THRESHOLD ENCRYPTION POUR LE PROJET KLUIS

JULIEN CATHALO

Ce document présente le concept de threshold encryption, d'abord de manière générale, puis motive et décrit la manière dont il est utilisé dans l'architecture proposée par la section Recherches de Smals dans le cadre d'un projet nommé « Kluis ». Il termine par quelques exemples d'applications existantes qui utilisent threshold encryption.

En mars 2011, Smals a déposé une demande de brevet pour ce système auprès de l'office européen des brevets.

1. Introduction

Dans le cadre du projet Kluis, la section Recherches de Smals a conçu un système permettant à une personne de stocker, écrire et consulter des données dans une base de données distante de manière hautement sécurisée. Le système permet de répartir la confiance entre à deux (ou plus) entités pour traiter les données. En chiffrant les données et en faisant appel à des entités qui ont des intérêts différents, on offre aux utilisateurs une meilleure garantie de confidentialité des données, car même si l'une de ces entités tentait d'accéder aux données, le système l'en empêcherait. Ce système peut par exemple être utilisé par des acteurs de soins de santé pour consulter et mettre à jour les dossiers médicaux de patients.

Les participants du système sont :

- Un acteur (personne qui souhaite lire ou écrire des données)
- Deux serveurs de déchiffrement partiel (SDP1 et SDP2)
- Un serveur de base de données (BDD)

En matière de sécurité, les données consultées et modifiées par l'acteur sont confidentielles ; elles ne doivent donc pas être consultables par des personnes non autorisées. Elles doivent être stockées de manière chiffrée dans la base de données. Imaginons le pire scénario envisageable concernant la confidentialité : le système est attaqué par un administrateur (de l'un des serveurs de déchiffrement partiel) mal intentionné, ayant accès au support physique des données et muni d'une clé de déchiffrement. Le système de chiffrement utilisé doit nous protéger de ce scénario en empêchant qu'un tel attaquant puisse lire les données en clair.

En matière de performances, les données doivent pouvoir être traitées rapidement, et l'acteur doit pouvoir effectuer ces opérations sur un simple PC.

Un simple système de chiffrement classique ne permet pas a priori de résoudre ce problème sans grandement compliquer l'architecture et diminuer le niveau de sécurité. Heureusement, une approche innovante innovante en matière de cryptographie permet de relever ce défi. Elle repose sur une méthode de chiffrement proposée en 1989 par les chercheurs Yvo Desmedt et Yair Frankel qui s'appelle « threshold encryption ». L'architecture que nous proposons utilise cette méthode de chiffrement pour permettre :

- que n'importe quel acteur puisse aisément chiffrer des données
- que n'importe quel acteur puisse déchiffrer les données à condition que les deux serveurs de déchiffrement collaborent au déchiffrement
- qu'aucun serveur de déchiffrement partiel ne puisse, *seul*, déchiffrer les données

En d'autres termes, cela signifie que les données et / ou l'un des serveurs de déchiffrement partiel peuvent être hébergés et administrés par une entité en laquelle l'acteur n'a pas une confiance absolue. Seule une collusion entre les différents serveurs de déchiffrement partiel pourrait compromettre les données.

En quelques mots, un système de threshold encryption est un système de chiffrement à clé publique dans lequel le déchiffrement nécessite la collaboration de plusieurs entités. Nous expliquons en plus de détails ce concept dans la suite de ce document et nous le comparons à d'autres approches.

2. Threshold Encryption

On peut voir un système de threshold encryption comme un système de chiffrement à clé publique classique dans lequel la clé privée qui sert au déchiffrement serait partagée entre plusieurs utilisateurs avec un système de partage de secret ou « secret sharing ». Nous allons donc commencer par expliquer ce qu'est un système de secret sharing et ce qu'est un système de chiffrement à clé publique.

2.1. Secret Sharing

Prenons l'exemple d'une banque munie d'un coffre qui s'ouvre avec un mot de passe. Seul le directeur de la banque connaît le mot de passe. Il a 4 employés et il voudrait qu'en son absence ils puissent en cas de nécessité accéder au coffre, mais sous une condition : il faut qu'au moins deux employés soient présents (pour éviter qu'un seul employé malhonnête puisse voler le contenu du coffre). Le directeur cherche donc un moyen de « partager » son mot de passe en 4 morceaux (il en donnera un à chaque employé) de manière à ce que n'importe quel groupe de deux employés puisse réunir ses deux morceaux pour reconstituer le mot de passe.

Une première idée serait de distribuer certaines lettres du mot de passe à ses employés. Par exemple, si le mot de passe est « azertyui », on donne « azerty__ » au premier employé, « azer__ui » au second, « az__tyui », « __ertyui » au dernier. Cette solution peut sembler fonctionner à première vue, car effectivement deux employés peuvent reconstituer le mot de passe : par exemple, avec « azerty__ » et « az__tyui » on retrouve « azertyui ». Mais le souci est que l'on révèle trop d'information sur le mot de passe à chaque employé. En effet, un employé seul peut tester toutes les combinaisons de deux lettres à partir de son share pour retrouver le secret. C'est donc une mauvaise idée d'utiliser un tel système, il n'offre pas un bon niveau de sécurité.

La solution au problème du directeur s'appelle le partage de secret.

Un système de partage de secret permet de créer, à partir d'une valeur secrète (comme un mot de passe par exemple) des morceaux ou « shares » de secret de telle manière qu'un nombre fixé de shares permet de reconstituer le secret.

Plus précisément, un tel système a deux paramètres :

- n, le nombre total de shares
- t, le seuil, nombre minimal de shares permettant de reconstituer le secret

A partir du secret S, n shares sont générés et distribués à n utilisateurs. Quand t utilisateurs mettent en commun leurs shares, ils peuvent reconstituer le secret S. Mais si strictement moins de t utilisateurs sont présents, alors ils ne peuvent pas le reconstituer. En fait cela va plus loin : si t-1 utilisateurs mettent leur shares en commun, ils ne peuvent pas deviner la moindre information sur le secret (et on évite donc le problème de l'exemple du directeur qui distribuait certaines lettres de son mot de passe).

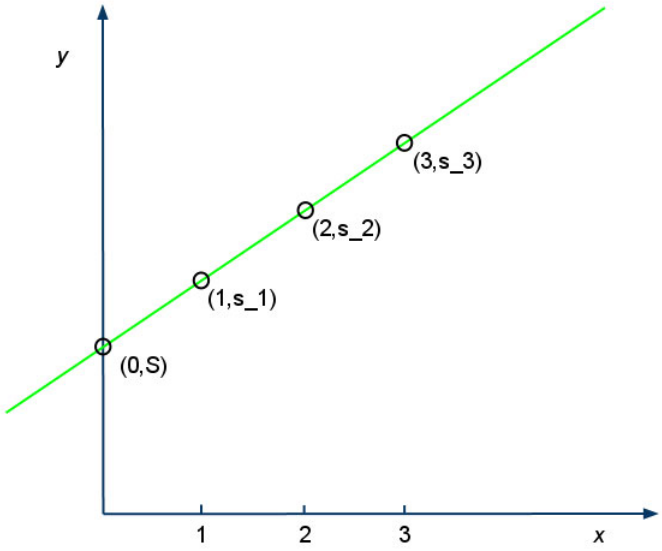
On peut aussi attribuer des poids aux utilisateurs en donnant à certains utilisateurs plusieurs shares.

Il existe de nombreux système de partage de secret, citons les deux premiers qui ont été inventés (tous les deux en 1979, indépendamment l'un de l'autre): le schéma de Shamir et celui de Blakley. Nous expliquons en détail le schéma de Shamir ci-après car c'est l'un des systèmes les plus simples à comprendre.

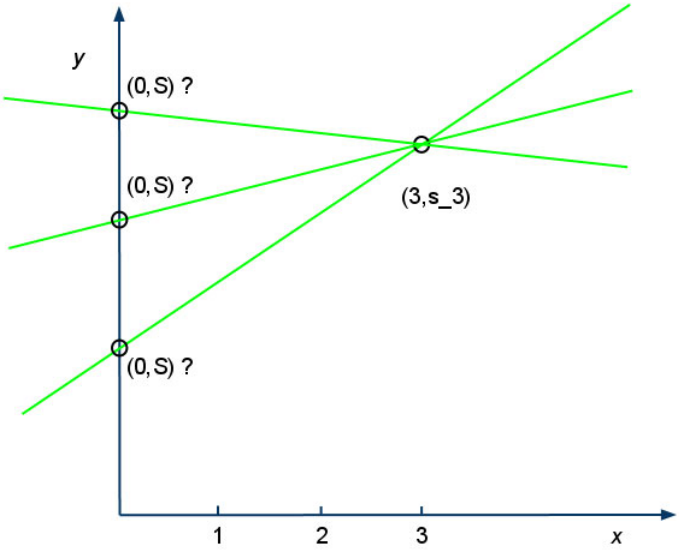
Ce schéma est basé utilise des polynômes. Commençons par expliquer le cas simple où t = 2. Le nombre d'utilisateurs n peut prendre n'importe quelle valeur plus grande ou égale à 2. On va utiliser le fait que :

- par 2 points du plan passe une seule droite
- par 1 point du plan passent une infinité de droites

L'entité chargée de générer les shares (dans notre exemple, le directeur), va donc créer une droite (d'équation y=ax+S, où a est tiré au hasard et S est le secret) et donner à chaque utilisateur les coordonnées d'un point de la droite, ces coordonnées seront le share de l'utilisateur. Bien sûr, on ne donne à personne les coordonnées du point (0,S) puisque qu'il révèle le secret S ! Et il faut aussi que tous les utilisateurs reçoivent des points différents.

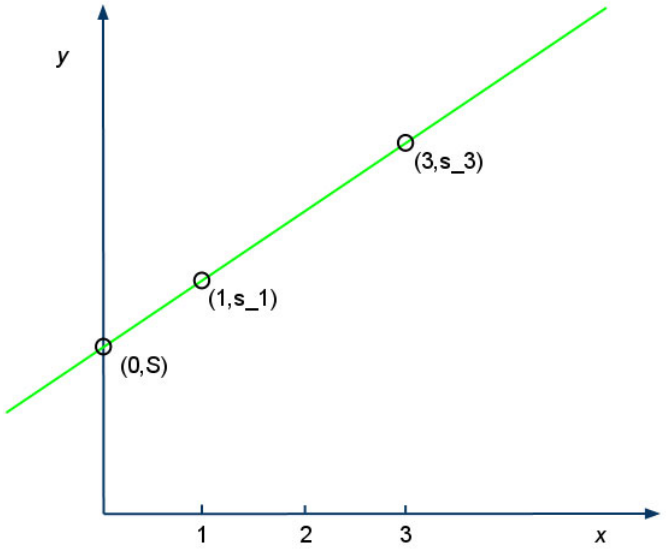


Dans cet exemple, on donne à l'utilisateur 1 les coordonnées (1,s_1), à l'utilisateur 2 les coordonnées (2_s_2) et à l'utilisateur 3 les coordonnées (3,s_3). Chaque utilisateur est incapable de déterminer la droite tout seul. Par exemple, le point connu par l'utilisateur 3 correspond à une infinité de droites possibles, donc à une infinité de valeurs de S possibles :



Mais dès que deux utilisateurs (n'importe lesquels) mettent leurs points en commun, ils peuvent avec un peu de calcul trouver la seule droite qui passe par

ces deux points et donc la valeur du secret S. Par exemple, si les utilisateurs 1 et 3 mettent leur points en commun ils peuvent retracer la droite :



C'est donc comme cela que fonctionne le système avec un seuil t = 2.

Quand t > 2, on utilise des polynômes de degré t-1 mais c'est le même principe qui est appliqué : on génère un polynôme de degré t-1 et chaque share est la valeur de ce polynôme en un point différent. Si t utilisateurs mettent leurs secrets en commun, ils retrouvent le polynôme (via une interpolation de Lagrange), mais si t-1 ou moins seulement le font, ils n'obtiendront pas la moindre information sur le secret.

D'autres idées peuvent être utilisées ; par exemple, le schéma de Blakley utilise des intersections de plans.

Remarque : il ne faut pas confondre secret sharing et un autre concept de cryptographie, le « key exchange » (aussi appelé échange de clés). Un système de secret sharing permet à une personne de prendre une donnée secrète et de la partager en plusieurs shares. Un système de key exchange (dont le plus connu est celui de Diffie-Hellman) permet à deux personnes d'établir ensemble une donnée secrète.

2.2. Public Key Encryption

Un système de chiffrement à clé publique classique ou chiffrement asymétrique permet à Alice de chiffrer un message à destination de Bob simplement en utilisant la clé publique de Bob (donnée publique connue de tous) ; Bob pourra déchiffrer le message en utilisant la clé privée correspondante (donnée secrète connue seulement par Bob).

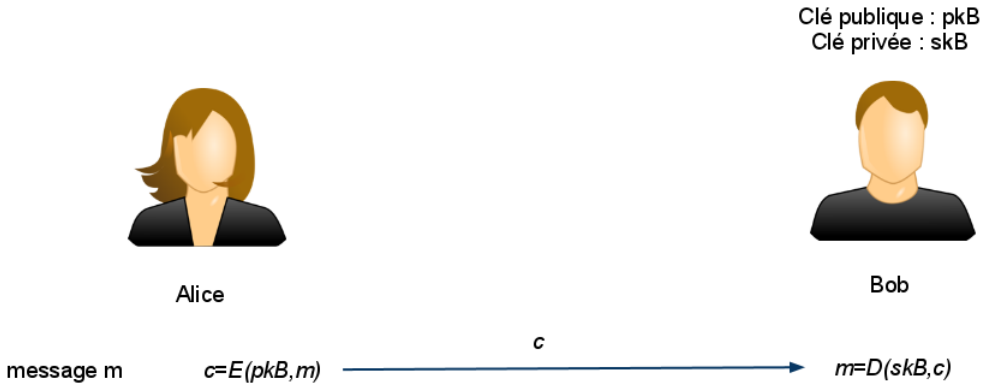


Figure 1

Sur cette figure, Alice chiffre le message m avec la clé publique de Bob pkB ; elle obtient un message chiffré c qu'elle envoie à Bob. Bob déchiffre c en utilisant sa clé privée skB.

2.3. Secret Sharing + Public Key Encryption = Threshold Encryption

Un système de threshold encryption (concept proposé en 1989 par Yvo Desmedt et Yair Frankel) est en quelque sorte la rencontre entre un système de chiffrement à clé publique classique et un système de partage de secret.

Avec un système de threshold encryption, il y a toujours une seule clé de chiffrement ; elle correspond à un groupe d'utilisateurs et chaque utilisateur possède sa propre clé de déchiffrement. On définit au départ deux paramètres t et n comme dans un système de partage de secret : n correspond au nombre total d'utilisateurs et t au seuil.

L'idée est qu'au moins t utilisateurs doivent collaborer pour déchiffrer un message adressé au groupe. N'importe quel ensemble de t utilisateurs peuvent se réunir pour déchiffrer un message ; mais si moins d'utilisateurs collaborent, ils seront incapables de retrouver le message.

Dans ce système les paramètres sont les suivants :

- Le nombre total d'utilisateurs : n
- Le seuil : t
- Une clé publique : pk
- Une clé privée par utilisateur : sk_1, sk_2, …, sk_n

N'importe qui peut chiffrer un message m en utilisant la clé publique pk :

c = E(pk,m)

Un utilisateur i peut calculer un « share de déchiffrement » en utilisant le message chiffré c et sa propre clé privée sk_i

s_i = ShareDecrypt(sk_i,c)

On peut combiner t shares des déchiffrement s_j1, …, s_jt pour reconstituer le message m :

m=Combine(pk,c,s_j1,…,s_jt)

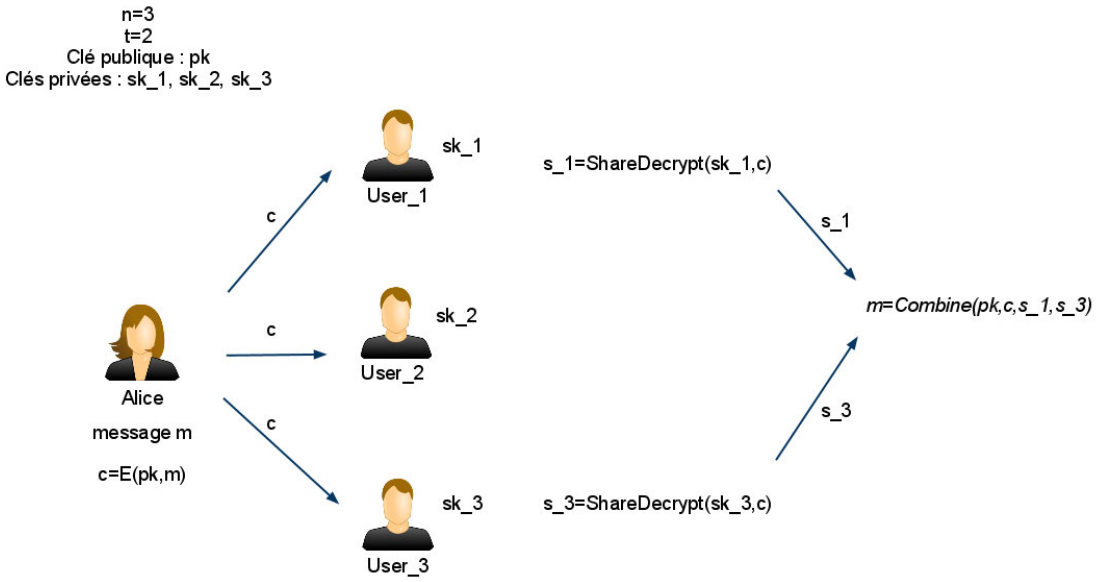


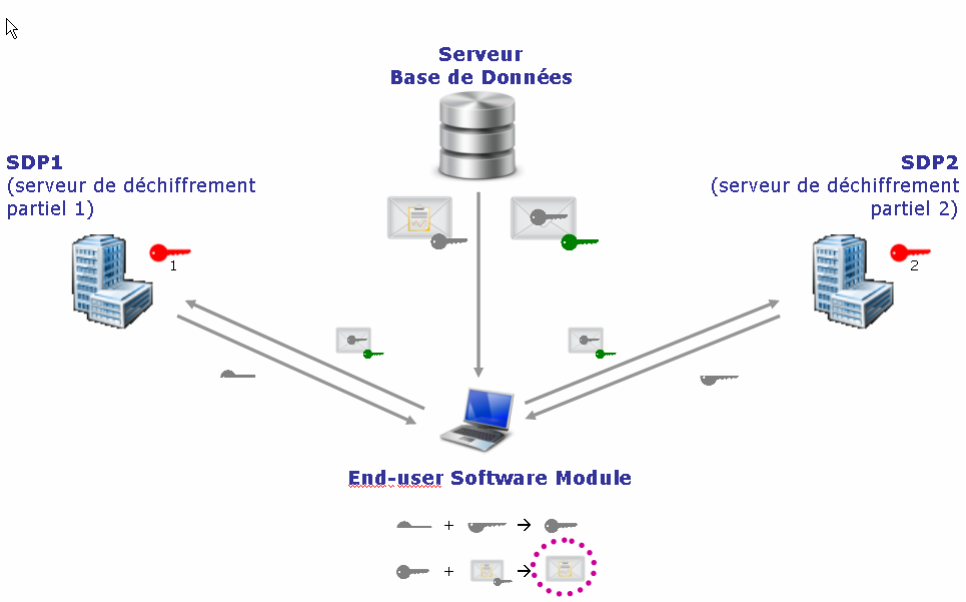
Figure 2

Sur cet exemple avec 3 utilisateurs et un seuil de 2, Alice chiffre un message m avec la clé publique pk et envoie le message chiffré c aux utilisateurs. Les utilisateurs 1 et 3 calculent des shares de déchiffrements, respectivement s_1 et s_3 (ici l'utilisateur 2 ne collabore pas). Avec les deux shares s_1 et s_3 on peut reconstituer le message clair. Avec uniquement s_1 ou uniquement s_3, il est impossible de retrouver la moindre information sur le message m. Ici ce sont les utilisateurs 1 et 3 qui interviennent mais toute combinaison d'au moins deux utilisateurs qui collaborent permet de retrouver le message.

3. Threshold Encryption dans le cadre du Kluis

3.1. Fonctionnement

Dans notre exemple, nous utilisons une version simple de threshold encryption où t = n = 2. Mais on peut tout à fait ajouter des serveurs de déchiffrement partiel (donc augmenter n) et augmenter le seuil (donc augmenter t, à condition que t soit inférieur ou égal à n). On a donc une clé publique pk à laquelle correspondent deux clés privées sk_1 et sk_2 ; le serveurs de déchiffrement partiel 1 (SDP1) possède sk_1 et le SDP 2 possède sk_2. Le serveur de base de données BDD stocke les donnéees chiffrées.



3.2. Avantages et comparaison avec d'autres approches

La solution permet :

- que n'importe quel acteur puisse aisément chiffrer des données (il utilise pour cela la clé publique pk)
- que n'importe quel acteur puisse déchiffrer les données à condition que chaque serveur de déchiffrement partiel collabore au déchiffrement
- qu'aucun serveur de déchiffrement partiel ne puisse déchiffrer *seul* les données (ceci est garanti par les propriétés du schéma de threshold encryption utilisé)

Il est très important de remarquer que les données elles-mêmes ne sont jamais envoyées aux SDP1 et SDP2. **Les données ne leur sont accessibles à aucun moment**, même pas sous forme chiffrée. Ces serveurs ne font que participer au déchiffrement de la clé qui protège les données.

De plus, l'acteur n'a pas à stocker de secret : il ne déchiffre le message que grâce aux shares de déchiffrement envoyés par les SDPs. Ceci est désirable sur le plan de la sécurité puisque le client utilisera a priori un simple PC.

On peut comparer cette solution avec d'autres approches possibles :

Système entièrement à clé symétrique :

On pourrait imaginer un système utilisant uniquement des clés symétriques. L'acteur chiffrerait les données en utilisant une clé symétrique de session pour chaque champ ; ensuite, cette clé symétrique serait stockée d'une manière partagée (dans le sens « partage de secret ») par les SDPs. Cette solution demanderait un stockage de clé au niveau des SDPs et compliquerait grandement les interactions, notamment lors du chiffrement.

Système à chiffrement asymétrique, clé privée confiée à un serveur :

On pourrait aussi imaginer la solution suivante : les données sont simplement chiffrées l'acteur avec la clé publique d'un serveur et sont ensuite stockées dans la base de données. Quand l'acteur veut lire une donnée, la base de données lui envoie la donnée chiffrée et l'acteur la transmet au serveur qui la déchiffre. Cette solution est simple et performante mais offre une trop grande

vulnérabilité : si la base de données tombe dans les mains d'un administrateur mal intentionné du serveur, il peut lire toutes les données.

On voit donc que la solution proposée avec threshold encryption offre de nombreux avantages par rapport à des alternatives.

4. Real stories : cas réels d'utilisation de threshold encryption et secret sharing

Comme déjà expliqué, un algorithme de threshold encryption se construit sur l'idée de partager la clé privée de déchiffrement avec un algorithme de secret sharing. Ces concepts datent de la fin des années 70 et ont depuis été le sujet de centaines de publications scientifiques. Il existe plusieurs produits disponibles sur le marché qui utilisent ces concepts pour protéger des données et des clés. Dans certains cas, c'est une clé symétrique de chiffrement / déchiffrement qui est partagée via un système de secret sharing. La liste suivante présente quelques cas réels d'utilisation de secret sharing et de threshold encryption ; elle constitue un exemple et n'est pas exhaustive.

4.1. Utilisation de secret sharing dans les HSM

Un HSM (Hardware Security Module) est un cryptoprocesseur dédié à l'exécution d'opérations de cryptographie. Plus précisément un HSM a trois fonctions principales :

- Stocker des clés de manière sécurisée, offrant une résistance à des attaques software et hardware
- Effectuer des opérations de cryptographie (chiffrement, déchiffrement, signature, etc.) de manière performante et donc de libérer le serveur d'application de ces tâches qui (dans le cas asymétrique) demanderaient des temps de calcul trop importants
- Contrôler l'accès aux clés

4.1.1. Sophos Utimaco[1]

La société Utimaco (propriété de Sophos depuis 2008) propose une gamme de HSMs nommés SafeGuard CryptoServer. Ces produits utilisent le système de secret sharing de Shamir (basé sur des polynômes et présenté plus haut) pour protéger le backup de la master key. Ainsi, la master key n'est pas sauvegardée sur un support unique mais répartie sur n supports de backup gardés par n administrateurs; il faut en réunir t pour reconstituer la clé. Ce système permet donc d'obtenir un bon compromis entre :

- La nécessité d'éviter de perdre les données : la master key ne doit être perdue en aucun cas (si c'est une clé de déchiffrement, les données seraient alors perdues)
- La sécurité : aucun administrateur ne peut retrouver seul la master key (et même une coalition de t-1 administrateurs ne peut la retrouver). La confidentialité des données est donc assurée.

[1] http://hsm.utimaco.com/products/hsm-series/

4.1.2. SafeNet[2]

Le secret sharing est aussi utilisé dans certains HSMs de la marque SafeNet comme par exemple le produit Luna CA4. Ce HSM est dédié aux PKI et plus précisément au rôle de CA (Certification Authority). Dans une PKI, le CA doit émettre des certificats et donc générer des signatures à l'aide d'une clé privée (appelée « PKI root key »). Cette clé est particulièrement sensible dans le sens où sa compromission impliquerait l'insécurité de l'ensemble de la PKI : un attaquant en possession de cette clé pourrait générer de faux certificats.

Le Luna CA4 chiffre cette clé avec l'algorithme 3DES et la clé 3DES est répartie entre plusieurs administrateurs via un système de secret sharing. Cette protection est donc semblable avec celle d'Utimaco et offre les mêmes avantages (compromis entre la nécessité de ne pas perdre la root key et la sécurité).

4.2. Utilisation de secret sharing dans un logiciel PKI Open Source

Le logiciel de PKI open source OpenXPKI[3] utilise le secret sharing pour la même application que le produit SafeNet Luna CA4. Les clés privées du CA root et des CAs intermédiaires sont protégées via le système de secret sharing de Shamir.

4.3. Utilisation de secret sharing pour un système d'enchères au Danemark

Le secret sharing est aussi un composant de base du système de "secure multi-party computation" testé à grande échelle au Danemark pour un système complexe d'enchères[4].

Au Danemark, une seule entreprise achète des betteraves à sucre aux producteurs. Les producteurs ont des contrats de livraison de betteraves qui les lient à cette entreprise et sont échangeables. Pour automatiser le processus d'échange de ces contrats, une solution était nécessaire pour déterminer un prix juste d'échange en fonction de l'offre et de la demande. Plus précisément, il s'agit du « Market Clearing Price », prix pour lequel il y a autant d'offres de vente de contrats que de demandes d'achat de contrat. Mais ceci posait des problèmes de confidentialité (car le prix auquel un fermier consent à vendre ses contrats révèle des informations confidentielles sur sa situation économique et sa productivité). En janvier 2008, une équipe constituée de chercheurs basés au Danemark et aux Pays-Bas a mis en place un système pour traiter ce problème. Le système est basé sur un protocole de secure multi-party computation. Les différents participants placent leurs offres et leurs demandes de contrat via un serveur web ; le serveur retourne ensuite le prix juste. Lors de l'expérience, 25 000 tonnes de betteraves ont ainsi changé de propriétaire.

---

[2] http://www.safenet-inc.com/products/data-protection/hardware-security-modules-hsms/

[3] http://www.openxpki.org/

[4] « Secure Multiparty Computation Goes Live » : http://eprint.iacr.org/2008/068.pdf

4.4. Utilisation de threshold encryption pour le vote électronique

Le principe de threshold encryption est utilisé dans des systèmes de vote électronique pour assurer la confidentialité des votes. Dans un système de vote électronique, chaque bulletin de vote est chiffré avec une clé publique. Plusieurs tiers de confiance (aussi appelés « trustees ») interviennent pour déchiffrer, mais aucun ne doit pouvoir obtenir le contenu du bulletin de vote. Ce principe a été implémenté dans le système de vote « Helios Voting »[5], développé par Ben Adida, professeur à Harvard. Dans ce système, c'est l'algorithme « Threshold ElGamal Encryption » qui est implémenté pour le chiffrement des bulletins de vote. Le système utilise t=3 et n=3 avec 6 trustees. Trois clés de déchiffrement sont donc utilisées, et chaque clé est connue par 2 trustees différents.

Ce système a déjà été utilisé dans plusieurs élections et notamment en Belgique pour élire le recteur de l'Université catholique de Louvain en 2009[6].

4.5. Autres exemples

Le secret sharing est aussi utilisé :

- Dans les produits PKI de Verisign
- Dans le produit Keon Desktop de RSA Security
- Des produits Atos Wordline

Enfin, plusieurs librairies implémentant le secret sharing, par exemple :

- Shamir Secret Sharing in Java

  http://sourceforge.net/projects/secretsharejava/
- ssss

  http://freshmeat.net/projects/ssss/

[5] http://heliosvoting.org

[6] http://www.uclouvain.be/270428.html