Outils cryptographiques pour le vote électronique

Pierrick Gaudry

Caramba – LORIA UMR 7503
CNRS, Université de Lorraine, Inria

Collège de France – 13 novembre 2025

Travail commun avec
- Véronique Cortier et Stéphane Glondu (partie Belenios)
- Véronique Cortier et Quentin Yang (partie dépouillement en aveugle)

Plan

Propriétés souhaitées : le secret

Pourquoi le secret du vote ?

- Parfois pas souhaitable (e.g. vote de représentants élus).
- Relève de la protection de la vie privée.
- Préserver le libre choix du vote.

Difficulté : le résultat révèle de l'information

- Bureau de vote minuscule.
- Cas de l'unanimité.
- "Les nouveaux du village".
- Des électeurs se liguent et sacrifient leur voix pour révéler le vote d'une cible.

Les définitions formelles modernes tentent de capturer ces scénarios.

Propriétés souhaitées : la vérifiabilité

Exactitude du résultat

- Deux pré-requis :
  - liste fixe des électeurs légitimes.
  - chaque électeur a une intention de vote bien définie.
- Le résultat doit correspondre à l'intention des électeurs.
- Subtilité liée aux électeurs malhonnêtes.

Vérifiabilité

- Rendre les participants (électeurs, auditeurs) acteurs de la vérification du résultat pour convaincre.
  - Vérifiabilité **de l'intention :** mon bulletin contient mon intention de vote.
  - Vérifiabilité **individuelle :** je "vois" mon bulletin présent dans l'urne.
  - Vérifiabilité **universelle :** quiconque peut vérifier que le résultat correspond aux bulletins dans l'urne.
  - Vérifiabilité **de la légitimité :** quiconque peut vérifier que les bulletins proviennent d'électeurs légitimes.

Le modèle de confiance

Pour chaque propriété de sécurité :

- Choix d'un ensemble de composants que l'on **suppose honnêtes** ;
- Preuve que la propriété est vraie même si tous les autres composants **collaborent avec l'attaquant**.

La **CNIL** liste 3 niveaux de sécurité, selon l'enjeu de l'élection, avec 3 modèles de confiance associés.

Exemples :

- Les bulletins sont chiffrés ; la clé est répartie sur n autorités. Le secret du vote tient sous réserve qu'au moins une autorité et la machine de l'électeur sont honnêtes.
- Idéalement, la vérifiabilité du résultat tient sous très peu d'hypothèses.

Les systèmes Helios et Belenios

Helios : (Ben Adida)

- Date des années 2000 ; nombreuses idées datent des années 90.
- Plateforme en ligne où chacun peut mener ses propres élections.
- Utilisations en 2009 pour l'élection du président de l'Univ. Catholique de Louvain ; puis depuis 2010 pour les élections de l'IACR.

Helios est l'un des principaux **paradigmes de vote par Internet** sur lequel se greffent de multiples variantes.

Belenios :

- Date de 2015. (V. Cortier, PG, S. Glondu)
- Variante corrigeant certains problèmes d'Helios, notamment pour l'éligibilité.
- Mieux adapté au contexte français (support multi-langue ; possibilité de voter blanc ; écrit en OCaml ;-)).
- Il y a également une plateforme en ligne (annuellement, environ 1500 élections et 200,000 électeurs).

Helios/Belenios : setup

Prérequis : un **tableau d'affichage public** PBB, dont on ne peut pas effacer les données.

Setup (1/3) : la commission électorale publie sur PBB

- la **liste électorale** (ou juste son haché, si celle-ci ne peut pas être publiée) ;
- les identités de divers **tiers de confiance** : autorités de déchiffrement, autorité de codes de vote ;
- la liste des questions et des réponses possibles, avec **les règles** (par ex. cocher entre 1 et 3 noms parmi les 10 proposés ; vote blanc autorisé ?).

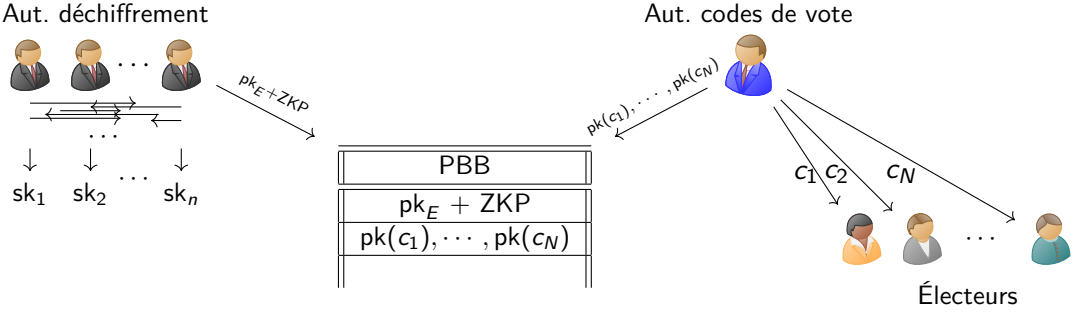
Setup (2/3) : les n autorités de déchiffrement

- effectuent un protocole de **génération distribuée de clé**
- conservent chacune leur part secrète sk_i
- publient sur PBB une clé publique pk_E
- publient des preuves zero-knowledge (ZKP) que tout s'est bien passé.

Helios/Belenios : setup

Setup (3/3) : l'autorité de codes de vote

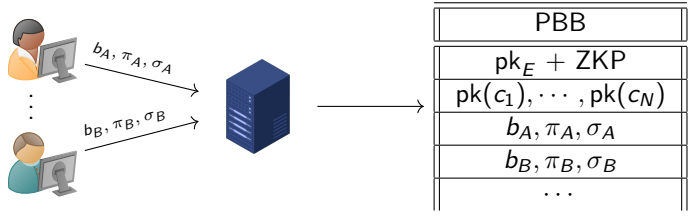
- calcule une **clé de signature** pour chaque électeur
- envoie à chaque électeur son **code de vote** c_i, i.e. la partie secrète de la clé (pour signer)
- publie la **partie publique** $\mathrm{pk}(c_i)$ de chaque clé (pour vérifier les signatures)



Helios/Belenios : phase de vote (cas oui/non)

L'**électeur** A fait son choix $v_A \in \{0,1\}$, et son **ordinateur** (supposé honnête) :

- calcule le **chiffré du vote** $b_A = \mathrm{enc}_{\mathrm{pk}_E}(v_A)$;
- produit une preuve zero-knowledge (**ZKP**) π_A que le bulletin est bien formé ;
- **signe** avec son code de vote ; $\sigma_A = \mathrm{sig}_{c_A}(b_A, \pi_A)$;
- s'authentifie auprès du serveur et lui envoie le tout pour publication sur PBB.



Vérifications par le serveur et des auditeurs :

- vérifient que la signature σ_A est valide, grâce à la clé $\mathrm{pk}(c_A)$ présente sur PBB ;
- vérifient que la ZKP π_A est valide ;
- s'il y a un bulletin déjà publié pour A, marquent le premier comme obsolète.

Helios/Belenios : phase de dépouillement

Cas simple : les électeurs votent 0 (non) ou 1 (oui).

Propriété **semi-homomorphe** du chiffrement pour calculer un **bulletin accumulé** :

$$\begin{aligned} B &= b_1 \times b_2 \times \cdots \times b_N = \mathsf{enc}_{\mathsf{pk}_E}(v_1) \times \mathsf{enc}_{\mathsf{pk}_E}(v_2) \times \cdots \times \mathsf{enc}_{\mathsf{pk}_E}(v_N) \\ &= \mathsf{enc}_{\mathsf{pk}_E}(v_1 + v_2 + \cdots + v_N) \\ &= \mathsf{enc}_{\mathsf{pk}_E}(\text{nombre de oui}) \end{aligned}$$

Remarque : tout le monde peut calculer B à partir des données sur PBB.

Les **autorités de déchiffrement :**

- utilisent chacune leur secret sk_i pour **calculer conjointement le déchiffrement** $\mathsf{dec}(B)$ de B ;
- produisent une **ZKP** de bon déchiffrement π_{dec} ;
- **publient** l'ensemble sur PBB.

Rôle des ZKPs

- ZKP que le setup s'est bien passé :
  - chaque autorité confirme qu'il connait sk_i et pourra dépouiller ;
  - la dernière autorité à agir dans le setup n'a pas choisi son sk_i en fonction de ce que les autres ont publié.
- ZKP de bonne formation du bulletin chiffré :
  - un électeur (partisan du oui) pourrait chiffrer 2 au lieu de 1 pour que son vote compte double ;
  - un électeur (partisan du non) pourrait chiffrer -1 au lieu de 0 pour que son vote compte double ;
  - respect des règles dans le cas d'une question plus complexe.
- ZKP de bon déchiffrement :
  - garantit que $\mathrm{dec}(B)$ est bien le déchiffrement de B ;
  - si toutes les autorités de déchiffrement sont malhonnêtes, elles cassent le secret du vote, mais ne peuvent pas changer le résultat.

Les ZKPs réduisent les hypothèses sur qui est honnête.

Plan

Rappel : le chiffrement ElGamal (exponentiel)

Soit $\mathbb{G}$, groupe cyclique de générateur g, d'ordre premier q (e.g. courbe elliptique).
Clé secrète $\mathsf{sk} \in \mathbb{Z}_q$; **clé publique** associée $\mathsf{pk} = g^{\mathsf{sk}} \in \mathbb{G}$.

On suppose que le **problème du logarithme discret** (retrouver sk connaissant pk) est difficile dans $\mathbb{G}$. *(et DDH aussi...)*

Chiffrement ElGamal exponentiel :
Soit m un petit entier à chiffrer.

$$\mathsf{enc}_{\mathsf{pk}}(m) = (c_1, c_2) = (g^r,\ \mathsf{pk}^r \cdot g^m),$$

où r est un élément aléatoire uniforme dans $\mathbb{Z}_q$.

Déchiffrement :
On calcule $\tilde{m} = c_2/(c_1^{\mathsf{sk}})$, puis on teste toutes les valeurs possibles de m, jusqu'à trouver celle telle que $\tilde{m} = g^m$.

Caractère **semi-homomorphe** :

$$\begin{aligned} \mathsf{enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{enc}_{\mathsf{pk}}(m_2) &= (g^{r_1} g^{r_2},\ \mathsf{pk}^{r_1} g^{m_1} \mathsf{pk}^{r_2} g^{m_2}) = (g^{r_1+r_2},\ \mathsf{pk}^{r_1+r_2} g^{m_1+m_2}) \\ &= \mathsf{enc}_{\mathsf{pk}}(m_1 + m_2). \end{aligned}$$

ZKP de connaissance d'un log discret

Données publiques : h un élément de $\mathbb{G}$, de générateur g.

Prouveur **P** connait $x \in \mathbb{Z}_q$ tel que $h = g^x$; il veut convaincre le vérifieur **V**.
(Utilisation en vote : les autorités prouvent qu'elles connaissent les clés de déchiffrement.)

- **Mise en gage : P** tire $k \in \mathbb{Z}_q$ au hasard, calcule $K = g^k$, et l'envoie à **V**.
- **Défi : V** choisit $e \in \mathbb{Z}_q$ au hasard et l'envoie à **P**.
- **Réponse : P** calcule $s = k - xe \bmod q$ et l'envoie à **V**.
- **Résultat : V** accepte si et seulement si $K = g^s h^e$.

On vérifie que si tout se passe bien,

$$g^s h^e = g^{k-xe}(g^x)^e = g^k g^{-xe} g^{xe} = K,$$

et **V** accepte la preuve.

Schéma d'un protocole Sigma

Un **protocole Sigma** se déroule en 3 phases :

$$\begin{array}{ll} \mathbf{P} \longrightarrow \mathbf{V} & \text{mise en gage} \\ \mathbf{V} \longrightarrow \mathbf{P} & \text{défi} \\ \mathbf{P} \longrightarrow \mathbf{V} & \text{réponse} \end{array}$$

et à la fin **V** accepte ou rejette la preuve.

Les propriétés attendues sont (de manière informelle) :

- **Complétude**. **V** accepte toujours la preuve lorsque **P** connait le secret et effectue le protocole.
- **Zero-knowledge**. Si **V** suit le protocole, il n'apprend rien, sauf que **P** connait le secret.
- **Robustesse**. Si **P** ne connait pas le secret, **V** rejette la preuve avec grande probabilité.

Modélisation de ZK et de robustesse

Propriété de zero-knowledge :
On exhibe un algorithme probabiliste polynomial, appelé **simulateur** qui, à partir des données publiques, produit un triplet de données indistinguable des donnés transitant lors des 3 phases du protocole Sigma.

(Évidemment, le simulateur ne fabrique pas le triplet dans l'ordre ; il a le droit de choisir la mise en gage à la fin.)

Propriété de robustesse :
On exhibe un algorithme probabiliste polynomial, appelé **extracteur** qui, étant données deux exécutions valides distinctes avec la même mise en gage, calcule le secret de **P**.

(Si on ne borne pas le temps d'exécution, l'extracteur pourrait calculer des logarithmes discrets ; c'est triché.)

Remarque : Ce sont des définitions qui se combinent bien avec les **preuves de sécurité** au niveau global.

ZK et robustesse de ZKP de connaissance de log

- $\mathbf{P} \to \mathbf{V}$: $K = g^k$, où $k \in \mathbb{Z}_q$
- $\mathbf{V} \to \mathbf{P}$: $e \in \mathbb{Z}_q$
- $\mathbf{P} \to \mathbf{V}$: $s = k - xe \bmod q$
- $\mathbf{V}$ accepte ssi $K = g^s h^e$.

Simulateur pour zero-knowledge :

- Choisir un défi $e \in \mathbb{Z}_q$ au hasard.
- Choisir une réponse $s \in \mathbb{Z}_q$ au hasard.
- Calculer la mise en gage $K = g^s h^e$.

C'est un **triplet valide**.
Le triplet (K, e, s) suit la **même distribution** que si l'on suit le protocole.

Extracteur pour robustesse :
Soient (K, e, s) et (K, e', s'), deux exécutions valides distinctes. Alors

$$K = g^s h^e = g^{s'} h^{e'}.$$

Donc $h^{e-e'} = g^{s'-s}$, puis $h = g^{(s'-s)/(e-e')}$, et l'on extrait le secret $x = \frac{s'-s}{e-e'} \bmod q$.
(Le dénominateur est non-nul, car les triplets sont distincts.)

La transformation Fiat-Shamir

Dans le vote (et très souvent) : le Vérifieur n'est pas en interaction avec le Prouveur.
On veut une ZKP **non-interactive**.

Solution : (Fiat-Shamir, 1986)

- Remplacer la **temporalité** (mise en gage *avant* défi) par une **notion calculatoire**.
- On prend H une fonction de hachage cryptographique (à sens unique).
- Le **défi** est défini comme le **haché de la mise en gage** (et du contexte de la preuve).

C'est une idée simple, efficace, prouvée sûre si H est modélisée comme un **oracle aléatoire**. *(NB : cette modélisation est, de fait, interactive.)*

$\mathbf{P} \longrightarrow \mathbf{V}$	mise en gage
$\cdots$	défi $= H$(contexte, mise en gage)
$\mathbf{P} \longrightarrow \mathbf{V}$	réponse

ZKP d'exponentiation correcte

Données publiques : h un élément de $\mathbb{G}$, et deux éléments de groupe M et C.

Prouveur **P** connait $x \in \mathbb{Z}_q$ tel que $h = g^x$ et $M = C^x$; il veut convaincre le **V**.

(Utilisation en vote : h est la clé publique de **P***, et il prouve qu'il déchiffre correctement.)*

- **Mise en gage : P** tire $k \in \mathbb{Z}_q$ au hasard, calcule $A = g^k$ et $B = C^k$; puis envoie (A, B) à **V**.
- **Défi : V** choisit $e \in \mathbb{Z}_q$ au hasard et l'envoie à **P**.
- **Réponse : P** calcule $s = k - xe \bmod q$ et l'envoie à **V**.
- **Résultat : V** accepte si et seulement si $A = g^s h^e$ et $B = C^s M^e$.

Exercice : complétude (facile), fabriquer un simulateur pour ZK et un extracteur pour robustesse.

Le cadre général de Maurer

Les deux ZKPs précédentes rentrent dans un **cadre général (Maurer 2009)**.

Soit $\phi : G \to H$, un **morphisme** entre deux groupes finis d'exposant premier q.

Supposons que ϕ est à sens-unique, que $z \in H$ est une donnée publique et que **P** veut prouver à **V** qu'il connait x tel que

$$\phi(x) = z.$$

Le **protocole Sigma** suivant est une ZKP :

- **P** choisit $k \in G$ au hasard, calcule $K = \phi(k)$ et l'envoie à **V**.
- **V** choisit un défi $e \in \mathbb{Z}_q$ au hasard et l'envoie à **P**.
- **P** calcule $s = k \cdot x^e$ et l'envoie à **V**.
- **V** accepte ssi $\phi(s) = K \cdot z^e$.

Complétude : $\phi(s) = \phi(k)\phi(x)^e = Kz^e$, car ϕ est un morphisme.
Robustesse : extracteur donné par : $K = \phi(s)/z^e = \phi(s')/z^{e'}$. Donc $\phi(s/s') = z^{e-e'}$, puis $z = \phi((s/s')^{(e-e')})$, ce qui donne x.

Exemples d'application du cadre de Maurer

Connaissance de log discret :

$$\begin{array}{rccc} \phi: & \mathbb{Z}_q & \mapsto & \mathbb{G} \\ & x & \to & g^x \end{array}$$

Exponentiation correcte :

$$\begin{array}{rccc} \phi: & \mathbb{Z}_q & \mapsto & \mathbb{G} \times \mathbb{G} \\ & x & \to & (g^x, C^x) \end{array}$$

Extrait de Eprint 2025/1625 (Cortier et al.)
Une ZKP de bon mélange de chiffrés ElGamal (adapté de Terelius-Wikström) :

Let ϕ be the group morphism from $\mathbb{Z}_q^5 \times \mathbb{Z}_q^N \times \mathbb{Z}_q^N$ to $\mathbb{G}^{N+6}$, that, taking as input

$$(x_1, x_2, x_3, x_4, x_5), (\xi_1, \ldots, \xi_N), (\zeta_1, \ldots, \zeta_N)$$

gives

$$g^{x_1}, g^{x_2}, g^{x_3}, g^{x_4} \cdot \prod h_i^{\xi_i}, g^{x_5} \cdot \left(\prod (a_i^{u_i})^{x_1}\right) \cdot \left(\prod a_i'^{\xi_i}\right)^{-1},$$

$$\mathsf{pk}^{x_5} \cdot \left(\prod (b_i^{u_i})^{x_1}\right) \cdot \left(\prod b_i'^{\xi_i}\right)^{-1}, g^{\zeta_1} \hat{c}_0^{\xi_1}, \ldots, g^{\zeta_N} \hat{c}_{N-1}^{\xi_N}.$$

It is (boring but) easy to check that

$$\phi((k, \bar{r}, \hat{r}, \tilde{r}, r'), \vec{u}', \vec{\hat{r}}) = z =$$
$$= \left(K, (\prod c_i)/(\prod h_i), \hat{c}_N / h^{\prod u_i}, \prod c_i'^{u_i}, 1, 1, \hat{c}_1, \ldots, \hat{c}_N\right),$$

which is publicly computable.

Le OR de deux ZKPs (CDG 94)

Soient deux ZKPs, pour des données secrètes x_1 et x_2 :

	ZKP1 (x_1)	ZKP2 (x_2)
Mise en gage	com1	com2
Défi	chal1	chal2
Réponse	resp1	resp2

ch = chal1 + chal2

P veut montrer qu'il connait x_1 ou x_2 (sans dire lequel).

Cas où **P** connait x_1 :

- **P** utilise le **simulateur** de ZKP2 et produit une trace d'exécution valide (com2, chal2, resp2).
- **P** tire com1 au hasard, et envoie (com1, com2) à **V**
- **V** tire un défi ch au hasard et l'envoie à **P**
- **P** calcule chal1 = ch - chal2, et calcule resp1 correspondant.
  P envoie (chal1, resp1) et (chal2, resp2) à **V**
- **V** accepte si les deux executions sont valides et si chal1+chal2 = ch.

Le OR de deux ZKPs : pourquoi ça marche ?

La vue de **V** est **complètement symétrique** en ZKP1 / ZKP2.
Le côté du simulateur est **masqué**.

Si **P** ne connait ni x_1, ni x_2, il va devoir utiliser les deux simulateurs. La probabilité que ch $=$ chal1 $+$ chal2 est négligeable.

Exemples en vote :

- j'ai chiffré 0 ou 1 (et pas autre chose) ;
- j'ai voté blanc ou bien j'ai coché exactement 3 candidats.
- (utilisé dans des protocoles de résistance à l'achat de vote) : produire une "designated verifier zkp".

Plan

Pourquoi dépouiller en aveugle ?

Question : qu'est-ce que le résultat d'une élection ?

- Le score de chaque candidat · e ?
- Juste le / les gagnant · e · s ?
- ... y compris si le mode de calcul est complexe ?

C'est un **moyen d'améliorer le secret** du vote.

- **Cas de l'unanimité :** publier juste le nom qui a gagné.
  Ou encore, publier le/la gagnant · e avec un intervalle pour la marge. Par ex. *"A a gagné avec entre 50% et 60% des voix.*
- **Lutte contre l'attaque *à l'italienne* :**
  Se produit lorsque les bulletins sont complexes (e.g. on trie les 20 candidats).
  L'électeur peut utiliser les informations les moins signifiantes pour "signer" son bulletin.
  Faire tout ou partie du calcul de dépouillement sans révéler les bulletins.

Objectif : Les autorités de confiance n'ont pas plus d'information que ce qui est publié.

Principe du calcul multi-partite (MPC)

Données du problème :

- Des **données publiques** D (ici, l'urne, qui contient les bulletins chiffrés avec pk_E) ;
- Des **autorités** T_i, qui possède chacune une **information secrète** sk_i (ici, les autorités de déchiffrement, et leur part de la clé secrète de pk_E) ;
- Une **fonction publique** $F(D, \mathrm{sk}_1, \ldots, \mathrm{sk}_n)$. (ici, la fonction qui déchiffre les bulletins, calcule le résultat, et retourne uniquement ce dernier).

Objectif :

- Construire un **protocole** effectué par les T_i, retournant la valeur de $F()$ et des données d'audit (des ZKPs) ;
- *(robustesse)* Le résultat doit être vérifiable par tout le monde (aucun moyen de publier un résultat faux) ;
- *(zero-knowledge)* Si au moins un des T_i est honnête, aucune information autre que la valeur de $F()$ ne doit fuiter.

Briques de base pour le MPC

Rappel des **propriétés du chiffrement** ElGamal exponentiel :

- **Semi-homomorphe** : $\mathrm{enc}(a) \times \mathrm{enc}(b) = \mathrm{enc}(a + b)$;
- **Re-randomisable** : $\mathrm{rerand}(\mathrm{enc}(a)) = \mathrm{enc}(a) \times \mathrm{enc}(0)$ préserve la valeur du clair, mais est indistinguable d'un chiffré aléatoire ;
- **Déchiffrement distribué** ddec : les T_i disposent d'un protocole MPC pour déchiffrer (robuste et zero-knowledge).

Principe :

- Manipuler des chiffrés qui contiennent des **bits** $\in \{0, 1\}$.
- Fabriquer des **portes logiques**, puis combiner.
- Au niveau élémentaire :
  - **Masquer** les inputs ;
  - **Dechiffrer** ;
  - Dé-masquer.
- *(et des ZKPs à chaque étape.)*

La porte AND (simplifiée)

(B. Schoenmakers, P. Tuyls, 2004)

Input : $X = \mathrm{enc}(x)$, avec $x \in \{0,1\}$;
$Y = \mathrm{enc}(y)$, avec $y \in \{0,1\}$
Output : $Z = \mathrm{enc}(z)$, avec $z = xy$ (i.e. x AND y).

- **Initialisation :** $X_0 = X$ et $Y_0 = \mathrm{enc}(-1) \times Y^2$;
  (les chiffrements de $x_0 = x$ et de $y_0 = (2y-1) \in \{-1,1\}$)
- **Pour** i **allant de** 1 **à** n **l'autorité** T_i :
  - choisit au hasard $s_i \in \{-1,1\}$;
  - calcule $X_i = \mathrm{rerand}(X_{i-1}^{s_i})$;
  - calcule $Y_i = \mathrm{rerand}(Y_{i-1}^{s_i})$;
  - produit une ZKP π_i que le calcul est conforme ;
  - publie X_i, Y_i, π_i (et oublie s_i).
- Les autorités **déchiffrent** collectivement Y_n et obtiennent et publient $y_n \in \{-1,1\}$.
- Le **résultat** est $Z = (X \times X_n^{y_n})^{1/2}$.

$$x_0 = x,\ y_0 = 2y-1$$
$$\downarrow\ T_1,\ s_1 = 1$$
$$x_1 = x_0,\ y_1 = y_0$$
$$\downarrow\ T_2,\ s_2 = -1$$
$$x_2 = -x_0,\ y_2 = -y_0$$
$$\downarrow\ T_3,\ s_3 = 1$$
$$x_3 = -x_0,\ y_3 = -y_0$$
$$\downarrow$$
$$\begin{aligned} x_3 y_3 &= x_0 y_0 \\ &= x(2y-1) \\ &= 2xy - x \end{aligned}$$

Les invariants de l'algorithme AND

À chaque tour, on a :

- $X_i = \mathrm{enc}(x_i)$, pour $x_i = s_i x_{i-1}$;
- $Y_i = \mathrm{enc}(y_i)$, pour $y_i = s_i y_{i-1}$;

et donc à la fin

$$X_n = X^s = \mathrm{enc}(sx),\ Y_n = Y_0^s = \mathrm{enc}(y_n) = \mathrm{enc}(sy_0),$$

où $s = \prod s_i \in \{-1, 1\}$, connu d'aucun T_i.

Ainsi, s **masque parfaitement** y_0 (qui doit rester secret).
Donc le déchiffrement de Y_n en y_n ne dévoile aucune information.

Pour finir :

$$X \times X_n^{y_n} = \mathrm{enc}(x) \times \mathrm{enc}((sy_0) \cdot (sx)) = \mathrm{enc}(x + s^2 x(2y - 1)) = \mathrm{enc}(2xy),$$

avec $2z = 2xy \in \{0, 2\}$ qui reste secret.

NAND est universelle !

La porte NOT est facile car $\text{enc}(1-x) = \text{enc}(1)/\text{enc}(x)$.
On peut donc coder **toute fonction de dépouillement**, pourvu que les électeurs encodent leur choix bit-à-bit.

Exemple : l'**addition d'entiers** sur m bits.

Input : $(X_0, \cdots, X_{m-1}), (Y_0, \cdots, Y_{m-1})$ chiffrements bit-à-bit de x et y

Output : $Z_0, \cdots, Z_{m-1}$, chiffrement bit-à-bit de $x + y$ modulo 2^m

- $R = \texttt{AND}(X_0, Y_0)$
- $Z_0 = X_0 Y_0 / R^2$ (* $x_0 \oplus y_0$ *)
- **Pour** $i = 1$ à $m - 1$:
  - $A = X_i Y_i / \texttt{AND}(X_i, Y_i)^2$ (* $x_i \oplus y_i$ *)
  - $Z_i = AR / \texttt{AND}(A, R)^2$ (* $x_i \oplus y_i \oplus r$ *)
  - $R = (X_i Y_i R / Z_i)^{\frac{1}{2}}$
- Retourner $Z_0, \cdots, Z_{m-1}$

Modèle de coût

Atypique par rapport à un circuit électronique usuel :

- La porte AND requiert des **communications**, avec des points de **synchronisation** entre les T_i.
- Il est intéressant de **paralléliser** pour limiter les synchronisations / masquer les communications.
- Aucun problème de **fan-in / fan-out**.
- Certaines opérations sont **gratuites** ; on peut passer temporairement par des entiers autres que $\{0, 1\}$.
- **Interdiction des branchements** qui dépendent des secrets.

Exemples :

- Circuit d'addition en profondeur logarithmique (Brent-Kung 1982).
- Réseau de tri (tri fusion pair-impair de Batcher 1968 ; souvent utilisé en GPU).

Une boîte à outils MPC pour le dépouillement

Functionality	Option	Algorithm	Exp per trustee	Comm. cost	Transcript size
`Dec`	P/EG	`Dec`	$5a$	B	$4a$
`RandBit`	P/EG	`RandBit`	$3a+2$	R	$6a$
`AND`	EG	`CGate` [39]	$29a$	$R+4B$	$31a$
	P	`Mul` [41]	$10a$	$2B$	$11a$
`Select`	P/EG	`Select`	`AND`	`AND`	`AND`
`SelectInd`	P/EG	`SelectInd`	n`AND`	`AND`	n`AND`
`Neg`$^{\text{bits}}$	P/EG	`Neg`$^{\text{bits}}$	$(m-1)$`AND`	$(m-1)$`AND`	$(m-1)$`AND`
`Add`$^{\text{bits}}$	P/EG	`Add`$^{\text{bits}}$ [39]	$(2m-1)$`AND`	$(2m-1)$`AND`	$(2m-1)$`AND`
	Sublinear P/EG	`UFCAdd`$^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$`AND`	$2(\log m+1)$`AND`	$m(\frac{3}{2}\log m+2)$`AND`
`Sub`$^{\text{bits}}$	P/EG	`Sub`$^{\text{bits}}$	$(2m-1)$`AND`	$(2m-1)$`AND`	$(2m-1)$`AND`
	LT P/EG	`SubLT`$^{\text{bits}}$	$(2m-1)$`AND`	$(2m-1)$`AND`	$(2m-1)$`AND`
	LT+EQ P/EG	`SubLT`$^{\text{bits}}$	$(3m-2)$`AND`	$(2m+\log m)$`AND`	$(3m-2)$`AND`
	Sublinear P/EG	`UFCSub`$^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$`AND`	$2(\log m+1)$`AND`	$m(\frac{3}{2}\log m+2)$`AND`
`LT`$^{\text{bits}}$	LT P/EG	`SubLT`$^{\text{bits}}$	$(2m-1)$`AND`	$(2m-1)$`AND`	$(2m-1)$`AND`
	LT+EQ P/EG	`SubLT`$^{\text{bits}}$	$(3m-2)$`AND`	$(2m+\log m)$`AND`	$(3m-2)$`AND`
	Sublinear P/EG	`CLT`$^{\text{bits}}$	$(4m-3)$`AND`	$2(\log m+1)$`AND`	$(4m-3)$`AND`
	Sublinear+EQ P/EG	`CLT`$^{\text{bits}}$	$(5m-4)$`AND`	$2(\log m+1)$`AND`	$(5m-4)$`AND`
`EQ`$^{\text{bits}}$	P/EG	`EQ`$^{\text{bits}}$	$(2m-1)$`AND`	$(\log m+1)$`AND`	$(2m-1)$`AND`
`EQ`	Precomp P	`EQH` [32]	$21ma+75a$ $+4(m+1)$	$R+8B$	$(22m+28)a$
`GT`	Precomp P	`GTH` [32]	$(27m+146\log m)a$ $+8m+9a+5\log m$	$(2R+13B)\log m$	$(28m+50\log m)a$ $+6a$
`BinExpand`	P	`BinExpand` [40]	$12ma+53a+3m$	$R+2mB$	$(17m+21)a$
`Aggreg`$^{\text{bits}}$	EG	`Aggreg`$^{\text{bits}}$	$3n$`AND`	$(\log n+1)\log n$`AND`	$3n$`AND`
`Mul`$^{\text{bits}}$	P/EG	`Mul`$^{\text{bits}}$	$3m^2$`AND`	$2m^2$`AND`	$3m^2$`AND`
`Div`$^{\text{bits}}$	P/EG	`Div`$^{\text{bits}}$	$(3m-1)r$`AND`	$2mr$`AND`	$(3m-1)r$`AND`
`MinMax`$^{\text{bits}}$	naive P/EG	`MinMax`$^{\text{bits}}$	$(8m-2)n$`AND`	$2m\log n$`AND`	$(8m-2)n$`AND`
	sublinear P/EG	`MinMax`$^{\text{bits}}$	$(12m-6)n$`AND`	$2\log n(\log m+2)$`AND`	$(12m-6)n$`AND`
`Mixnet`	EG	[47]	$(9n+11)a$ $+n-6$	R	$10(n+1)a$
	P	[47]	$(8n+10)a$	R	$10(n+1)a$

Figure 7. Cost of various MPC primitives: basic functionalities for logic, integer arithmetic, and a few advanced functions. The Option column includes whether this is available in Paillier (P) or ElGamal (EG). The notations are a for the number of authorities, m for the bit-length of the operands, n for the number of operands, r for the precision (in the division). All logarithms are in base 2. The communication costs are expressed in terms of broadcast (denoted B) and full rounds (denoted R). The unit of the transcript size is the key length. This corresponds to half the size of a ciphertext in both Paillier

Extrait de *A toolbox for verifiable tally-hiding e-voting systems*, V. Cortier, P. Gaudry, Q. Yang. 2022.

Nombreuses variantes.

Support de Paillier et ElGamal.

Compatible avec systèmes de vote classiques.

Exemple d'application : dépouillement de Condorcet

- Les électeurs **classent** les k candidats ;
- Le classement est codé par un **entier** pour chaque candidat, puis chiffré ;
- Pour le dépouillement, les T_i calculent en MPC la **matrice de préférence** ;
- Puis, le ou les vainqueurs sont obtenus par un calcul en MPC de **plus court chemin** dans un graphe pondéré.
- **Coûts** : $O(k \log k)$ pour les électeurs ; $O(nk^3)$ pour les T_i.

Implémentation, pour $n = 3$ T_i, chacune utilisant 4 threads de calcul :

Électeurs	5 candidats	10 candidats	20 candidats
64	1m50s / 49 MB	8m30s / 0.30 GB	45m / 1.8 GB
128	2m40s / 87 MB	12m / 0.51 GB	1h27m / 2.9 GB
256	4m35s / 160 MB	20m / 0.88 GB	2h37m / 4.8 GB
512	8m10s / 305 MB	34m / 1.6 GB	4h43m / 8.6 GB
1024	15m / 595 MB	1h05m / 3.1 GB	8h50m / 16 GB

Plan

Résumé

Le **vote électronique** est une application rêvée pour la **cryptographie moderne** :

- **Besoin de ZKPs** dès les premières propriétés de sécurité.
- On les retrouve **effectivement déployées** pour les élections à fort enjeu
  - Législatives 2022 et suivantes, pour les Français de l'Étranger
  - Élections par Internet en Suisse, en Estonie...
- Autres techniques pour les **avancées académiques** pas encore déployées :
  - **MPC** (dépouillement en aveugle, mais aussi résistance à la coercition)
  - Approches différentes d'Helios/Belenios : utilisation de *Fully-Homorphic Encryption*, de *Functional Encryption*, de *Verifiable Delay function*, etc.

Et les ZKPs génériques ?

Ce sont des "systèmes de preuve" capables de faire des **ZKPs efficaces** (et très courtes) pour **n'importe quelle propriété**.

Développement très important depuis une décennie, sous l'impulsion de l'écosystème blockchain (Zcash créé en 2016). Exemples : Plonk, Bulletproof, Ligero.

Ont atteint une **maturité impressionnante** :

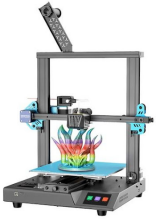
- **Sécurité** avec de moins en moins d'hypothèses.
- Outils pour **compiler automatiquement** la propriété.
- De plus en plus **efficaces**.

90's

$\longrightarrow$

$\approx$ 2010

$\longrightarrow$



aujourd'hui

ZKPs génériques pour le vote électronique

Quelques **cas d'usage** possibles :

- Faire des ZKPs **sortant du monde des morphismes** de groupe.
  E.g. *Je connais un message dont le haché est dans une liste publique de hachés.*
  - Cf par ex. Cortier, Debant, Goestchmann, Hirschi, USENIX 2024. Utilisation d'OpenID pour signer des bulletins de vote.
  - Avoir un tableau d'affichage qui ne contient plus les bulletins complets.
- Aider la **transition post-quantique** :
  - ZKPs complexes sur la bonne formation des bulletins
  - ZKPs complexes pour le dépouillement
  - Cas hybrides classique/post-quantique, où les ZKPs doivent "traverser les mondes"

Les preuves zero-knowledge n'ont pas dit leur dernier mot en vote électronique !