'élection aux conséquences explicites sur la vie privée

Trois options à la main de l'organisateur, toutes désactivées par défaut et gelées au lancement, échangent de la vie privée contre des exigences institutionnelles. Chacune déclenche un avertissement explicite au votant sur la page de vote :

- La **divulgation nominative de participation** publie, pour les votants ayant effectivement voté, leur nom en clair et une empreinte e-mail à portée d'élection (SHA3-256 de l'identifiant d'élection et de l'adresse normalisée). Les abstentionnistes ne sont jamais divulgués. L'empreinte est une pseudonymisation délibérément faible — quiconque détient déjà un e-mail candidat peut confirmer la participation ; son objet est l'audit de la liste électorale, pas la dissimulation.
- Le **vote non anonyme** incorpore l'identifiant de vote dans la charge utile signée, reliant publiquement chaque bulletin à un identifiant de vote ; il force l'activation de la divulgation nominative, complétant une chaîne publique bulletin-vers-nom. Destiné aux votes attribuables par conception (p. ex. assemblées de copropriété). L'identifiant de vote incorporé est validé par le serveur, pas prouvé cryptographiquement — une signature en anneau cache le signataire, donc le registre seul ne peut pas prouver cette affirmation ; c'est documenté plutôt que caché.

— Le **vote pondéré** attache un poids entier à chaque votant, publié sur les deux faces du registre et lié à l'intérieur de chaque bulletin signé ; le dépouillement pondéré est vérifiable par égalité de multi-ensembles sans relier les bulletins aux votants. Dans une élection pondérée *anonyme*, une valeur de poids quasi unique ré-identifie son bulletin (k-anonymat parmi les poids égaux) — l'organisateur en est averti en conséquence.

5 Le registre public et sa blockchain

5.1 Deux faces délibérément non reliables

Chaque élection publie un **registre public** constitué de : (1) sa définition gelée ; (2) la liste complète des clés publiques de chaque anneau ; (3) un **registre d'émargement** — une ligne par votant, identifié uniquement par un identifiant de vote UUIDv4 aléatoire, portant son mode de provisionnement de clé (`server_generated`, `organizer_imported` ou `voter_enrolled` — la distinction est publique, par votant), le nombre de distributions, le drapeau de vote, l'anneau de vote et le poids ; et (4) un **registre des bulletins** — un enregistrement par bulletin : identifiant d'anneau, charge utile signée telle quelle, signature, image de clé. Rien de publié ne relie un bulletin à un identifiant de vote (hors des modes opt-in de §4.8), et les deux faces utilisent des ordres canoniques différents.

5.2 Stockage des bulletins : exact à l'octet et en ajout strict

Les octets stockés du bulletin sont l'unique source de vérité pour la vérification, l'audit et le dépouillement ; les règles de stockage existent pour les garder identiques au bit près de la signature à la publication :

— La charge utile est stockée **telle quelle dans une colonne binaire** (`bytea`), jamais dans un type JSON normalisant : le `jsonb` de PostgreSQL réordonne les clés et réécrit les espaces, ce qui casserait la vérification de signature exacte à l'octet. Des vues dérivées, analysées, peuvent être mises en cache séparément ; les octets restent canoniques.
— Les lignes de bulletin portent des **identifiants UUIDv4 aléatoires et aucun horodatage** — il n'y a délibérément ni identifiant ordonné ni date de création qui divulguerait l'ordre de dépôt.
— Les tables des bulletins et des ancres de chaîne sont **en ajout strict par déclencheur en base** : tout `UPDATE` ou `DELETE` lève une exception. Les deux exceptions codifiées sont le drapeau `is_chain_published` du bulletin, basculé par le constructeur de blocs dans la même transaction qui valide un bloc — c'est aussi la *seule* barrière que lisent la page de registre destinée aux humains et la blockchain, de sorte que les deux surfaces exposent toujours le même ensemble de bulletins — et le statut OpenTimestamps de l'ancre de chaîne (§5.9), à sens unique `pending` → `completed`.
— Les contraintes d'unicité — (élection, image de clé) sur les bulletins et (élection, clé publique) sur les membres d'anneau — sont des filets de sécurité au niveau de la base, qui tiennent même face à des bogues applicatifs.

5.3 Règles anti-chronologie

L'horodatage des votes est traité comme un canal auxiliaire pertinent pour l'anonymat et supprimé de bout en bout :

— Les bulletins ne sont publiés **que par lots d'au moins 600 enregistrements** (le

plancher de publication par lots — un bureau de vote français moyen), selon un calendrier randomisé, triés par image de clé au sein de chaque bloc ; le reliquat n'est vidé qu'à la clôture.

— Le registre d'émargement est publié **une seule fois**, comme un instantané unique dans le bloc final, de sorte qu'aucune ligne d'émargement ne partage une fenêtre de publication avec un bulletin.

— Aucun enregistrement publié ne porte d'horodatage ; le tri au sein de chaque bloc est déterministe et dérivé du contenu (ordre des octets), jamais de l'ordre d'arrivée.

— La chaîne révèle tout de même un ordre partiel grossier (un bulletin du bloc k a été déposé avant le scellement du bloc $k{+}1$) à la granularité ≥ 600, et la première apparition d'un anneau d'urgence est un signal grossier de « postériorité » pour ses membres. Ces deux résidus sont documentés et acceptés.

5.4 La chaîne : enveloppe et chaînage

Le registre est distribué sous la forme d'une **suite de fichiers JSON en ajout strict, chaînés par empreinte et signés par la plateforme** (« de type blockchain » signifie exactement cela : ajout strict plus chaînage par empreinte — *ni* décentralisée, *ni* ancrée à l'extérieur). Chaque bloc, quel que soit son type, s'ouvre sur la même enveloppe :

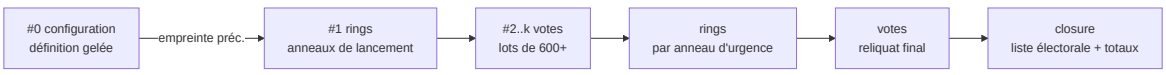
```
{
  "protocol_version": 1,
  "type": "configuration | rings | votes | closure",
  "election_id": "0c2e7e2a-5df3-4c0e-9d2b-3f6a8f0c9e21",
  "block_number": 0,
  "previous_hash": "null pour le bloc 0, sinon 128 caractères hexa (SHA3-512)"
}
```

— Le `previous_hash` du bloc k est l'**empreinte SHA3-512 des octets stockés du bloc** $k{-}1$ (hexadécimal minuscule) ; il n'est `null` que pour le bloc de genèse.

— `block_number` est une séquence consécutive par élection démarrant à 0.

— Les blocs ne sont jamais qu'ajoutés en tête de chaîne ; rien n'est inséré avant un bloc existant et aucune empreinte n'est jamais réécrite. Un anneau d'urgence créé en cours d'élection est simplement un bloc `rings` ajouté entre deux blocs `votes`.

— L'empreinte du bloc lui-même et sa signature ne sont **pas** dans le JSON (un document ne peut pas contenir sa propre empreinte) ; elles vivent dans l'ancre de stockage et dans un fichier compagnon détaché (§5.7, §5.8).



5.5 Types de blocs

Quatre types de blocs existent, dans un cycle de vie fixe ; les formes faisant autorité sont des JSON Schemas (Draft 2020-12), versionnés par version de protocole et partagés par les côtés écriture et lecture. Ils sont publiés, avec la référence du format champ par champ et la procédure de vérification, sur la page *Protocole blockchain et formats* de la plateforme (https://condorcet.vote/blockchain-protocol). Le **résultat de l'élection n'est jamais un bloc** : c'est un recalcul déterministe à partir des blocs publiés.

`configuration` — bloc 0 (genèse). La définition gelée de l'élection : titre, nom de l'organisateur, type d'élection, méthode de vote officielle, sièges, mode d'accès, les trois options de vie privée de §4.8, `max_ring_size`, `initial_electoral_roll_size` (la taille de la liste à l'ouverture — des votants peuvent encore être ajoutés pendant le vote, §4.1, et la liste du bloc de clôture donne alors la taille finale, de sorte que le *nombre* d'ajouts est public), les *paramètres* de bulletin (dont les contraintes de dépouillement sont dérivées de façon déterministe — les contraintes elles-mêmes ne sont jamais stockées), et les candidats triés par étiquette.

Le champ `method` désigne la méthode dont le classement fait le résultat officiel. Il est pris dans une liste fermée — `Schulze Winning`, `Schulze Margin`, `Ranked Pairs Winning`, `Ranked Pairs Margin`, `Copeland`, `Dodgson Quick`, `FPTP` — chaque valeur étant le nom exact que résout la bibliothèque de dépouillement, et il est lié au type d'élection (`FPTP` pour une élection à choix unique, une méthode de Condorcet sinon). Faisant partie du bloc 0, la méthode est engagée avant le premier vote : elle ne peut pas être choisie une fois les bulletins connus.

```
{
  "protocol_version": 1,
  "type": "configuration",
  "election_id": "0c2e7e2a-...",
  "block_number": 0,
  "previous_hash": null,
  "title": "Best lake of the Alps",
  "organizer_name": "Club Alpin",
  "election_type": "preferential",
  "method": "Schulze Winning",
  "seats": 1,
  "voter_access_mode": "closed_nominative_list",
  "enrolment_mandatory": false,
  "non_anonymous_voting": false,
  "nominative_participation": false,
  "weighted_voting": false,
  "max_ring_size": 600,
  "initial_electoral_roll_size": 55,
  "ballot_rules": {
    "blank_allowed": true,
    "full_ranking_required": false,
    "partial_ranking_allowed": true,
    "ties_allowed": false
  },
  "candidates": [
    { "label": "Annecy" },
    { "label": "Bourget" },
    { "label": "Leman" }
  ]
}
```

Le schéma lui-même impose la dépendance à sens unique `non_anonymous_voting = true` ⇒ `nominative_participation = true` — une configuration qui la viole n'est jamais scellée. Les métadonnées de présentation purement cosmétiques (logos, photos, message de remerciement) ne sont *pas* dans le bloc, pas hachées, et ne font pas partie de la définition vérifiée.

`rings` — bloc 1, puis un par anneau d'urgence. La liste complète des clés publiques de chaque anneau, clés en forme préfixée, triées par ordre croissant ; anneaux triés par identifiant d'anneau. Qu'un bloc `rings` soit un bloc de lancement ou d'urgence se déduit de sa position — aucun drapeau supplémentaire :

```
{
  "protocol_version": 1,
  "type": "rings",
  "election_id": "0c2e7e2a-...",
  "block_number": 1,
  "previous_hash": "3fa9...e2c1",
  "rings": [
    { "ring_id": "a1b2...-uuid", "public_keys": ["pk_3f..._9a1b2c3d", "pk_7c..."] }
  ]
}
```

`votes` — blocs de bulletins par lots. Bulletins triés par image de clé (sans chronologie) ; chaque entrée porte l'identifiant d'anneau (pour que le vérificateur sache contre quel jeu de clés vérifier), l'image de clé et la signature en forme préfixée, et la **charge utile signée telle quelle, comme chaîne JSON opaque** — jamais re-sérialisée, de sorte que le vérificateur retrouve les octets signés exacts en lisant la chaîne :

```
{
  "protocol_version": 1,
  "type": "votes",
  "election_id": "0c2e7e2a-...",
  "block_number": 7,
  "previous_hash": "bc41...11d9",
  "ballots": [
    {
      "ring_id": "a1b2...-uuid",
      "key_image": "ki_9d..._4e5f6a7b",
      "signature": "blsag_4e...",
      "payload": "{\"v\":1,\"election\":\"0c2e7e2a-...\",\"ballot\":\"A>B>C\",\"weight\":1}
    }
  ]
}
```

`closure` — le bloc final et terminateur de chaîne. Aucun bloc ne peut lui succéder : posséder le bloc de clôture signé *est* la preuve qu'une chaîne est complète. Il porte les totaux de tête de chaîne et le **registre d'émargement** — la liste à la clôture, votants du lancement et ajouts en cours de vote confondus, sans marqueur — publié ici une seule fois, trié par identifiant de vote :

```
{
  "protocol_version": 1,
  "type": "closure",
  "election_id": "0c2e7e2a-...",
  "block_number": 42,
  "previous_hash": "77aa...03bc",
  "totals": {
```

```
    "block_count": 43,
    "ballot_count": 1300,
    "ring_count": 5,
    "voted_count": 1300,
    "voted_weight": 1300
  },
  "electoral_roll": [
    {
      "voter_id": "d4e5...-uuid",
      "key_provisioning_mode": "server_generated",
      "ring_id": "a1b2...-uuid",
      "distribution_count": 2,
      "has_voted": true,
      "weight": 1
    }
  ]
}
```


Le registre d'émargement est le **dénominateur des bornes anti-bourrage** (§5.11) et le substrat de la détection de fraude par votant — sans lui, « pas de bourrage » ne serait pas vérifiable. Le `key_provisioning_mode` de chaque ligne distingue les clés générées par le serveur, importées par l'organisateur et enrôlées par le votant, de sorte que la part des votants dont la plateforme a un jour détenu la clé est elle-même un fait public, par élection. Quand la divulgation nominative (§4.8) est active, les lignes des votants ayant voté portent en plus `voter_name` et `email_digest` ; jamais celles des abstentionnistes. Au scellement, le constructeur contre-vérifie chaque total qu'il peut prouver depuis le bloc lui-même (`block_count`, `voted_count`, `voted_weight`) et refuse un bloc intérieurement incohérent.

Une élection **annulée** n'émet plus aucun bloc et aucun bloc de pierre tombale : la chaîne s'arrête simplement sans bloc de clôture, ce qui est en soi la preuve cryptographique de l'annulation pour quiconque détient le préfixe.

5.6 Sérialisation : signer le fichier, pas un modèle canonique

La règle fondatrice du format est :

> **octets stockés = fichier servi = octets hachés = octets signés.**

Chaque bloc est sérialisé exactement **une fois**, à sa création ; ces octets sont écrits sur le stockage, servis au public tels quels (passe-plat d'octets bruts, jamais à travers une couche de réponse JSON qui ré-encoderait), hachés pour le chaînage, et signés. Conséquences et règles :

- **Aucun JSON canonique (RFC 8785) n'est nécessaire.** La canonicalisation n'importe que lorsqu'un vérificateur doit *régénérer* les octets depuis un modèle logique et tomber d'accord avec le signataire. Ici, le vérificateur hache et vérifie le fichier même qu'il télécharge, donc le formatage est sans effet sur l'intégrité — le format choisit délibérément un JSON UTF-8 indenté, lisible, chaque élément de l'unique grande collection (`candidates`, `rings`, `ballots`, `electoral_roll`) étant émis sur une ligne compacte.
- **Les blocs sont produits en flux, jamais matérialisés.** Les gros blocs (un rôle de clôture peut atteindre des centaines de mégaoctets) sont écrits itérativement dans un fichier temporaire pendant que le SHA3-512 est calculé au fil de l'eau, et validés élément

par élément contre le schéma d'item — mémoire constante côté écriture comme côté lecture (le vérificateur de lecture est lui aussi un parseur en flux).
— **La charge utile du bulletin reste une chaîne opaque** dans les blocs `votes` et n'est jamais re-sérialisée (§5.5).
— **Les clés JSON dupliquées sont rejetées d'emblée**, des deux côtés. Les parseurs divergent sur les clés dupliquées (premier-gagne contre dernier-gagne), si bien qu'un bloc à clés dupliquées pourrait présenter deux ensembles de bulletins différents à deux vérificateurs tout en étant haché et vérifié à l'identique pour les deux — un vecteur de falsification fermé par une passe de lexeur en flux avant que le moindre contenu ne soit accepté.
— **Tous les nombres sont des entiers non négatifs** ; aucun flottant n'apparaît dans aucun bloc.
— **L'ordre du contenu est déterministe et dérivé du contenu** : candidats par étiquette, clés publiques croissantes, bulletins par image de clé, lignes du rôle par identifiant de vote — toujours en ordre d'octets (`memcmp`), jamais une collation linguistique ; le constructeur refuse un contenu non ordonné au scellement.

5.7 Scellement et ancrage : comment un bloc est produit

La production des blocs est asynchrone (jobs en file d'attente, sérialisés par élection) et suit un pipeline fixe dont la ligne de base de données validée est l'arbitre :

1. **Construire** le bloc logique depuis les données courantes ; le **sérialiser** une fois, en flux, dans un fichier temporaire.
2. **Valider** pendant le scellement : conformité au schéma de l'enveloppe et de chaque item produit en flux (un bloc malformé n'est jamais haché ni signé), plus la cohérence de chaîne — `previous_hash` égal à l'empreinte de contenu stockée de la tête courante et `block_number` égal à tête + 1.
3. **Hacher et signer** : `content_hash = SHA3-512(octets)` calculé au fil de l'eau ; une signature OpenPGP détachée est prise sur exactement les mêmes octets (§5.8).
4. **Écrire** le fichier de bloc et son compagnon `.sig` sur le stockage objet, indexés par (élection, numéro de bloc). Le contenu est déterministe, donc une écriture rejouée est idempotente.
5. **Valider l'ancre** : une transaction de base de données insère la ligne d'ancre en ajout strict (empreinte de contenu + signature + séquence) et bascule les drapeaux `is_chain_published` des bulletins du lot. Une contrainte d'unicité sur (élection, numéro de bloc) rend toute numérotation fourchue impossible — un écrivain concurrent perdant échoue et réessaie contre la nouvelle tête.

La validité est *indépendante de l'emplacement* : un bloc est valide si et seulement si ses octets se hachent vers l'empreinte de contenu validée et que la signature se vérifie. Une ligne validée dont l'objet manque est détectable et re-dérivable ; un objet orphelin sans ligne est inerte. Un vidage de clôture qui échoue est rejoué par un balayage de rattrapage permanent jusqu'à ce que le bloc de clôture existe — un délai, jamais une chaîne amputée.

Le public consomme des fichiers bruts — aucune API requise : un manifeste par élection listant chaque bloc (numéro, type, empreinte, horodatage de signature, URL des artefacts et statut OpenTimestamps), les artefacts `{election}_{n}.json`, `{election}_{n}.json.sig` et `{election}_{n}.json.ots` — chacun nommé avec son identifiant d'élection — servis depuis

le stockage objet derrière un CDN, et les certificats de signature de la plateforme à un point d'accès TLS bien connu. Tout est cachable et mirrorable par conception.

5.8 Signature de plateforme et gestion des clés

Chaque bloc est signé avec une **signature détachée OpenPGP v6 (RFC 9580), Ed25519**, produite par un build épinglé et embarqué de la CLI Sequoia `sq`. Un tiers vérifie un bloc avec une seule commande standard :

```
$ sq verify --signer-file signing-keys.pgp \
    --signature-file 3.json.sig 3.json
```

Points de conception, énoncés factuellement :

- La signature couvre les octets du fichier eux-mêmes. L'heure de création de la signature est un sous-paquet haché ; un sous-paquet de notation (`domain@condorcet.vote = chain-block/v1`) sépare par domaine les signatures de blocs de chaîne de toute autre classe d'artefacts que la plateforme pourrait signer plus tard. Un vérificateur standard qui ignore la notation vérifie tout de même le bloc.
- Les clés de signature vivent dans un jeu de clés ordonné (dernière entrée active, entrées antérieures retirées mais vérifiables). La rotation appose sur la clé sortante une auto-signature d'expiration à l'instant de la rotation, de sorte que tout vérificateur conforme à la RFC 9580 rejette ce qu'une clé retirée signerait plus tard, tandis que tout ce qu'elle a signé dans sa fenêtre continue de se vérifier. Le jeu complet de certificats est publié à un point d'accès TLS bien connu.
- C'est de l'hygiène de confinement, pas une immunité à la compromission : une clé *fuitée* peut antidater des signatures dans sa fenêtre historique — une fenêtre que l'ancrage OpenTimestamps par bloc borne (§5.9). La révocation véritable reste un travail futur documenté.
- Les signatures v6 exigent un vérificateur compatible RFC 9580 (Sequoia, rpgp, gopenpgp, OpenPGP.js récent) ; GnuPG ne peut pas les vérifier aujourd'hui. Les instructions de vérification publiées nomment Sequoia.

5.9 Ancrage temporel externe : OpenTimestamps

Chaque bloc est de plus ancré dans **Bitcoin via OpenTimestamps**. Au scellement, juste après la validation de la ligne d'ancrage en base, la plateforme soumet le **SHA-256 brut** du fichier de bloc aux serveurs calendar publics — délibérément sans le nonce de confidentialité habituel d'OTS, puisque les octets sont publics par conception — de sorte que l'engagement auprès du calendar égale `sha256({n}.json)` et que tout outillage OpenTimestamps standard vérifie la preuve contre le bloc téléchargé, sans aucune confiance dans la plateforme :

```
$ ots verify -f 3.json 3.json.ots
```

La preuve vit comme un troisième fichier compagnon, `{election}_{n}.json.ots`, téléchargeable aux côtés du bloc et de sa signature quel que soit son état : une preuve *pending* porte déjà les engagements des calendars ; une fois Bitcoin confirmé — typiquement en quelques heures — une réconciliation horaire collecte l'attestation, remplace le fichier stocké par la preuve enrichie et fait passer le statut OTS publié du bloc de `pending` à `completed` (le manifeste expose l'URL de la preuve et ce statut). La soumission est strictement au mieux (« best-effort ») : une panne des calendars ne retarde ni ne fait jamais échouer la publication d'un bloc, et le même balayage horaire re-soumet toute preuve perdue de cette façon.

Ce que l'ancrage ajoute : l'heure de création de la signature est une affirmation de la plateforme elle-même, mais une attestation Bitcoin ne l'est pas — elle prouve que les octets exacts du bloc existaient **au plus tard au bloc Bitcoin attesté**, indépendamment des clés de la plateforme. Cela borne la fenêtre d'antidatage d'une clé de signature fuitée (§5.8) et renforce le modèle de mise en miroir (§5.10) d'une preuve d'existence par tiers. La plateforme rapporte ce que la preuve affirme et n'effectue elle-même aucune vérification sur la chaîne Bitcoin ; les vérificateurs le font avec l'outillage standard, comme ci-dessus.

5.10 Ce que « immuable » veut dire ici — et ce qu'il ne veut pas dire

La signature de plateforme prouve l'**authenticité** (la plateforme a produit un bloc) ; à elle seule, elle ne prouve pas l'**immuabilité**, parce que la plateforme détient la clé de signature et pourrait re-signer une histoire réécrite. L'immuabilité est obtenue *en pratique* par la **mise en miroir indépendante** : chaque bloc est un fichier brut, cachable et mirrorable, et tout observateur qui a conservé une tête de chaîne antérieure peut opposer ses copies signées à une réécriture, rendant la falsification manifeste. En l'absence de tout témoin, une plateforme malveillante pourrait réémettre sans détection une histoire alternative validement signée. C'est le modèle de tiers de confiance rendu précis — digne de confiance au moment du vote, auditable ensuite *dans la mesure où le public a effectivement conservé des copies* — et il est énoncé ici plutôt qu'édulcoré. La version 1 du protocole ajoute toutefois un ancrage externe : chaque bloc porte une preuve d'existence OpenTimestamps dans Bitcoin (§5.9). L'ancrage prouve *quand* des octets existaient — une histoire réécrite ne peut pas obtenir d'attestations antidatées — mais il ne prouve pas à lui seul qu'une histoire donnée est la seule publiée ; la mise en miroir indépendante reste la défense contre la réécriture.

5.11 Ce que quiconque peut vérifier

À partir des seuls artefacts publiés, sans aucune confiance dans la plateforme :

1. **Intégrité de la chaîne** — la signature de chaque bloc se vérifie contre le jeu de certificats publié ; chaque `previous_hash` égale le SHA3-512 du fichier précédent (`openssl dgst -sha3-512 block-k.json`) ; la numérotation est consécutive depuis 0 ; chaque bloc porte le même UUID d'élection ; exactement un bloc de clôture termine la chaîne ; chaque bloc se valide contre son JSON Schema et ne contient aucune clé dupliquée. La preuve OpenTimestamps de chaque bloc se vérifie de plus contre son fichier — existence au plus tard au bloc Bitcoin attesté (§5.9).
2. **Authenticité des bulletins** — `verify_vote` passe pour chaque bulletin publié contre les clés publiées de son anneau.
3. **Aucune réutilisation de la même clé** — toutes les images de clé sont uniques au sein de l'élection.
4. **Aucun bourrage par anneau** — pour chaque anneau, les bulletins publiés égalent le nombre de votants dont la ligne d'émargement désigne cet anneau.
5. **La borne globale contraignante** — total des bulletins $\leq$ total des lignes d'émargement ayant voté ; dans les élections pondérées, le multi-ensemble des poids signés des bulletins égale le multi-ensemble des poids des lignes ayant voté. Un bourrage à l'intérieur de ces bornes force des lignes d'émargement fabriquées, chacune détectable par le votant concerné (§6.2).
6. **Le dépouillement** — re-sérialiser les charges utiles publiées en un document CEF, dériver les contraintes de vote des paramètres de bulletin publiés selon la règle documentée,

et recalculer avec la version de moteur référencée (ou toute implémentation conforme CEF) ; la classification blanc/nul et le classement final se reproduisent exactement. L'ensemble des élus correspond aux *sièges* premières positions de ce classement.

Chaque votant peut en outre vérifier sa propre ligne (aucune activité fabriquée) et localiser son propre bulletin via son reçu (§4.7) — y compris avant la publication par lots, grâce à une recherche exacte à débit limité, indexée par l'image de clé, valeur à haute entropie que seul un détenteur de reçu peut présenter.

5.12 Reproductibilité du dépouillement

Le dépouillement est effectué par le moteur open-source CondorcetPHP via une API typée, avec la méthode engagée dans le bloc de configuration (§5.5) ; le résultat publié enregistre la version du contrat moteur et la version de la bibliothèque. Pour une élection Condorcet, la plateforme calcule aussi, dans la même passe, les autres méthodes de sa liste et affiche leurs classements à titre indicatif ; seul le classement de la méthode engagée est officiel. Le document CEF proposé au téléchargement est une commodité, jamais une entrée — il est reproductible depuis le registre. La classification des bulletins (*exprimé*, *blanc*, *nul* — ce dernier scindé en nul-par-format et nul-par-contrainte) est une règle déterministe, publiée, sur les paramètres de bulletin publiés, de sorte que des re-dépouillements indépendants reproduisent les décomptes exacts. La réponse du moteur n'est pas elle-même signée : la garantie d'intégrité n'est pas le moteur mais la reproductibilité du calcul à partir d'entrées signées et publiées — une divergence entre implémentations de dépouillement serait un bogue logiciel, pas un échec de vérifiabilité, parce que la liste fixe des votes signés est la garantie.

6 Modèle de menace

6.1 Menaces couvertes

Menace	Défense
Relier un bulletin à un votant (public, organisateur, export logique ultérieur de la base)	Signature en anneau ; aucune correspondance votant-clé ou votant-bulletin stockée ; stockage et publication sans chronologie
Double vote avec la même clé (y compris rejeu)	Unicité des images de clé, contrainte en base, publiquement vérifiable
Double vote avec des clés différentes	Registre d'émargement : bascule gardée de `has_voted` plus plafond de distribution, tous deux transactionnels
Bourrage d'urne par la plateforme	Bornes bulletins/émargement par anneau et globale (§5.11) ; le bourrage force des lignes d'émargement fabriquées, détectables par les votants concernés
Assèchement d'anneau / acculer un votant dans un anneau clairsemé	Plafond de distribution, provisionnement 1,5x, affectation pondérée par déficit, anneaux d'urgence précoces

Menace	Défense
Traçabilité chronologique des bulletins	UUID aléatoires, aucun horodatage, publication par lots en ordre d'image de clé
Falsification de l'historique publié	Chaîne d'empreintes + signatures de plateforme + mise en miroir indépendante (falsification manifeste pour tout témoin)
Contenu de bloc falsifié ou ambigu	Validation par schéma + rejet des clés dupliquées des deux côtés ; ordre du contenu déterministe
Corrélation inter-élections d'une clé réutilisée	Images de clé à portée d'élection (DDH)

6.2 Limites acceptées — documentées, non dissimulées

— **Une plateforme activement malveillante au moment du vote.** La plateforme connaît transitoirement la correspondance votant-clé à la distribution (mode généré par le serveur) et votant-bulletin à la soumission (tous modes). En mode généré par le serveur, une plateforme malveillante pourrait précalculer l'image de clé de chaque clé qu'elle a frappée et ainsi désanonymiser chaque bulletin du registre. Les clés importées et enrôlées suppriment à la fois ce pouvoir de précalcul et toute possibilité pour la plateforme de signer à la place du votant ; les élections BYO-only suppriment entièrement les secrets détenus par le serveur. Cette confiance bornée *est* le pilier central du modèle, et la raison d'être des modes BYO.

— **Coercition et achat de votes.** Les reçus sont prouvables, et indépendamment des reçus, les classements publiés en clair permettent les attaques par motif (attaques « italiennes ») : un coerciteur impose un ordre reconnaissable de candidats peu saillants et cherche ce bulletin exact dans le registre. Cela concerne chaque votant quel que soit son mode de clé, ne nécessite aucun lien votant-bulletin, et c'est le prix de la publication de bulletins vérifiables en clair. Le protocole vise donc les élections privées et associatives, pas les scrutins publics sensibles à la coercition.

— **Participation fabriquée pour des absents.** La plateforme pourrait marquer un votant absent comme ayant voté et bourrer un bulletin à l'intérieur des bornes publiées ; seul le votant concerné peut le détecter, en vérifiant sa propre ligne du registre. La divulgation nominative opt-in contre partiellement ce risque en permettant à un auditeur détenant la vraie liste d'e-mails de recalculer chaque empreinte. La même borne couvre des votants fictifs ajoutés pendant le vote (§4.1) : c'est la confiance déjà placée dans celui qui compose la liste, décalée dans le temps — le bloc de genèse publie la taille de la liste à l'ouverture, donc le nombre d'ajouts est public, jamais leur identité.

— **Dégénérescence d'anneau clairsemé.** Un anneau dont les *participants* se réduisent à un seul expose publiquement l'identifiant de vote de ce bulletin (un pseudonyme, pas un nom — le résoudre vers une personne exige une connaissance privée de la plateforme ou de l'organisateur, sauf si la divulgation nominative est active). L'atténuation est structurelle (formule d'affectation, anneaux d'urgence précoces, partition équilibrée en BYO) mais le résidu — un dernier anneau sous le plancher à la clôture — est inhérent à toute élection vérifiable dotée d'un registre d'émargement public et il est accepté ; ces bulletins sont tout de même comptés. Avec les ajouts en cours de vote, les membres d'un anneau d'urgence sont surtout des votants ajoutés : y avoir voté se lit comme « arrivé

après l'ouverture » — une inférence au niveau du pseudonyme, jamais du bulletin.

— **Canaux auxiliaires du stockage physique.** PostgreSQL estampille les tuples avec l'identifiant de la transaction écrivante (`xmin`), ce qui crée momentanément une jointure physique déterministe entre un bulletin et sa bascule d'émargement (une seule transaction), invisible dans les exports logiques mais lisible avec un accès physique brut. Un nettoyage bi-quotidien réécrit les tuples d'émargement estampillés sous une unique transaction muette et exécute des vacuums déterministes pour que les versions mortes portant les estampilles d'origine soient supprimées ; une estampille fraîche survit au plus une cadence de nettoyage. Les WAL archivés et les journaux d'accès web restent une préoccupation d'infrastructure et de rétention, cohérente avec le modèle de confiance au moment du vote.

— **Compromission côté client pendant la fenêtre de signature.** La page de vote est autonome, sans framework, et servie sous la CSP premier-parti la plus stricte de la plateforme ; un XSS pendant la fenêtre de signature reste le risque résiduel côté client que cette politique minimise sans pouvoir l'éliminer.

— **Données de contact des votants au repos.** La plateforme détient toujours les noms et e-mails des votants en clair (la délivrance des VoteLink est exclusivement par e-mail, et l'organisateur a besoin d'une liste consultable), donc une compromission de la base les expose directement — un élargissement accepté et documenté du lien votant-e-mail côté plateforme. Les clés secrètes de vote, en revanche, sont toujours chiffrées au repos et supprimées à la distribution.

7 Garde-fous d'implémentation

Au-delà de la cryptographie, un petit nombre de principes d'implémentation portent directement des garanties du protocole. Ils sont énoncés ici au niveau des principes ; l'exploitation de la plateforme au sens large est hors du périmètre de ce document.

7.1 Protéger la fenêtre de signature

La page de vote est une page autonome, sans framework, sous la Content-Security-Policy la plus stricte de la plateforme : premier-parti uniquement, aucun script tiers, aucune analytique ni traçage où que ce soit, exécution WASM activée, envois de formulaires désactivés au profit d'une soumission fetch uniquement. Les assets compilés portent des empreintes d'intégrité de sous-ressources (SHA-384), de sorte qu'un nœud CDN compromis ne peut pas altérer silencieusement le chemin de code de signature. Le flux de vote vit sur son propre sous-domaine (`voting.condorcet.vote`) avec des cookies limités à l'hôte, isolant son état de session du reste du site — et **aucun CDN intermédiaire ne s'interpose entre le navigateur du votant et la plateforme pour les données de vote** : les anneaux, la distribution de clés et la soumission des bulletins voyagent directement vers l'origine sur TLS (seuls les assets statiques compilés sont mis en cache en périphérie).

7.2 Minimisation des données et hygiène de chronologie

La plateforme ne stocke rien dont le protocole n'a pas besoin et ne publie rien qui date ou identifie un bulletin : identifiants UUIDv4 aléatoires sur la surface publique, aucun horodatage de bulletin, octets de bulletin tels quels comme unique source de vérité, secrets chiffrés au repos et activement supprimés à leur point d'usage (distribution, vote, clôture), secrets transitoires en mémoire nettoyés par mise à zéro explicite côté PHP (objets-valeurs adossés à `sodium_memzero`)

comme côté Rust/WASM (tampons mis à zéro), et le nettoyage de chronologie physique de §6.2 tournant comme un invariant planifié en arrière-plan.

7.3 La base de données comme arbitre unique

Chaque invariant exposé aux utilisateurs — plafonds de distribution, bascule de `has_voted`, réclamations de clés, numérotation des blocs — est imposé par des écritures conditionnelles gardées et des contraintes d'unicité dans PostgreSQL, jamais par de l'état in-process : correct par construction en exécution multi-worker et multi-instance, sans ordre de verrous et sans interblocage. La vérification de signature (pur CPU sur des données publiques immuables) s'exécute *avant* la transaction d'écriture, de sorte que la cryptographie ne tient jamais un verrou. Les bulletins et les ancres de chaîne sont en ajout strict par déclencheur ; la production des blocs est sérialisée par élection avec une contrainte d'unicité comme filet de sécurité absolu contre les fourches.

7.4 Isolation de la publication

Les artefacts de chaîne vivent sur leur propre bucket de stockage objet avec leurs propres identifiants d'accès, séparés de tout autre contenu stocké et servis par leur propre nom d'hôte CDN — un défaut dans n'importe quel chemin de téléversement ne peut jamais écrire dans le bucket de la chaîne. Le bucket active en outre le mécanisme de **verrouillage d'objets (écriture unique) du fournisseur de cloud** : une fois écrit, un fichier de bloc ne peut être ni écrasé ni supprimé, même avec les identifiants de stockage de la plateforme — une application au niveau du stockage de la règle d'ajout strict, complétant les déclencheurs en base et la chaîne d'empreintes. Les blocs sont servis comme des octets bruts ; aucune couche de framework ne peut les reformater.

7.5 Discipline d'ingénierie

La suite de tests exerce la **véritable cryptographie WASM** — jamais de simulacres — sur les chemins serveur (Extism) comme navigateur (wasm-bindgen), y compris des parcours de vote complets pilotés par navigateur ; le crate lui-même fige des vecteurs de protocole (tout changement silencieux d'une chaîne de domaine, d'un format de liaison ou d'une disposition d'octets fait échouer le build), exécute des balayages de malléabilité au niveau du bit et des cibles de fuzzing sur tous les parseurs, et audite ses dépendances en CI. Les spécifications sont versionnées dans le dépôt et maintenues en synchronisation avec le code par convention explicite ; ce livre blanc est en aval des deux.

8 Arbitrages de conception : cryptographie avancée contre facilité d'utilisation

Le protocole choisit systématiquement le *mécanisme le plus simple qui garde la garantie vérifiable*, et accepte en échange des coûts documentés. Les décisions ci-dessous sont présentées factuellement — chacune est un échange, pas un avantage gratuit.

8.1 La vérifiabilité par la publication, non par une cryptographie de dépouillement avancée

Les réseaux de mélange, les dépouillements homomorphes et les preuves de dépouillement à divulgation nulle de connaissance ont été considérés puis écartés comme disproportionnés pour

les cas d'usage visés. À la place, la vérifiabilité provient de la publication de bulletins signés en clair et du recalcul déterministe du dépouillement. **Gagné :** quiconque peut auditer avec `sq`, `openssl` et n'importe quel dépouilleur conforme CEF ; aucune cérémonie de confiance ; aucun logiciel de vérification ésotérique. **Coût :** les classements publiés permettent les attaques par motif (§6.2) et excluent la résistance à la coercition ; la plateforme doit être digne de confiance au moment du vote.

8.2 Des reçus plutôt que l'absence de reçus

La vérifiabilité individuelle (chaque votant peut trouver et vérifier son propre bulletin) a été préférée à l'absence de reçu ; les deux sont mutuellement exclusives sans une machinerie substantiellement plus lourde. **Gagné :** chaque votant est un détecteur de fraude potentiel, sans coût d'interaction supplémentaire. **Coût :** les reçus sont prouvables et transférables (§3.4), donc l'achat de votes et la coercition ne peuvent pas être empêchés cryptographiquement — d'où le positionnement explicite sur les élections privées.

8.3 Clés générées par le serveur par défaut, BYO-only comme mode de confiance minimale

Exiger de chaque votant qu'il génère et protège une paire de clés déplacerait le principal mode d'échec sur les participants les moins techniques. Le défaut laisse donc la plateforme frapper et distribuer les clés à la demande — zéro friction, au prix d'une confiance dans la plateforme au moment de la distribution (y compris son pouvoir théorique de précalcul des images de clé, §6.2). Le curseur de confiance est par votant (tout votant peut importer ou enrôler sa propre clé, sans repli côté plateforme) et va jusqu'au **mode BYO-only**, une option d'élection à réglage unique qui supprime entièrement les secrets détenus par le serveur et, avec eux, l'essentiel des arbitrages côté distribution de cette section — au prix du transfert de la garde des clés sur chaque votant (voir le tableau comparatif en §4.2). Le choix entre les deux se fait par élection, pas comme une constante de la plateforme.

8.4 Un vote sans compte, par liens de capacité envoyés par e-mail

Les votants ne créent jamais de compte ni de mot de passe ; leur VoteLink est une capacité au porteur délivrée par e-mail. **Gagné :** la décision d'utilisabilité au plus fort impact de tout le système ; aucun cycle de vie d'identifiants pour les votants. **Coût :** la plateforme détient nécessairement l'e-mail de chaque votant en clair, et la possession du lien est la possession du vote (atténué par le hachage du jeton au repos, le périmètre de session et la limitation de débit ; accepté au sein du modèle).

8.5 Une taille d'anneau plafonnée : un plafond, une taille de lancement par élection

Les anneaux sont plafonnés à 600 clés par élection, et tous les anneaux de lancement d'une élection partagent une taille unique sous ce plafond, dérivée de la liste gelée (§4.3) — plutôt que des anneaux par votant ou un dimensionnement entièrement dynamique. Au plafond, un bulletin coûte environ 19 Ko de signature et environ une demi-seconde de temps de signature comme de vérification ; une petite élection au plancher de 100 clés en coûte le sixième. L'ensemble d'anonymat cryptographique est borné à l'anneau plutôt qu'à l'électorat entier — mais l'ensemble d'anonymat *effectif* est celui des votants participants de l'anneau, que le rembourrage n'agrandit jamais : pour toute élection qui tient dans un seul anneau (jusqu'à 400 votants),

c'est tout l'électorat participant, à 100 clés exactement comme à 600. **Gagné :** l'uniformité (tous les anneaux de lancement d'une élection sont identiques, tous les anneaux d'urgence sont au plafond), la résistance à l'assèchement d'anneau (sous le plafond, la capacité couvre toujours trois distributions par votant provisionné par le serveur), une vérification publique simple et des performances prévisibles sur des appareils modestes. Le plafond de 600 correspond au plancher de publication par lots — l'effectif d'un bureau de vote — et c'est aussi la taille de chaque anneau d'urgence, de sorte qu'une liste qui grossit en cours d'élection est absorbée par quelques grands anneaux plutôt que fragmentée en beaucoup de petits.

8.6 Le cache de redistribution

En toute rigueur, le protocole voudrait que la correspondance votant-clé n'existe jamais hors de l'appareil du votant. En pratique, les votants rechargent des pages et perdent des onglets ; sans re-délivrance, chaque rechargement brûlerait l'une des trois distributions plafonnées. La plateforme conserve donc le secret distribué, chiffré et indexé par votant, pendant au plus 180 minutes et le supprime au vote — un élargissement délibéré, borné par TTL, de la fenêtre de confiance au moment du vote, échangé contre le fait de ne pas priver de vote des votants légitimes. Les élections BYO-only éliminent cet arbitrage purement et simplement : sans distribution, le cache — et la liaison transitoire qu'il représente — n'existe jamais.

8.7 Une plateforme centrale aux preuves mirrorables, pas un registre décentralisé

Exploiter un registre décentralisé ajouterait un poids opérationnel et conceptuel pour les organisateurs que cette plateforme vise. À la place, la chaîne est produite de façon centralisée mais conçue pour une **garde indépendante bon marché** : des fichiers JSON bruts, une seule commande de vérification standard, aucune API nécessaire — plus un ancrage OpenTimestamps par bloc (§5.9) qui ajoute une preuve d'existence par tiers à coût négligeable. La garantie d'immuabilité est exactement aussi forte que la mise en miroir dont elle bénéficie (§5.10) — énoncée comme une propriété, pas présentée commercialement comme équivalente à un consensus.

9 Limites connues et travaux prévus

— **Revue indépendante.** Une revue externe du protocole et de son implémentation est prévue — avec une attention particulière à la construction d'image de clé à portée d'élection, déviation délibérée et étroite de la construction bLSAG publiée. Son rapport sera lié depuis la page sécurité de la plateforme.

— **Ancrage temporel de niveau eIDAS.** L'ancrage OpenTimestamps par bloc est implémenté (§5.9) ; une TSA RFC 3161 pourrait le compléter là où une valeur probante de niveau eIDAS est requise.

— **Révocation véritable des clés.** La V1 repose sur l'expiration à la rotation ; une sémantique de révocation dure (avec certificats de révocation conservés hors bande) est esquissée pour une version ultérieure — l'ancrage OpenTimestamps par bloc (§5.9) fournit l'ancrage externe auquel elle était conditionnée.

— **Politique de rétention et d'archivage.** La rétention à long terme des registres et des métadonnées opérationnelles (alignement RGPD) est volontairement différée et suivie dans les questions ouvertes des spécifications.

10 Références

— [LWW04] J. K. Liu, V. K. Wei, D. S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups.* IACR ePrint 2004/027.

— [ZtM20] koe, K. M. Alonso, S. Noether. *Zero to Monero* (2e éd., 2020) — la construction bLSAG et son analyse de chaînabilité.

— [CP92] D. Chaum, T. P. Pedersen. *Wallet Databases with Observers.* CRYPTO '92.

— [FS86] A. Fiat, A. Shamir. *How to Prove Yourself : Practical Solutions to Identification and Signature Problems.* CRYPTO '86.

— [Ristretto] The Ristretto Group. https://ristretto.group — groupe d'ordre premier sur Curve25519 ; implémenté par `curve25519-dalek`.

— [RFC7693] M-J. Saarinen, J-P. Aumasson. *The BLAKE2 Cryptographic Hash and MAC.* RFC 7693.

— [BLAKE3] J. O'Connor, J-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn. *BLAKE3 : one function, fast everywhere.* 2020.

— [FIPS202] NIST. *SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions.* FIPS PUB 202.

— [RFC9580] P. Wouters et al. *OpenPGP.* RFC 9580 (OpenPGP v6).

— [RFC8032] S. Josefsson, I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA).* RFC 8032.

— [cryptoVote] Le crate Rust `crypto_vote`. https://github.com/CondorcetVote/cryptoVote

— [Condorcet] J. Boudry. *Condorcet* — le moteur CondorcetPHP et le Condorcet Election Format (CEF). https://github.com/julien-boudry/Condorcet

— [Extism] Extism — le cadre de plug-ins WebAssembly utilisé pour la vérification côté serveur. https://extism.org

— [Sequoia] Sequoia-PGP — l'implémentation OpenPGP dont la CLI `sq` signe et vérifie les blocs de la chaîne. https://sequoia-pgp.org

---

Éditeur. Condorcet.Vote est opéré par **Condorcet.Vote SASU**, société par actions simplifiée de droit français — SIREN 000 000 000, RCS [ville] *(identifiants à compléter avant publication).*