HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département TIC
Filière Télécommunications
Orientation Sécurité de l'information

**Travail de Bachelor**

# Chiffrement/Signature d'Emails

**Étudiant** **Mickael Bonjour**
**Enseignant responsable** Prof. Alexandre Duc
**Année académique** 2019-2020

Yverdon-les-Bains, le 23 juillet 2020

Département TIC
Filière Télécommunications
Orientation Sécurité de l'information
Étudiant Mickael Bonjour
Enseignant responsable Prof. Alexandre Duc

Travail de Bachelor 2019-2020

Chiffrement/Signature d'Emails

**Résumé publiable**

Dans ce travail de bachelor je vais analyser les solutions actuelles de messagerie sécurisée afin d'identifier les primitives cryptographiques dont j'ai besoin. Ensuite j'établirais un *Proof Of Concept* amenant la technologie choisie dans un cadre de messagerie électronique sécurisée.

| | | |
|---|---|---|
| Étudiant : | Date et lieu : | Signature : |
| Mickael Bonjour | ..... | ..... |
| Enseignant responsable : | Date et lieu : | Signature : |
| Prof. Alexandre Duc | ..... | ..... |
| Nom de l'entreprise/institution : | Date et lieu : | Signature : |

# Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.



HEIG-VD

Vincent Peiris
Chef de département TIC

Yverdon-les-Bains, le 23 juillet 2020

# Authentification

Le soussigné, Mickael Bonjour, atteste par la présente avoir réalisé ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Yverdon-les-bains, le 23 juillet 2020

Mickael Bonjour

# Cahier des charges

## Résumé du problème

Les outils de chiffrement et de signature d'email actuels se résument principalement à S/-MIME et à PGP.
Ces deux solutions sont anciennes, souffrent assez régulièrement de nouvelles vulnérabilités et ne proposent pas certaines propriétés cryptographiques qui pourraient être utiles (par exemple, la "forward secrecy". Le but de ce travail de bachelor est d'étudier quelles propriétés seraient utiles pour la sécurisation des emails, de proposer un nouveau protocole les implémentant et de développer un proof of concept.

### Problématique

La problématique principale est résumée ci-dessus mais le principal problème c'est surtout que les technologies utilisées sont vieilles et elles souffrent de vulnérabilités par conception qui ont été mitigées en enlevant des options à l'utilisateur.

### Solutions existantes

PGP, S/MIME, PEP, messagerie instantanée (Signal)...

### Solutions possibles

Une solution possible est d'utiliser le système mis en place dans ce travail, cependant il faudrait une relecture et des analyses plus approfondies pour s'en assurer. Un des points possibles c'est de passer plus à de la messagerie instantanée dans la mesure du possible. Ou encore de s'orienter sur des nouvelles technologies comme PEP ou PGP mais en appliquant strictement les *Best Practices* et en se formant un peu sur leur utilisation qui n'est pas donnée à tout le monde.

# Cahier des charges

Voici un résumé du cahier des charges sous formes d'objectifs à atteindre :

- Analyser les besoins d'un système d'E-mails actuel.
- Analyser et étudier les solutions de sécurité existantes.
- Comprendre et évaluer les propriétés cryptographiques défendues.
- Établir une liste des propriétés cryptographiques voulues pour un système de mails sécurisés.
- Trouver une primitive cryptographique satisfaisant les besoins énoncés et l'étudier pour en comprendre les bases et les besoins nécessaires en termes de sécurité.
- Établir la spécification pour un nouveau protocole en utilisant la primitive choisie.
- Faire un Proof Of Concept du protocole proposé.

Si le temps le permet :

- Comprendre plus en détails les mathématiques derrière la primitive utilisée.
- Faire un prototype de client mail utilisant une architecture mise en place pour le POC.

## Déroulement

Tout d'abord je vais m'intéresser à faire une évaluation des concepts existants en messagerie sécurisée, tel que PGP et S/MIME pour les emails ou encore Signal pour la messagerie instantanée. Ayant vu ce qu'il se fait j'essaie de trouver une solution alternative pour le chiffrement et la signature d'emails. De là je vais conceptualiser un protocole et l'implémenter au sein d'un *Proof Of Concept*.

## Livrables

Les délivrables seront les suivants :

1. Une documentation contenant :
   - Une analyse de l'état de l'art
   - La décision qui découle de l'analyse
   - Spécifications
   - L'implémentation faites et les choix faits
   - Proof Of Concept
   - Les problèmes connus
2. Le code du *Proof Of Concept* fait, expliqué à l'aide de commentaires.

# Table des matières

# Chapitre 1

# Introduction

Ce travail de Bachelor a pour but de sensibiliser à la vulnérabilité dans les systèmes actuels de messagerie électronique. Il propose aussi un nouveau protocole permettant de sécuriser ce type de messagerie à l'aide d'une primitive cryptographique peu implémentée, le *Certificateless Public Key Cryptography*. Ma démarche dans ce travail de bachelor est de voir si des solutions s'offrent à nous en considérons ce qui se fait sur le marché actuellement. En essayant d'améliorer les solutions actuelles proposées qui peuvent souffrir d'un manque de sécurité assez souvent ou (et plus souvent) un manque de simplicité d'utilisation.
Ce travail est découpé en plusieurs parties. En effet, on commence par une analyse de l'état de l'art, ce qui existe et voir pourquoi il faudrait de nouvelles solutions. Puis une présentation de la primitive cryptographique utilisée pour ma proposition dans ce travail. Enfin la présentation de l'architecture de mon protocole est une implémentation proposée en *Proof Of Concept* ainsi que les choix importants qui ont été faits en rapport à cette implémentation.

# Chapitre 2

# Analyse - État de l'art

La problématique principale est la difficulté d'utilisation des sécurités mises en places au-dessus des protocoles de base pour les emails. De plus des vulnérabilités (EFAIL en 2.3.2) qui réussissent à récupérer le texte clair a démontré que la sécurité n'était pas bien implémentée. Les vulnérabilités proviennent plus d'un défaut de conception inhérent aux mails. Je vais ici décrire les principaux problèmes trouvés sur PGP et S/MIME lors de mes tests d'utilisation et ce que j'ai trouvé durant mes recherches. De plus je vais analyser des systèmes de mails sécurisés tel que Protonmail et Tutanota.

## 2.1 Protocoles existants

Ici je vais regarder les protocole existants afin de mettre en place des liaisons sécurisées de messagerie éléctronique.

### 2.1.1 PGP

**Fonctionnement.** PGP (Pretty Good Privacy ou Assez bonne confidentialité) est un moyen de chiffrer des données (mails, fichiers, ...) qui est beaucoup représenté lorsque l'on parle de sécurité email car c'est le plus utilisé avec S/MIME (c.f. 2.1.3). C'est une méthode de chiffrement hybride (utilise le chiffrement symétrique et assymétrique) qui fonctionne comme montré sur la Figure 2.1.

Ce fonctionnement hybride est défendu à cause de la lenteur et la non-praticité d'un chiffrement asymétrique sur un certain nombre de données. Ainsi en chiffrant uniquement la clé symétrique qui a servi à chiffrer le tout l'on peut déchiffrer bien plus rapidement et simplement le message (typiquement avec un chiffrement symétrique tel qu'AES qui a le droit à des instructions dédiées dans certains processeurs). Contrairement à des chiffrements

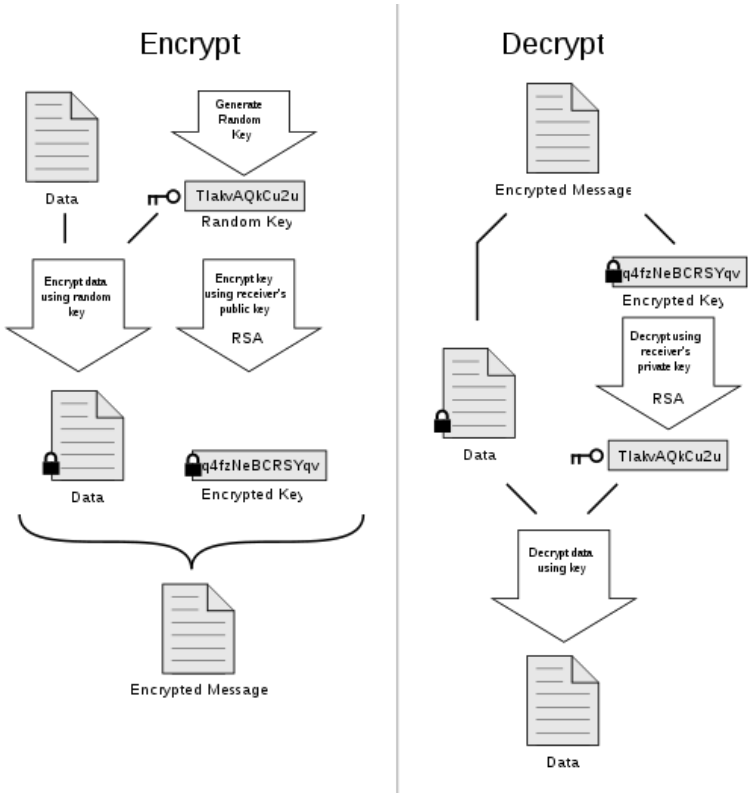


FIGURE 2.1 – Le fonctionnement global de PGP

asymétriques qui sont plus contraignants. Et l'on n'utilise pas directement le chiffrement symétrique car il a besoin d'un secret partagé dès le début de la communication. PGP utilise un système de clés... PGP est aussi critiqué pour son manque de "Forward Secrecy"...

**Propriétés cryptographiques.** Le problème qui est souvent reproché à PGP c'est qu'il n'implémentes pas de *Forward Secrecy*. La *Forward Secrecy* permet d'affirmer que si l'on a une brèche à un instant $T$, et qu'un attaquant récupère cette clé, il ne pourra pas déchiffrer les anciens messages. De plus, la gestion des clés PGP est très problématique, en effet lors de mes tests il était difficile de connecter un serveur de clés p. ex. Ou de recevoir une clé d'un correspondant pour la sauvegarder. Et même en la recevant, comment savoir si cette clé n'a pas été modifiée via un *MITM* p.ex. ? -> utiliser un autre canal pour vérifier l'empreinte.

**Web of Trust.** Comment faire confiance a une clé -> surtout pour email -> Comment initialiser une confiance ?

**Autocrypt.** Autocrypt est une manière d'échanger des clés entres emails, ces échanges ne sont pas considérés sécurisé par la communauté (Wikipedia -> à creuser). C'est une façon de s'échanger des clés de manière automatisée mais pas forcément sécurisée (utilise les mails).

**Utilisation.** Pour mes tests j'ai fait en sorte de trouver l'utilisation la plus simple possible pour voir si un utilisateur lambda pouvait arriver à mettre en place ce genre de sécurité. Il s'est avéré que cela était assez simple au départ, mais dès lors que l'on veut envoyer un mail chiffré à un correspondant cela se complique un peu. J'ai juste eu à installer un Add-On sur mon logiciel de messagerie (Thunderbird dans mon cas) qui s'appelle Enigmail. Ensuite Enigmail a générer mes clés PGP (de manière totallement opaque -> à creuser). Puis j'ai écrit un mail, en appuyant sur un petit cadenas mon mail partait chiffré et signé (uniquement si on a la clé du correspondant). Bien, cependant c'est très opaque et on ne sait pas ce qu'Enigmail et Autocrypt font réellement derrière les décors. L'utilisateur doit encore choisir s'il veut chiffrer ses mails ou non par contre il faut que le destinataire utilise PGP et que l'on ait sa clé publique. J'ai donc expérimenté à plus bas niveau ce qu'il se passait.

### 2.1.2 PEP

**Citation.** *By default, communications between pep peers always work end-to-end encrypted – no eavesdrop-ping in between shall be possible by design.*

**Utilisation.** pep assure un chiffrement de bout-en-bout par design, ils n'ont en effet pas de serveurs en soit et chiffre à l'aide d'un *handshake* fait entres les deux personnes via des *trustwords*. Ce sont des mots qu'il faut vérifier entres les deux partis afin d'être sûr que la connexion est bien authentifiée. -> à creuser mais à priori PEP utilise PGP pour le chiffrement des messages.

### 2.1.3 S/MIME

**Fonctionnement.** Basé sur le même principe que PGP principalement, mais avec des certificats pour prouver la légitimité des clés publiques. Pour l'utilisation il faut se créer un certificat, plusieurs classes de confiance existe.

**Propriétés cryptographiques.**

**Utilisation.**

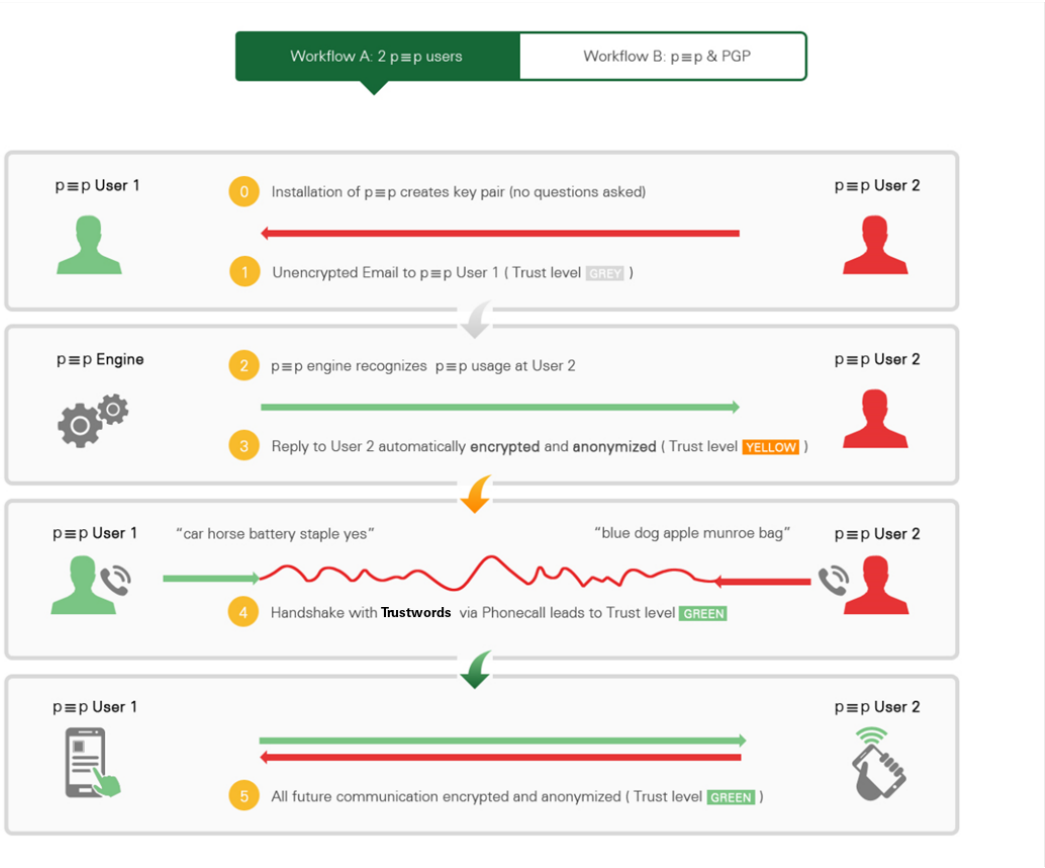


FIGURE 2.2 – Le fonctionnement global de pEp

## 2.2 Implémentations existantes

### 2.2.1 Protonmail

**Revendications.** Protonmail revendique beaucoup de propriétés cryptographiques, tel que le zero-access encryption. Et l'end-to-end chiffrement + zero-knowledge pour les messages sécurisés, même avec leur fonctionnalité de (Chiffrement vers l'extérieur) utilisant AES256-GCM. Pour l'authentification Protonmail utilise une manière fortement sécurisée (SRP) pour ne pas avoir d'informations direct sur le mot de passe de l'utilisateur.

**Fonctionnement.** Protonmail a plusieurs modes de fonctionnement dépendant du destinataire final. En effet de Protonmail à Protonmail les mails sont chiffrés à l'aide de PGP automatiquement. L'on peut utiliser Protonmail pour utiliser PGP si l'on a la clé de notre destinataire par exemple. Et l'on peut écrire un mail chiffré à quelqu'un qui n'utilise pas

PGP grâce à une fonctionnalité de chiffrement vers l'extérieur. Cette fonctionnalité enverra une URL au destinataire qui, en la consultant, pourra déchiffrer le mail en utilisant un mot de passe communiqué de manière sécurisées entres les deux partis auparavant.

**Open Source.** Tout leur code est open-source afin d'avoir une validation externe, de plus ils ont un programme de Bug Bounty pour les chercheurs.

### 2.2.2 Tutanota

**Fonctionnement.** Tout ce que j'ai vu pour le moment c'est que Tutanota utilise AES128-CBC ? Mais dans PGP ou ailleurs ?

## 2.3 Attaque existantes

### 2.3.1 Défauts webmail

Selon un chercheur [9] l'infrastructure de Protonmail aurait des failles via son webmail. Mais son papier est en fait plus général et parle des webmails en règle général. Il part du principe que les serveurs de Protonmail ne sont pas des serveurs à faire confiance, pour ainsi prouver le zero-knowledge de Protonmail. Par contre, le fait qu'il ne peuvent pas être mis en confiance est un problème selon lui, car c'est ces serveurs qui vont délivrer le code d'OpenPGP afin de faire le chiffrement. Cela indique que si Protonmail était corrompu le fait d'avoir le code délivré par Protonmail pourrait avoir des effets néfastes. Comme p.ex l'extraction de la clé privée PGP. La conclusion est que dès le moment où vous avez utilisé une fois le webmail de protonmail la clé PGP est corrompue.

### 2.3.2 EFAIL

Malgré ces sécurités qui pourraient être mises en place à l'heure actuelle, une attaque nommée EFAIL [11] a été faite en 2018 et est toujours possible aujourd'hui(à vérifier / tester). En effet cette attaque a seulement été mitigée en évitant d'afficher les contenus HTML et les images dans boites mails de base. Car le problème vient de là principalement, des problèmes sont liés aussi aux modes de chiffrement utilisé (typiquement CBC et CFB) grâce à des "gadgets". Cette attaque permet en fait d'injecter une image dans l'HTML du message (typiquement dans les headers du mail), puis faire en sorte de récupérer le contenu du message déchiffrer dans un paramètre de l'URL.

## 2.4 Signal

L'analyse s'est faite aussi pour la messagerie instantanée à cause de sa ressemblance avec la messagerie électronique.

### 2.4.1 Fonctionnement

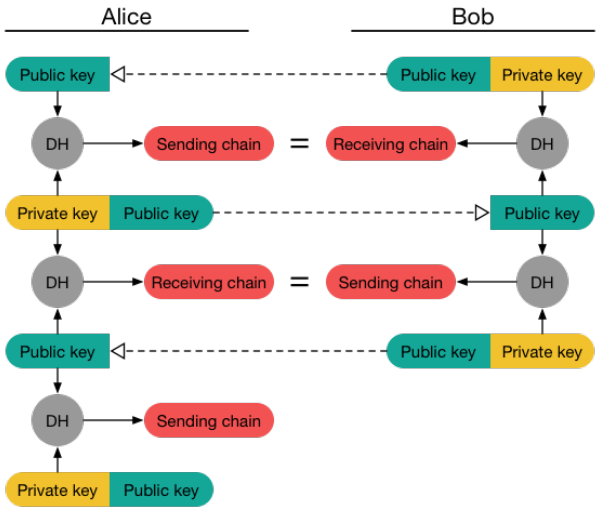


FIGURE 2.3 – Schéma fonctionnement de Signal[10]

### 2.4.2 Problèmes d'intégrations

Le problème avec le protocole Signal quant à mes besoins niveaux mails est la *forward secrecy* qui est très fort. En effet comme vu dans le chapitre précédent il utilise une clé par message grâce au *Double Ratchet*. Cependant ce fonctionnement comporte un gros problème en rapport aux mails, en effet si l'on veut pouvoir récupérer les anciens mails reçus/envoyés cela devient vraiment compliqué. En effet, la *forward secrecy* est une propriété utile dans un système de mail, mais faut pouvoir aussi récupérer les messages facilement si l'on connait la clé privée.

## 2.5 Compromis

Pour passer à l'implémentation concrète d'un nouveau protocole il faut faire des compromis et aller chercher dans des primitives moins connues.
Je suis tout de même rester sur un système de clés publiques comme PGP le fait. Cependant cette primitive a une identité propre à chaque clé publique ce qui évite un systéme de certificat trop complexe.
De plus pour avoir une *forward secrecy* l'on peut ajouter une notion de temps ou de token à l'ID pour chaque batch de messages.

### 2.5.1 Résultats des recherches

Comme mentionné avant les recherches ont beaucoup été orientées sur le protocole Signal qui a une très bonne forward secrecy, résilience et break-in recovery. Cependant le problème avec l'utilisation des mails c'est d'avoir envie de consulter tout ces mails depuis n'importe quel appareil. Ce n'est malheureusement pas le cas avec Signal à moins de conserver une *root key* quelque part qui ferait s'effondrer les caractéristiques principales du protocole.
S/MIME est la solution prédominante pour s'envoyer des mails chiffrés cependant il est compliqué de l'utiliser. Il et en effet difficile d'obtenir un certificat pour envoyer des mails et d'échanger avec une autre personne ayant S/MIME. De plus la complexité d'un systéme de PKI et l'overhead induit est assez conséquent.
En faisant quelques essais PGP de mon côté je me suis heurté à beaucoup de difficultés et de problèmes avec les clés PGP, notamment pour se les échanger mais pour envoyer ensuite ce n'est pas si complexe, le problème étant de bien voir les primitives utilisées pour chiffrer/signer notre email, en effet les solutions *plug-and-play like* ne permettent pas une gesion précise des primitives cryptographiques utilisées.

## 2.6 Primitives

### 2.6.1 Primitives analysées

- Certificateless PKC [1]
- HIBE - Hierarchised Identity Based encryption
- Identity based encryption [12]

### 2.6.2 Primitive choisie

- Certificateless PKC [1]
Cette primitive a été choisiec car elle est similaire à de l'identity based avec un ID pour

désigner une clé publique. Le problème avec l'identity based encryption c'est le fait que le serveur central génère la clé publique et la clé secrète de l'utilisateur, cela amène ce qu'on appelle le *key escrow* problème. C'est le fait qu'une entité connaisse à elle seule toutes les clés de tout les utilisateurs. Ce problème est résolu dans le certificateless en introduisant des *Partial Private Keys* permettant d'avoir une clé secrète partiellement générée par le serveur (KGC - Key Generation Center) et par l'utilisateur puis assemblée pour former la clé privée seulement connue de l'utilisateur. De plus, cette primitive a l'avantage de ne pas introduire de certificats et ainsi évites la complexité d'une infrastructure de PKI (Public Key Infrastructure).

## 2.7 Recherches sur la primitive

Dans cette section je vais introduire les détails techniques et les principes mathématiques utilisés. De plus, le choix de schéma parmi tous ceux analysés est détaillé ici.

### 2.7.1 Principes mathématiques

Les variantes de *Certificateless Cryptography* choisies utilisent un concept appelé les *pairings* ou *bilinear map groups*. Des groupes tels que $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ d'un ordre premier $p$ pour lesquels il existe un mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ avec les propriétés suivantes :
1. Bilinéarité : $e(g^a, h^b) = e(g, h)^{ab}$ pour tout $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$ et $a, b \in \mathbb{Z}$ ;
2. Pas de dégénérescence : $e(g, h) \neq 1_{\mathbb{G}_T}$ tant que $g, h \neq 1_{\mathbb{G}_{1,2}}$ ;

### 2.7.2 À savoir

Avant d'analyser les différents schémas je vais présenter les différenets facteurs présentés dans les tableaux et ainsi établir une légende de ceux-ci expliquée.
Types : Les types présentés peuvent être soit concret soit générique. Les types concrets sont des schémas qui présentent leurs algorithmes en utilisant des calculs bien établis et présentent l'entierté du fonctionnement de leur schéma, tandis que les schémas présenté génériques peuvent s'appuyer sur d'autres problèmes et se baser sur des algorithmes déjà existants.
Modèles de sécurité : Ces modèles définissent sur quoi le schéma va se reposer pour établir sa sécurité et comment il va l'évaluer face à un adversaire. À nouveau il existe deux modèles présents dans les schémas analysés, le *Random Oracle Model* et le *Standard Model*. Le *Random Oracle Model* se base sur des oracles aléatoires mais est un peu controversé, en effet l'aléatoire cryptographiquement sûr est difficile à atteindre, ainsi habituellement le *Random Oracle Model* implémente ces oracles via des fonctions de hachage. Le modèle standard se base lui sur des problèmes mathématiquement difficiles tel que DDH (Decisional

Diffie Hellman).
Différentes sécurité : Pour évaluer les schémas de certificateless public key cryptography il y a différents niveaux de sécurité établis pour 2 types d'adversaires différents. Ces adversaires ont été décrits dans le papier d'Al-Riyami-Paterson [1] pour la première fois. Ces deux adversaires sont de Type I (*outsider adversaries*) et de Type II (*honest but curious KGC*). L'adversaire de Type I est appelé *outsiders* et est permis de remplacer des clés publiques, obtenir des clés partiels privées, et des clés privées puis faire des requêtes de déchiffrements. L'adversaire de Type II est en fait un KGC connaissant la Master Secret Key et qui peut donc générer des PPK, obtenir des clés privées et faire des requêtes de déchiffrement tout en faisant confiance à ce KGC pour pas qu'il ne remplace de clés publiques. Pour chacun des adversaires il existe différents niveaux de sécurité :

### 2.7.3 Schémas Certificateless de Chiffrement

Pour choisir parmi les nombreux schémas existants en certificateless pour le chiffrement j'ai établi un tableau comparatif des différentes manières de faire, inspiré de [13]. En suivant ce tableau je me suis rendu compte que la construction de Dent-Libert-Paterson [5] était probablement la plus adaptée en vue des propriétés qu'elle présentait. Le tableau se trouve en annexe B.

### 2.7.4 Détails techniques

Les détails techniques sur le chiffrement avec la *Certificateless Cryptography*. Le chiffrement se base sur le problème difficile *The Decision 3-Party Diffie-Hellman Problem* (3-DDH).
C'est de décider si $T = g^{abc} ayant (g^a, g^b, g^c, T) \in \mathbb{G}_4$.
Pour expliquer les détails techniques je vais ici montrer les calculs faits dans le schéma choisi [5] et les expliquer, cependant dans le schéma il est noté les calculs avec $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ mais il est mentionné que c'est facilement adaptable pour $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ ce que j'ai fait. De plus, la conversion vers un groupe additif (travaillant sur les courbes elliptiques) est faite afin que les calculs ici puissent être lus avec mon implémentation :
**Setup**$(1^k, n)$ **:** Avec $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ avec un ordre $p > 2^k$. $g$ est un générateur de $\mathbb{G}_1$. Ensuite $g_1 = g * \gamma$ pour un $\gamma \leftarrow \mathbb{Z}_p^*$ aléatoire. Puis $g_2 \leftarrow \mathbb{G}_2$. Deux vecteurs (U,V) seront tirés aléatoirement dans $\mathbb{G}_2^{n+1}$ en tant que fonctions de hash notés :

$$F_u(ID) = u' \sum_{i=1}^{n} u_j^{i_j} \quad \text{and} \quad F_v(w) = v' \sum_{i=1}^{n} v_j^{w_j}$$

L'on va aussi prendre une fonction de hash résistante aux collisions : $H : \{0,1\}^* \to \{0,1\}^n$.
Au final notre *mpk* (master public key) est :

$$mpk \leftarrow (g, g_1, g_2, U, V)$$

Et le $msk$ (master seret key) est $msk \leftarrow g_2 * \gamma$.
**Extract**$(mpk, \gamma, ID)$ **:** On prend $r \leftarrow \mathbb{Z}_p^*$ puis on retourne $d_{ID} \leftarrow (d_1, d_2) = (g_2 * \gamma + F_u(ID) * r, g * r)$
**SetSec**$(mpk)$ **:** Retourne un secret aléatoirement choisi $x_{ID} \leftarrow \mathbb{Z}_p^*$.
**SetPub**$(x_{ID}, mpk)$ **:** Retourne $pk_{ID} \leftarrow (X, Y) = (g * x_{ID}, g_1 * x_{ID})$.
**SetPriv**$(x_{ID}, d_{ID}, mpk)$ **:** On choisit $r' \leftarrow \mathbb{Z}_p^*$ puis on reprends $(d_1, d_2) \leftarrow d_{ID}$ et l'on va prendre en secret key :

$$sk_{ID} \leftarrow (s_1, s_2) = (d_1 * x_{ID} + F_u(ID) * r', d_2 * x_{ID} + g * r')$$

Avec $sk_{ID}$ étant la clé secréte de l'utilisateur, donnée par l'Extract (notre Partial Private Key) et la valeur secrète de SetSec.
**Encrypt**$(m, pk_{ID}, ID, mpk)$ **:** Pour chiffrer $m \in \mathbb{G}_T$, l'on va reprendre $(X, Y) \leftarrow pk_{ID}$. Pour chiffrer ce message on va tiré aléatoirement $s \leftarrow \mathbb{Z}_p^*$ pui calculer :

$$C = (C_0, C_1, C_2, C_3) \leftarrow (m + e(Y, g_2) * s, g * s, F_u(ID) * s, F_v(w) * s)$$

Où $w \leftarrow H(C_0, C_1, C_2, ID, pk_{ID})$.
**Decrypt**$(C, sk_{ID}, mpk)$ **:** L'on peut reprendre $(C_0, C_1, C_2, C_3) \leftarrow C$ et la clé privée $(s_1, s_2) \leftarrow sk_{ID}$. Afin d'accélerer le déchiffrement le calcul suivant peut être fait en tirant une valeur aléatoire $\alpha \leftarrow \mathbb{Z}_p^*$ :

$$m = C_0 + \frac{e(s_2 + \alpha * g, C_2) * e(\alpha * g, C_3)}{e(C_1, s_1 + F_u(ID) * \alpha + F_v(w) * \alpha)}$$

Qui donnera $m$ le texte en clair si le chiffré était bien formaté ou un élément aléatoire dans $G_T$.

### 2.7.5 Schémas Certificateless de Signature

Pour choisir parmi les nombreux schémas certificateless pour la signature j'ai établi un tableau comparatif des différentes manières de faire inspiré de [13]. En analysant les différentes possibilités dans ce tableau il y a peu de solutions se dégage, en effet l'on peut voir que beaucoup de schémas de signature sont cassés, mon choix s'est porté au final sur la construction de Zhang et Zhang [16] pour des signatures robustes en Certificateless. J'ai pris cette construction car elle est résistante au Malicious KGC (si le KGC a été setup avec des paramètres vulnérables) datant de 2006 et n'a pas été cassée depuis. Le tableau se trouve en annexe B.

### 2.7.6 Détails techniques

Les détails techniques sur la signature avec la *Certificateless Cryptography*. La signature se base sur le problème difficile *The ComputationalDiffie-Hellman Problem* (CDH).

Ayant $P, aP, bP$ où $a, b$ aléatoires $\in \mathbb{Z}_q^*$ il n'est pas possible de trouver $abP$.
Pour expliquer les détails techniques je vais ici montrer les calculs faits dans le schéma choisi [16] et les expliquer. Cependant dans le papier original le groupe bilinéaire de couplage choisi est de forme $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ avec un *pairing* $e(g, h)$ avec $g \in \mathbb{G}_1, h \in \mathbb{G}_2$ alors que le papier annonce une construction tel que $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.
**Setup**$(1^k)$ **:** Tout d'abord l'on va prendre les groupes d'ordre $q$ énoncés auparavant. Puis on choisit un générateur $P \in \mathbb{G}_1$. La *master secret key* va être choisie aléatoirement $s \in \mathbb{Z}_q^*$. Puis la clé publique calculée : $P_{pub} = sP$. Finalement, trois fonctions de hash distinctes $H_1, H_2, H_3$ vont être choisies, chacune d'elle *mappant* de $\{0,1\}^*$ à $\mathbb{G}_2$. Pour cela j'ai choisi de faire du *Hash Domain Separation* comme expliqué dans le Chapitre 4. L'on définit les **params** $= (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P, P_{pub}, H_1, H_2, H_3)$
**Partial-Private-Key-Extract**$(params, s, ID_A)$ **:** Pour avoir la *Partial Private Key* $(D_A)$ de l'user $A$ avec l'identité $ID_A$. Calculer $Q_A = H_1(ID_A)$. Alors $D_A = sQ_A$.
**Set-Secret-Value :** La valeur secrète $x \in \mathbb{Z}_q^*$ est tirée aléatoirement.
**Set-Public-Key**$(params, x)$ **:** La clé publique $PK_A$ de l'utilisateur $A$ est $PK_A = xP$.
**Set-Private-Key**$(params, D_A, x)$ **:** La clé privé $SK_A$ de l'utilisateur $A$ est calculée comme ceci $SK_A = (D_A, x)$.
**CL-Sign**$(params, m, ID_A, SK_A)$ **:** Tout d'abord $r \in \mathbb{Z}_q^*$ est tiré aléatoirement puis on calcules les 2 composantes de la signature :

$$U = rP$$

$$V = D_A + rH_2(m, ID_A, PK_A, U) + xH_3(m, ID_A, PK_A)$$

Ainsi ces composantes forment la signature $\sigma = (U, V)$.
**CL-Verify**$(params, PK_A, m, ID_A, \sigma)$ **:** Tout d'abord l'on va calculer $Q_A = H_1(ID_A)$ puis effectuer ce calcul afin de vérifier la signature :

$$e(V, P) == e(P_{pub}, Q_A) * e(U, H_2(m, ID_A, PK_A, U)) * e(PK_A, H_3(m, ID_A, PK_A))$$

## 2.8 État de l'art

Dans cette section je vais analyser et comparer les différentes solutions trouvées utilisant la *Certificateless Cryptography* dans une implémentation concrète pour sécuriser des envois d'emails. J'ai trié les articles suivants par date de publication afin de voir comment les suivants ont repris les technologies etc.

### 2.8.1 Email Encryption System Using Certificateless Public Key Encryption Scheme

Analyse de l'article [6].

**Description.** Cet article présente une façon de faire pour chiffrer les mails à l'aide de *Certificateless Cryptography*. Il va d'abord comparer 6 schémas pour choisir celui à utiliser par rapport à ses propriétés. Ensuite il va comparer les différents algorithmes au niveau du temps avec une implémentation simple en J2SE.

**Détails techniques.** Les détails techniques ne sont pas très fourni dans cet article, en effet, il est mentionné uniquement le choix du schéma (Whang-Huang-Yang). Puis une comparaison des temps entre les différent algorithmes de la primitive. Finalement ils présentent la différence de temps entre le chiffrement du message via le certificateless et via une clé AES qui est chiffrée avec le certificateless.

**Conclusion.** Ce papier nous conforte dans l'idée de l'utilisation d'AES pour la rapidité du chiffrement qui va avec cette primitive. Cependant, ils n'expliquent pas comment la clé AES est prise et chiffrée réellement. Une implémentation existe en J2SE mais je ne l'ai pas trouvée. Le schéma choisi l'a été pour son avantage de ne pas utiliser les *pairings* et est donc plus rapide. Puis parmi les autres schémas qui n'utilisent pas les *pairings* à ce moment là, un est de type générique et l'autre est vulnérable aux *outsider attacks*.

### 2.8.2 An End-To-End Secure Mail System Based on Certificateless Cryptography in the Standard Model

Analyse de l'article [1] .

**Description.** Cet article présente une façon de chiffrer et signer dans un systéme de mail avec le schéma original de *Certificateless Cryptography* à savoir le schéma d'Al-Riyami et Paterson [1]. Un article complet définissant bien le contexte de mails et formalisant pour la première fois un moyen de chiffrer et signer des mails avec de la *Certificateless Cryptography*. Cela en expliquant dans les détails comment ils feraient, sans implémentations citées de ce schéma.

**Détails techniques.** Les détails techniques intéressant dans ce papier est la manière d'encapsuler la clé de chiffrement du message. Sinon le reste s'appuie sur le schéma d'Al-Riyami et Paterson. Pour établir une clé de chiffrement symmétrique afin de chiffrer le mail l'n va tout d'abord tirer une valeur aléatoire $t \in \mathbb{Z}_p$ puis la chiffrer avec CL-PKC en utilisant la clé publique du destinataire $t* = Enc_{P_B}$. Ce $t^*$ sera envoyé avec l' email. Pour en tirer une clé symmétrique on va établire : $K_{AB} = tx_A P_B$ à l'aide de la clé privée de la source et la clé publique du destinataire et enfin la valeur aléatoire tirée auparavant. Puis l'on va calculer la clé symmétrique $K = H_2(Q_A||Q_B||K_{AB})$.

1. `https://www.ijcsi.org/papers/IJCSI-10-2-3-264-271.pdf`

**Conclusion.** Ce papier est assez complet concernant la partie fonctionnement des mails en globalité et offres une bonne idée pour la construction d'une clé symmétrique par mail envoyé. Cependant la mise en place de la clé symmétrique et la preuve de son fonctionnement n'est pas très explicitée. D'ailleurs il y a selon moi une erreur dans le papier original pour la logique de déchiffrement de $t^*$ et de la récupération de la clé symmétrique. De plus, le systéme de signature d'Al-Riyami et Paterson a été cassé par [8].

### 2.8.3 Practical Implementation of a Secure Email System Using Certificateless Cryptography and Domain Name System

Analyse de l'article [4].

**Description.** Ce papier traite le problème de la même façon que le précédent mais essaies d'aller plus loin dans les détails d'un implémentation à plus grande échele (utilisation DNS). Il reprend le même schéma et les mêmes principes pour la création de la clé symmétrique de chiffrement. Le même schéma de signature est présent aussi, qui est cassé rappelons-le. Le but serait d'avoir une entrée DNS similaire au DKIM déjà utilisé pour les emails afin d'informer les utilisateurs quelle adresse donne es clés publiques du domaine en question.

**Détails techniques.** Beaucoup de détails concernant les domain policies qui pourraient être appliqués aux domaines pour la distribution des clés publiques. Proposition d'utiliser les headers d'emails pour transmettre la signature de l'email et informer le destinataire si l'email est chiffré ou non et de transmettre les IDS utilisés et le TImestamp utilisé. En effet, l'introduction d'un timestamp est proposé ici pour avoir un temps d'expiration au mail. Les DOmain Policies sont là pour informer les utilisateurs si les emails de ce domaines doivent être signés/chiffrés ou non.

**Conclusion.** Une implémentation est citée utilisant la librairie MIRACL et en utilisant le C++ comme langage de programmation. L'implémentation est citée comme extension *Thunderbird* en C++ / Javascript. Mais il n'y a pas de réel guide pour implémenter cela au monde réel avec des exemples de configuration DNS et autres. Pas vraiment d'explications sur l'utilisation d'une multitude de KGC ou un seul central, comment les synchroniser et autres... Par contre beaucoup d'explications sur comment pourrait fonctionner une entrée DNS afin d'informer aux utilisateurs où aller pour récupérer les clés publiques des utilisateurs du domaine en question et des policies qui pourraient s'appliquer à ce domaine.

### 2.8.4 PriviPK : Certificate-less and secure email communication

Analyse de l'article [2].

**Description.** Cet article propose une implémentation très concréte utilisant CL-PKC pour communiquer de manière sécurisée dans la messagerie éléctronique. Il attaque beaucoup d'aspects que les autres papiers n'ont pas mentionnés comme la *key transparency*. Le papier insiste sur la transparence du protocole pour l'utilisateur afin qu'il n'ait pas d'opérations fastidieuses à faire (comme c'est le cas dans PGP et S/MIME par exemple). Ce papier s'appuie sur un systéme de *key agreement* proposé dans la littérature de la cryptographie basée sur l'identité.

**Détails techniques.** CONIKS serveur, authentification via les clients mails déjà existants (gmail et yahoo), mise en place d'un systéme de key agreement id-based repensée pour le certificateless.

**Conclusion.** Ce papier est assez intéressant et c'est la seule véritable implémentation que j'ai trouvée, il y a un repo sur github. Cependant il s'appuie sur du *key agreement*. Comme le prochain systéme d'ailleurs. Par aileurs, il insiste sur la transparence et sur l'utilisation des authentifications déjà présentes sur les clients emails.

### 2.8.5 A certificateless one-way group key agreement protocol for end-to-end email encryption

Analyse de l'article [14].

**Description.** Dans cet article les auteurs présentent un moyen d'avoir une clé partagée entre n-partis et avec un seul message, ce qui permet dans un systéme de mail d'avoir qu'à envoyer un mail avec les informations nécessaires pour recomposer la clé partagée. Cette clé partagée est utilisée afin de chiffrer le mail et de l'envoyer ensuite avec les informations nécessaires à la création de la clé partagée. De plus, le systéme est n-parti, cela veut dire que l'on peut envoyer le mail à n personnes et le chiffrer avec la même clé. On enverra juste pas les mêmes informations de créations de la clé partagée à tous.

**Détails techniques.** Pour ce qui est des détails techniques on peut voir que le principe est de créer une clé partagée à l'aide des différents ID et clés publiques des destinataires. On aura une sous-clé $x_i$ pour chaque utilisateur $i$. L'on va construire un $y_i = x_0 \dots x_{i-1} + x_{i+1} \dots x_n$ pour un utilisateur où l'on additionnera tout les $x$ des utilisateurs sauf de l'utilisateur $i$. Ainsi à la réception du message l'utilisateur pourra recréer la clé partagée en faisant $y_i + x_i = x_0 + \dots + x_n = K$. Ce $K$ sera ensuite utilisé pour chiffrer le mail.

**Conclusion.** Ce systéme est simple et efficace mais ne permets pas la signature des éléments nécessaires à la création de la clé partagée, l'on peut donc envisager des DOS afin

qu'un utilisateur ne puisses plus lire ces messages. Cependant c'est une construction intéressante se basant sur un *key-agreement* via le *Certificateless Cryptography* et non pas sur ses possibilités de chiffrement/signature.

# Chapitre 3

# Architecture / Design du protocole

Dans ce chapitre, je vais m'intéresser à expliquer le fonctionnement de la *certificateless cryptography* et démontrer comment je l'ai utilisée afin de l'intégrer à un protocole de chiffrement de mail.

## 3.1 Architecture globale

Dans cette Figure 3.1, je présente uniquement l'architecture globale pour bien représenter les différents acteurs présents dans le protocole et ainsi avoir une vue d'ensemble pour faciliter la compréhension.

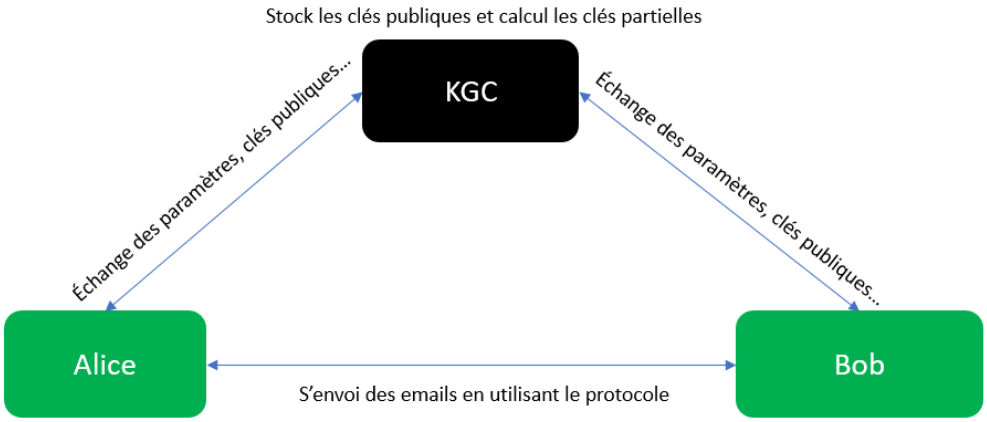


FIGURE 3.1 – Schéma global du protocole

## 3.2 Acteurs

Les parties impliquées sont les suivantes comme vu à la figure 3.1.

- Alice : L'envoyeur du mail en direction de Bob. Alice doit discuter avec le KGC pour construire sa clé privée (afin de signer) et récupérer la clé publique de Bob.
- Bob : Le destinataire du message, communique uniquement avec le KGC en ayant reçu le message d'Alice afin de récupérer sa clé publique pour vérifier la signature et de construire sa clé privé pour déchiffrer le message.
- KGC : Permet aux différents acteurs de pouvoir récupérer lesclés publiques des clients, mais aussi de recevoir les *Partial Private Keys* qui permettent aux acteurs de construire leur clé privée.

Ces partis sont les principaux présents dans un exemple de *Certificateless Cryptography* dans mon systéme de mail.

## 3.3 Fonctionnement Certificateless PKC

Je vais ici découper les différents algorithmes présent dans le certificateless public key cryptography. En passant par le chiffrement et la signature. Ces algorithmes seront accompagnés d'explications sur leur utilité. Les noms donnés aux algorithmes seront réutilisés ensuite pour les schémas afin de démontrer l'architecture du protocole mis en place. L'on peut voir des définitions spécifiques dans l'article sur lequel je me suis appuyé pour ce travail [5].

### 3.3.1 Chiffrement

Liste des différents algorithmes de *Certificateless Cryptography* et leur description, les détails techniques de leurs implémentations sont disponibles à la Section 2.7.

- *Setup.* (seulement une fois par le KGC).
- *Partial-Private-Key-Extract.* Calcul d'une clé privée partielles lorsque qu'un client le demande pour identité donnée.
- *Set-Secret-Value.* Le client ne le fait qu'une fois pour tirer sa valeur secrète.
- *Set-Private-Key.* Le client combine ses clés partielles et sa clé secrète pour obtenir une clé privée afin de déchiffrer les message reçus, chiffrés avec une certaine identité.
- *Set-Public-Key.* Le client ne le fait qu'une fois, il calcule sa clé publique en fonction de sa valeur secrète.
- *Encrypt.* Chiffre un message avec la clé publique du destinataire et son identité.
- *Decrypt.* Déchiffre un message utilisant sa clé privée et l'identité utilisée pendant le chiffrement.

### 3.3.2 Signature

Pour la signature les algorithmes sont les mêmes avec une différence dans leur conception et évidemment le *Encrypt* et *Decrypt* sont remplacé par *Sign* et *Verify*. Dans la littérature certificateless les schémas de signatures sont beaucoup plus cassés que ceux de chiffrement apparemment (voir tableau ). Il faut donc faire attention à suivre les schémas afin de vérifier que le schéma choisi ne soit pas mis à mal.

## 3.4 Design du protocole

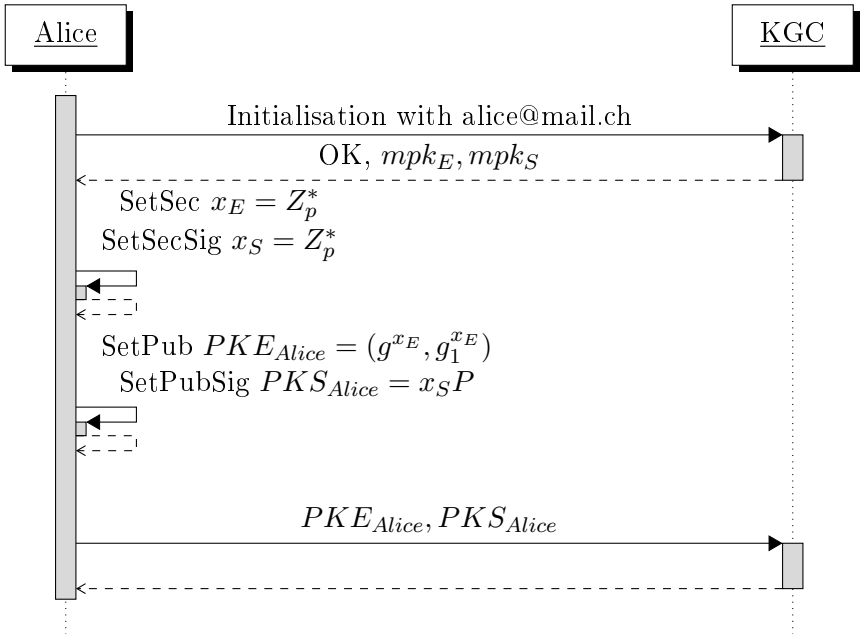


FIGURE 3.2 – Schéma de la première connexion

Dans la Figure 3.2 l'on voit la première connexion d'un utilisateur. Alice veut s'enregistrer auprès du KGC, ainsi le KGC lui renvoi les paramètres publiques (mpkS et mpkE) si aucun utilisateur n'a déjà cet email. L'utilisateur va alors crée sa valeur secrète tirée aléatoire modulo p puis générer sa clé publique. Pour finir Alice envoi sa clé publique au KGC afin qu'il l'associe à son ID et puisses le donner aux personnes qui veulent envoyer un mail à Alice.

Dans la Figure 3.3 l'on voit comment se déroulerait l'envoi d'un message à Bob :

- Tout d'abord, Alice va récupérer le clé publique de Bob via son ID (aka email).
- Elle devra aussi récupérer sa clé privée partielle de signature pour créer ses clés privées

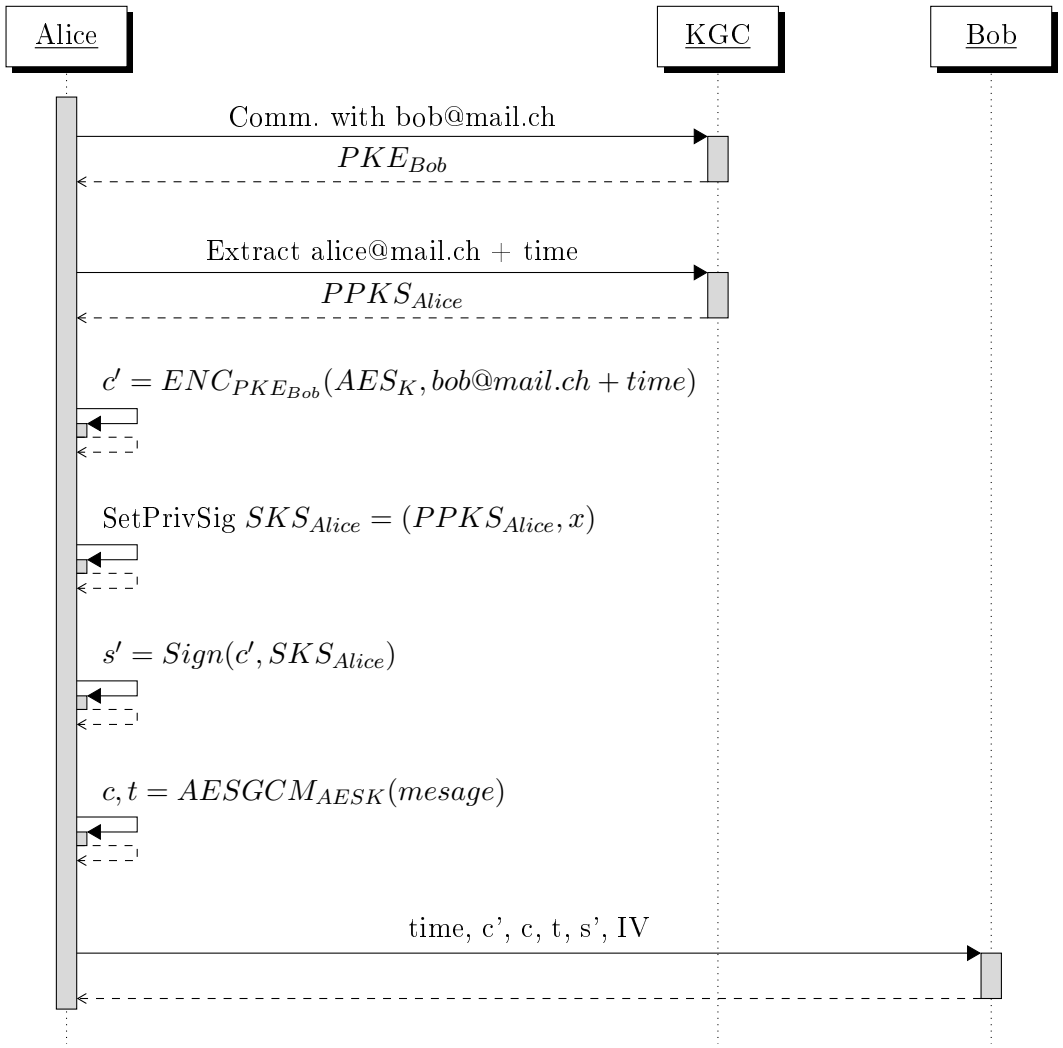


FIGURE 3.3 – Alice envoit un message à Bob

afin de signer le message. Elle va le faire à l'aide de son ID et du même timestamp qu'utilisé pour la suite.

- Elle va ensuite tirer une valeur aléatoire dans Gt qui représentera sa clé AES pour la suite, elle va chiffrer cet élément à l'aide de la clé publique de Bob et de son ID complété par un timestamp. Ce timestamp sert à garder une certaine Forward Secrecy. Le cipher sera c'.
- Elle va calculer la signature du cipher donné (s' sur la figure)
- Alice utilisera un chiffrement authentifié comme AES_GCM pour chiffrer et authentifié son mail à Bob, t pour le tag et c pour le cipher.

- Finalement elle va envoyer tout ces éléments à bob (à savoir, l'ID utilisé, c, c', t, s' et l'IV utilisé pour AES_GCM).

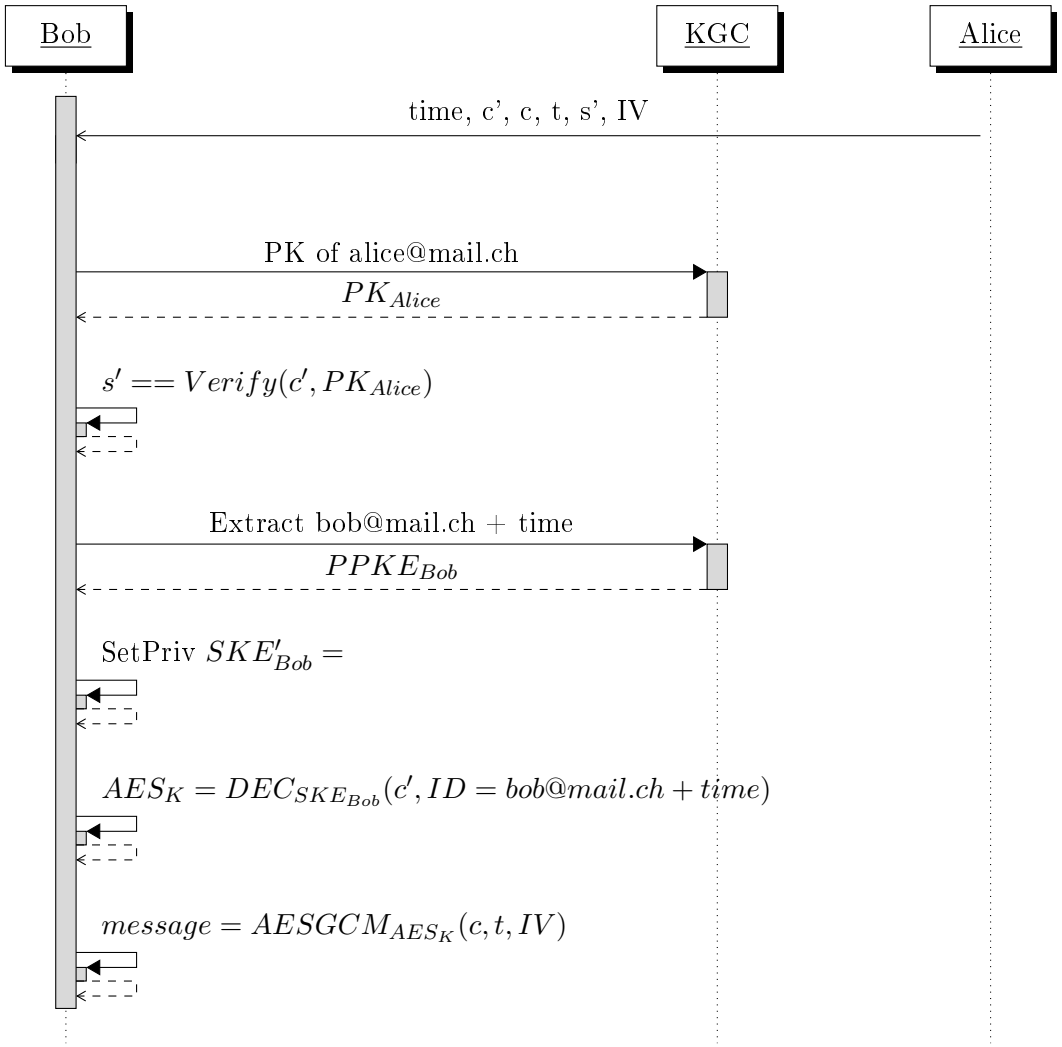


FIGURE 3.4 – Bob reçoit le message

Mais dans la Figure 3.4 l'on voit comment la réception du côté de Bob se déroulerait :

- A la réception la première chose à faire et de vérifier le cipher de la clé AES. Pour cela l'on va demander la clé publique d'Alice au KGC. Puis on va vérifier ce cipher c' à l'aide de sa signature s'.
- Ensuite Bob va récupérer sa clé privée partielle via le KGC en fournissant son ID avec le timestamp envoyé par Alice. Il va ainsi pouvoir former sa clé privée.
- Avec s clé privée il va pouvoir déchiffrer c' et obtenir la clé AES pour la suite.

- Une fois que l'on a la clé AES l'on peut simplement déchiffrer à l'aide d'AES_GCM c pour obtenir le message initial.

# Chapitre 4

# Implémentation

## 4.1 Choix d'implémentations

### 4.1.1 Langage

Au départ le choix du langage s'est porté sur sagemath (framework python) afin de mieux comprendre les différents calculs et faire un premier POC du chiffrement. Cependant l'implémentation du POC était lente et le changement d'algorithme pour les pairings était difficile. Je me suis donc orienté sur le C pour avoir de meilleures performances et pouvoir mieux gérer ma mémoire car c'est un point important dans des logiciels implémentant de la cryptographie. Pour pouvoir faire facilement des calculs sur les courbes elliptiques et les pairings en C il me fallait une librairie.

| Temps des algorithmes entres langages [s] | | |
|---|---|---|
| Algorithms | C | Sage |
| Setup | 0.2856898 | 6.5858234 |
| Encrypt | 0.0061584 | 7.6450206 |
| Decrypt | 0.00951 | 3.3274426 |

TABLE 4.1 – Table de comparaison des temps d'exécution pour les différents algorithmes de Certificateless Cryptography

### 4.1.2 Librairie cryptographique

La librairie utilisée est RELIC Toolkit [3], c'est une librairie en cours de développement qui se veut efficiente. Sa concurrence avec MIRACL m'a fait hésiter dans mon choix, mais MIRACL est plus codée en C++ avec des équivalences en C j'ai donc choisi RELIC. De plus

j'ai trouvé par exemple ici [**?**] que RELIC était généralement plus adapté dans le domaine universitaire pour des POC

### 4.1.3 Courbe utilisée

La courbe utilisée pour le POC est la BLS12-P381, en effet cette courbe est assez efficiente et compatible avec les pairings. De plus RELIC l'a dans ses options et fonctionne bien, elle a un niveau de sécurité de 128bits. Je voulais prendre une courbe avec une plus grande sécurité cependant RELIC ne l'a pas encore totalement implémenté (certains tests concernant $\mathbb{G}_2$ ne passes pas), mais la librairie étant toujours en cours de développement il faudrait suivre ça de près, le code ne changerait en effet pas.

### 4.1.4 Dérivation de la clé AES

Le but de mon schéma certificateless est de chiffrer puis signer une clé AES qui permettra à mon message d'avoir un chiffrement authentifié. Pour cela il me faut dériver un élément de $\mathbb{G}_t$ en clé AES, en effet le chiffrement dans le schéma certificateless se fait sur un élément de $\mathbb{G}_t$.
Pour cela j'ai utilisé une fonction permettant d'écrire sous forme compressée cet élément en bytes (fourni par la librairie RELIC utilisé et la fonction gt_write_bin()). Puis j'ai effectué un hachage avec SHA256 dessus, ainsi le résultat du hachage est une clé de 256 bits utilisable par AES-256-GCM. La fonction de hachage doit être par conséquent cryptographiquement sûre.

### 4.1.5 Fonctions de hachage - signature

Pour le schéma de signature il nous faut plusieurs fonctions de hachage différentes, en effet ce shéme est basé sur le *Random Oracle Model* comme définit dans le chapitre 2. Pour appliquer cela j'ai utilisé la même méthode de mapping disponible ans RELIC pour mapper une char array (tableau de byte) à un point sur G2 à savoir g2_map. Pour H1, la première fonction de hachage j'ai simplement utilisé cette fonction directement, mais pour H2 et H3 j'ai ajouté un byte devant les données à mapper respectivement les bytes '01' et '02'. Ceci afin de séparer les domaines des résultats des hashs, cela s'appelle du *Hash Domain Separation*. En effet l'on peut voir dans ce draft [7] définit comme une simulation pour prendre en compte plusieurs *Random Oracle*.

### 4.1.6 Sérialisation des données

Pour la sérialisation des données, typiquement les clés publiques et les clés privées partielles envoyées en réseau ou les clés publiques enregistrées dans les fichiers par exemple, j'ai utilisé

la librairie binn [1]. Cela permet de packer facilement des données binaires, pour cela RELIC met à disposition des méthodes g1_write_bin g1_read_bin qui a permis de faire ces enregistrements binaires. Ainsi les transferts de données sont simplifiés. Cependant il faut faire attention à certaines choses, on ne peut lire et écrire simultanément à l'aide de bin, si l'on crée un objet via un buffer on ne pourra modifier cet objet. Cela m'a posé des problèmes pour l'enregistrement des données secrétes, j'ai donc du copier l'objet lu pour pouvoir le modifier et sauver les nouveau paramétres.

### 4.1.7 Enregistrement des clés publiques (serveur)

Pour l'enregistrement j'ai utilisé une petite base de données NoSQL stockant les clés publiques des utilisateurs sur le KGC. Cela permet de facilement récupérer une clé publique pour un utilisateur si besoin. Pour implémenter cela j'ai utilisé la librairie UnQlite [2]. J'ai stocké les clés publiques pour le schéma de signature et de chiffrement séparemment, en effet, l'entrée pour la signature porte le nom "signature/ID" et le chiffrement "encryption/ID".

## 4.2 Implémentation clés de chiffrement

Pour pouvoir implémenter ce schéma de chiffrement et signature certificateless dans un système hybride il a fallu penser à une manière d'encapsuler la clé et les données. Pour cela j'ai essayé de faire un système comparable à la figure 4.1.

## 4.3 Fonctionnement global POC (KGC)

Ici je présente le fonctionnement global de mon implémentation du KGC pour mon POC. De plus je présente les problèmes connus et des propositions d'améliorations.

1. `https://github.com/liteserver/binn`
2. `https://unqlite.org/`

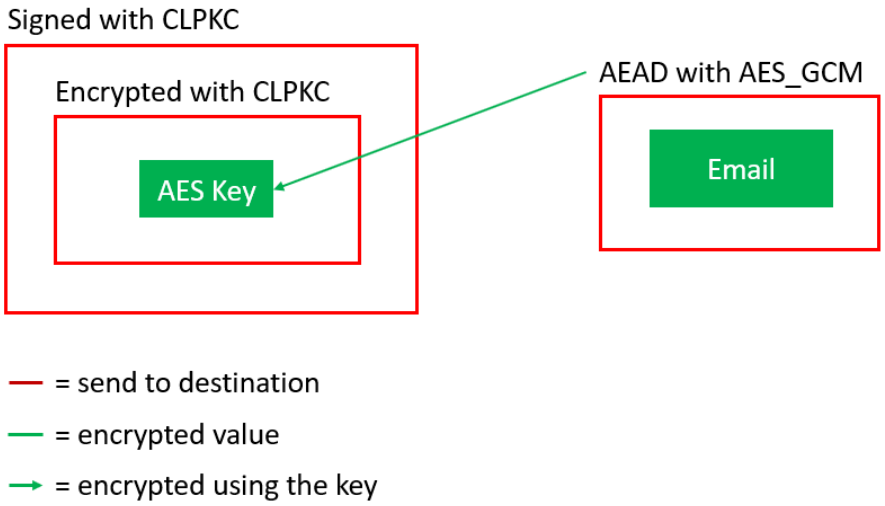


FIGURE 4.1 – Schéma encapsulation des données

### 4.3.1 Fonctionnement

### 4.3.2 Problèmes connus

### 4.3.3 Améliorations

## 4.4 Fonctionnement global POC (Client)

Ici je présente le fonctionnement global de l'implémentation du client mail pour mon POC, des améliorations possibles et des problèmes connus.

### 4.4.1 Fonctionnement

**Sécurité connexion mail.**

### 4.4.2 Problèmes connus

### 4.4.3 Améliorations

**Multiples destinataires.**

## 4.5 Comparaisons avec état de l'art

Dans cette section je vais présenter les différents protocoles et implémentations existantes présentées au chapitre 2 et les comparer à mon implémentation. Tout d'abord en présentant les différentes propriétés cryptographiques puis les temps d'exécution.

### 4.5.1 Propriétés cryptographiques

| Comparaisons des propriétés cryptographiques proposées | | |
|---|---|---|
| Implémentations | E2EE | Forward Secrecy |
| CLPKC-POC | Oui | Oui |
| PGP | Oui | Non |
| S/MIME | Oui | Non |

TABLE 4.2 – Table de comparaison des différentes propriétés cryptographiques

### 4.5.2 Temps des différentes implémentations

Comparasion du temps mis pour chiffrer et signer / déchiffrer et vérifier un mail entres les différentes implémentations existantes.

| Comparaisons des temps d'exécution entres différentes implémentations proposées | | |
|---|---|---|
| Implémentations | Chiffrement | Déchiffrement |
| CLPKC-POC | 0.0061584s | 0.00951s |
| PGP | 0 | 0 |
| S/MIME | 0 | 0 |

TABLE 4.3 – Table de comparaison des temps d'exécution entres les implémentations de mails chiffrés

### 4.5.3 Overhead induit

Ici je présente les différents *overhead* que j'ai remarqué en utilisant les différents systèmes de mails proposés.

| Comparaisons de l'overhead induit dans un mail | | |
|---|---|---|
| Implémentations | Taille overhead | Contenu |
| CLPKC-POC | Environ 1200 bytes | Signature, timestamp, nonce, Encrypted Session Key |
| PGP | Environ 300 bytes | Encrypted Session key |
| S/MIME | 0 | jE SAIS PAS |

TABLE 4.4 – Table de comparaison des différents overhead en rapport avec les solutions existantes

# Chapitre 5

# Conclusion

# Bibliographie

[1] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer, 2003.

[2] Mashael AlSabah, Alin Tomescu, Ilia A. Lebedev, Dimitrios N. Serpanos, and Srinivas Devadas. Privipk : Certificate-less and secure email communication. *Comput. Secur.*, 70 :1–15, 2017.

[3] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

[4] Suresh Kumar Balakrishnan and V. P. Jagathy Raj. Practical implementation of a secure email system using certificateless cryptography and domain name system. *I. J. Network Security*, 18(1) :99–107, 2016.

[5] Alexander W. Dent, Benoît Libert, and Kenneth G. Paterson. Certificateless encryption schemes strongly secure in the standard model. In Ronald Cramer, editor, *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 2008.

[6] Yee-Lee Er, Wei-Chuen Yau, Syh-Yuan Tan, and Bok-Min Goi. Email encryption system using certificateless public key encryption scheme. In James Jong Hyuk Park, Jongsung Kim, Deqing Zou, and Yang Sun Lee, editors, *Information Technology Convergence, Secure and Trust Computing, and Data Management - ITCS 2012 & STA 2012, Gwangju, Korea, September 6-8, 2012*, volume 180 of *Lecture Notes in Electrical Engineering*, pages 179–186. Springer, 2012.

[7] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad Wahby, and Christopher Wood. Hashing to elliptic curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-09, IETF Secretariat, June 2020. `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-hash-to-curve-09.txt`.

[8] Xinyi Huang, Willy Susilo, Yi Mu, and Futai Zhang. On the security of certificateless signature schemes from asiacrypt 2003. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and

Yongqing Li, editors, *Cryptology and Network Security, 4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005, Proceedings*, volume 3810 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2005.

[9] Nadim Kobeissi. An analysis of the protonmail cryptographic architecture. *IACR Cryptol. ePrint Arch.*, 2018 :1121, 2018.

[10] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. Technical report, 2016.

[11] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. Efail : Breaking S/-MIME and openpgp email encryption using exfiltration channels. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 549–566. USENIX Association, 2018.

[12] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[13] Hu Xiong, Zhen Qin, and Athanasios V. Vasilakos. *Introduction to Certificateless Cryptography.* CRC Press, Inc., USA, 2016.

[14] Jyh-haw Yeh, Srisarguru Sridhar, Gaby G. Dagher, Hung-Min Sun, Ning Shen, and Kathleen Dakota White. A certificateless one-way group key agreement protocol for end-to-end email encryption. In *23rd IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2018, Taipei, Taiwan, December 4-7, 2018*, pages 34–43. IEEE, 2018.

[15] Lei Zhang and Futai Zhang. A new provably secure certificateless signature scheme. In *Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008*, pages 1685–1689. IEEE, 2008.

[16] Zhenfeng Zhang, Duncan S. Wong, Jing Xu, and Dengguo Feng. Certificateless public-key signature : Security model and efficient construction. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, Singapore, June 6-9, 2006, Proceedings*, volume 3989 of *Lecture Notes in Computer Science*, pages 293–308, 2006.

# Table des figures

# Liste des tableaux

# Annexe A

# Outils utilisés pour la compilation

## A.1 RELIC Toolkit

Pour pouvoir faire des calculs de *Pairings* et sur des courbes elliptiques je me suis fier à RELIC Toolkit [3] qui est une librairie C permettant ce genre de calculs assez simplement. Cette librairie demande à être compilée avec une certaine courbe et certaines options (typiquement fonction de hachage et autres...). Des presets existent et c'est donc ce que j'ai utilisé pour ce POC. Cela demande donc de fournir la librairie précompilée avec les bonnes options pour l'utilisateur. L'inconvénient c'est donc que pour mettre à jour une courbe il va falloir recompiler toute la librairie et la fournir à l'utilisateur, néanmoins on n'aura pas à changer de code.

## A.2 Libsodium

Pour faire du chiffrement authentifié j'ai utilisé libsodium [1], en effet, m'étant un peu familiarisé avec la librairie il m'a semblé être le choix le plus évident en plus de fournir des méthodes de chiffrement simples à mettre en place. Nécessite d'avoir libsodium en librairie linkée.

1. https ://libsodium.gitbook.io/doc/

# Annexe B

# Fichiers

Je liste ici les fichiers annexes à mon rapport, ce qu'ils contiennent et comment les utiliser si besoin.

## B.1 Code du POC

Le code est en annexe du rapport avec le nom *POCCertificatelessCryptography*. Le *README.md* présent à la source devrait être suffisant pour compiler soit même le code. Le code est très commenté au niveau du *main.c* pour bien montrer les différentes étapes telles qu'elle pourraient arriver dans une implémentation finale.

## B.2 Tableaux comparatifs

Les tableaux comparatifs cité dans le chapitre 2 apparaissent sous forme de feuille dans un fichier excel se nommant *ComparatifsCLPKCSchemes.xlsx*. La feuille nommée CLEs contient un comparatif des schémas de chiffrement tandis que la feuille CLSs contient un comparatif des schémas de signatures.