TP3 - Vote électronique

(Evoting) **Exercise 1.** *E-voting El Gamal*

Le vote électronique est un système de votes dématérialisés utilisé à partir des années 90 afin de remplacer l'organisation de votes en présence. Il est aujourd'hui largement utilisé pour des votes à caractère privé (entreprise etc...), mais aussi à des fins politiques, notamment en Estonie. Toutefois, le vote doit respecter les 4 grands principes de la cryptographie. En présentiel, ces principes sont assurés par l'organisation mise en place :

- *Authenticité* $\sim$ dépouillement public
- *Confidentialité* $\sim$ isoloir et enveloppe opaque
- *Intégrité* $\sim$ urne scellée
- *Non répudiation* $\sim$ contrôle d'identité, signature

Dans un cadre électronique, la notion de non répudiation est compliquée, mais il faut tout de même assurer la confidentialité du vote : on ne peut retrouver le vote d'une personne; l'intégrité : le vote ne peut être modifié une fois déposé; et l'authenticité : le vote est le bon et les résultats transmis sont corrects. Afin d'assurer tout ces critères, à chaque vote est associé un identifiant, caractérisant le votant (contrôle d'identité). Avant d'envoyer son vote (le déposer dans l'urne), le votant l'encapsule d'une certaine manière (enveloppe) pour qu'il puisse ensuite être chiffré (l'identifiant assure la signature après chiffrement).

Un leader du vote électronique largement utilisé se nomme Hélios, il est basé sur le chiffrement El-Gamal, qui permet d'évaluer les scrutins sans nécessairement déchiffrer les votes grâce à ses propriétés homomorphes. Cette propriété est largement utilisée afin d'assurer la non divulgation des votes de chaque votant. La notion de *Confidentialité* (à laquelle on ne s'intéresse pas tant ici) est assurée par ce qu'on appelle les Preuves à Divulgation Nulle de Connaissances (Zero Knoledge Proof of Knowledge - ZKPoK), qui vient assurer qu'un votant connait son vote mais sa transmission ne permet pas d'obtenir des informations sur ce dit vote. Cette notion vient remplacer la notion d'*indistinguabilité CCA* et permet d'atteindre une autre sécurité nommée *Non-malléabilité-CPA*. Sans cette notion, un système de vote électronique ne peut être valide et sûr.

L'Agence National de la Sécurité des Système d'Informations (ANSSI) a dévoilé il y a quelques années un guide à des fins d'informations pour le grand public. Ce guide est disponible ici. Pour une meilleure compréhension, il est intéressant pour vous de lire les sections 3.1 et 3.2.1 afin de comprendre quelles sont les nécessités et les impacts d'une numérisation du vote.

Par la suite, on s'intéresse à la mise en oeuvre du chiffrement El Gamal (Annexe D), vers laquelle vous pouvez directement vous diriger. La documentation utilise la notation (k, K) pour le couple clé privée/publique, pour la suite n utilisera plutôt (a, h) par analogie au problème du logarithme discret. Vous choisirez un groupe d'ordre premier pour tester vos fonctions avec les messages (entiers) que vous souhaitez. Attention à bien définir vos ensembles/groupes pour effectuer vos opérations.
On s'intéresse dans un premier temps au chiffrement El Gamal à codage classique, qui permet la vériafiabilité d'un mélange de bulletin (configuration urne).

1. Écrire la fonction `KeyGen` qui prend en entrée un générateur g et l'ordre d'un groupe q et qui retourne un couple de clés privée/publique El Gamal `(sk = a,pk = h)`.

Candidat	Choix	Codage	Chiffré
C_1	7	0	$(g^{r_1}, h^{r_1} g^0)$
C_2	✓	1	$(g^{r_2}, h^{r_2} g^1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_M	✓	1	$(g^{r_M}, h^{r_M} g^1)$

Table 1: Génération des chiffrés suivants le choix du votant

2. Écrire la fonction `Encrypt_cc` qui prend en entrée un générateur g, l'ordre d'un groupe q, une clé publique h et un message m (un entier) et qui retourne le chiffré El Gamal de m sous la clé publique h.

3. Écrire la fonction `Decrypt_cc` qui prend en entrée un générateur g, l'ordre d'un groupe q, une clé secrète a et un chiffré $c = (c_1, c_2)$ et qui retourne le clair sous-jacent m.

Dans le cadre de l'accumulation des bulletins (peut être vu comme l'empilement des bulletin pour le dépouillement), alors il est nécessaire d'avoir un chiffrement additivement homomorphe. Le chiffrement précédent à codage classique est seulement multiplicativement homomorphe, comme vous avez pu le voir en TD. Pour cela, on redéfinit le chiffrement El Gamal en version codage exponentiel : un entier j est codé en un élément du groupe g^j. La génération de clé est identique au cas classique.

4. Écrire la fonction `Encrypt_ce` qui prend en entrée un générateur g, l'ordre d'un groupe q, une clé publique h et un message j (un entier) et qui retourne le chiffré El Gamal de j sous la clé publique h.

5. Écrire la fonction `Decrypt_ce` qui prend en entrée un générateur g, l'ordre d'un groupe q, une clé secrète a et un chiffré $c = (c_1, c_2)$ et qui retourne le clair sous-jacent j.

À partir d'ici, on obtient donc un chiffrement additivement homomorphe qui permet donc à partir de deux chiffrés $c_1 = (\alpha_1, \beta_1) = (g^{r_1}, h^{r_1} g^{j_1})$ et $c_2 = (\alpha_2, \beta_2) = (g^{r_2}, h^{r_2} g^{j_2})$ d'obtenir un chiffré $(\alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2)$ de $j_1 + j_2$.

On considère que pour M candidat.e.s $C_1, ..., C_M$, un.e votant.e à le droit de choisir C candidat.e.s parmis les M, et va ainsi générer M chiffrés de 1 ou de 0 suivant qu'iel choisisse ou non le ou la candidat.e :
Grâce à l'homomorphie additive du chiffrement à codage exponentiel, il est alors possible de compter le nombre de voix qu'un.e candidat.e a obtenues.

6. Écrire une fonction `Add_cipher` qui prend en entrée une liste de chiffrés c_i, chiffrés sous la même clé publique h et qui retourne le chiffré de la somme des messages j_i sous-jacents.

Dans un cadre pratique tel qu'une élection publique (politique), le dépouillement (déchiffrement) est effectué par chaque ville indépendamment les unes des autres. Toutefois, il peut être considéré dans le cas d'un vote électronique que chacune des N villes détient un morceau de la clé privée, et toutes les rassemblent afin de dépouiller le tout, c'est ce qu'on appelle un déchiffement centralisé.
Ainsi, pour que chaque établissement possède un morceau de clé privée, il est nécessaire de générer autant de paires de clés que de mairies et construire une clé publique grâce à toutes les clés publiques générée. Pour ce faire, on construit la clé publique $h = \prod_{i=1}^{N} h_i = \prod_{i=1}^{N} g^{a_i}$, et la clé privée associée est donc la liste de clés privées $(a_1, ..., a_N)$.

7. Écrire la fonction `KeyGen_central` qui prend en entrée un générateur g, l'ordre d'un groupe q et un entier N et qui retourne un couple de clés privées/publique El Gamal $((sk_i)_{i \leqslant N} = (a_i)_{i \leqslant N}, pk = h = \prod_{i \leqslant N} h_i)$. Vous pourrez appeler la fonction `KeyGen` précédente.

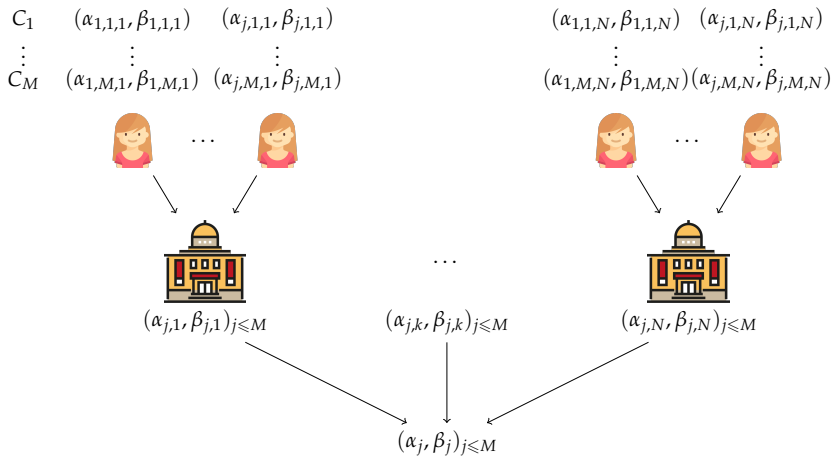

Figure 1: Schéma de centralisation du déchiffrement

On considère une somme de vote de diverses mairie M_k, chiffrés sous clés $h_k = g^{a_k}$ pour une clé secrète fragmentée $(a_1, ..., a_N)$:

$$\left(\prod_{k=1}^{N} \alpha_k, \prod_{k=1}^{N} \beta_k\right) = \left(\prod_{k=1}^{N} g^{r_k}, \prod_{k=1}^{N} h^{r_k} g^{m_k}\right) = \left(g^{\sum_{k=1}^{N} r_k}, h^{\sum_{k=1}^{N} r_k} g^{\sum_{k=1}^{N} m_k}\right) \tag{1}$$

où $r_k = \sum_{i_k} r_{i_k}$ représente le nonce de la mairie k, généré par l'ensemble de ses administrés et m_k la somme des votes des administrés de la mairie k pour le ou la candidat.e courant. Cette somme est obtenue au préalable par homomorphie additive sur le scrutins des administrés. Ainsi, le message est chiffré sous la clé fragmentée $(a_1, ...a_N, g^{\sum_{k=1}^{N} a_k})$, et il est donc nécessaire de déchiffrer correctement le message $\sum_{k=1}^{N} m_k$ sous-jacent. Pour cela, il faut tour à tour éliminer h^{r_k} pour les k mairies de la composante de droite du chiffré. Ainsi, on applique finalement k déchiffrements consécutifs de la composante de droite en utilisant la composante de gauche et la clé privée courante a_k.

	Votant i pour C_j	Mairie k	Total (N mairies)
Chiffré	$(\alpha_{i_k,1}, \beta_{i_k,1})$	$(\alpha_{k,1}, \beta_{k,1}) = (\prod_{i_k} \alpha_{i_k,1}, \prod_{i,k} \beta_{i_k,1})$	$(\alpha_1, \beta_1) = (\prod_k^N \alpha_{k,1}, \prod_k^N \beta_{k,1})$
Message	$m_{i_k,1}$	$\sum_{i_k} m_{i_k,1} = m_{k,1}$	$m_1 = \sum_k^N m_{k,1}$
Clé	h	(a_k, h)	$(a_1, ..., a_N, h = \prod_k^N h_k)$

Table 2: Fragmentation du déchiffrement pour un.e candidat.e j

8. Écrire les fonctions `Decrypt_central_cc` et `Decrypt_central_ce` à l'image des algorithmes 7 et 8 prenant en entrée un générateur g, l'ordre d'un groupe q, une liste de clé secrète (a_i) et un chiffré $c = (c_1, c_2)$ et qui retourne le clair sous-jacent m en codage classique puis exponentiel.

9. Pour finir, écrire une fonction `Depouillement` qui prend en entrée un tableau comportant les scrutins de tou.te.s les candidat.e.s (identifié.e par leur identifiant j) pour chacune des mairies, ainsi qu'une clé privée fragmentée $(a_1, ..., a_N)$ et qui retourne le candidat remportant l'élection.

Pour aller plus loin, vous pouvez vous intéresser à la notion de génération centralisée fragmentée à seuil, qui permet le déchiffrement centralisé avec seulement une partie des clés privées, ce qui permet de consulter le suffrage sans nécessairement la présence de toutes les parties.