Evaluation de la confiance dans les architectures de sécurité

Jean-Baptiste Orfila

▶ **To cite this version:**

Jean-Baptiste Orfila. Evaluation de la confiance dans les architectures de sécurité. Cryptographie et sécurité [cs.CR]. Université Grenoble Alpes, 2018. Français. NNT : 2018GREAM034 . tel-01985183

HAL Id: tel-01985183
https://theses.hal.science/tel-01985183v1

Submitted on 17 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Jean-Baptiste ORFILA

Thèse dirigée par **Jean-Guillaume DUMAS**

préparée au sein **Laboratoire Jean Kuntzmann**
et de **École Doctorale de Mathématiques, Sciences et Technologies de l'Infor-mation, Informatique**

Évaluation de la confiance dans les architectures de sécurité

Thèse soutenue publiquement le **3 juillet 2018**,
devant le jury composé de :

Madame, Marie-Laure Potet
Professeur, Université Grenoble-Alpes, Présidente
Monsieur, Franck Leprevost
Professeur, Université du Luxembourg, Rapporteur
Madame, Marine Minier
Professeur, Université de Lorraine, Rapporteur
Monsieur, Pascal Lafourcade
Maître de Conférences, Université Clermont-Auvergne, Examinateur
Monsieur, Patrick Redon
Ingénieur, Thales Gennevilliers, Examinateur
Monsieur, Jean-Guillaume Dumas
Professeur, Université Grenoble-Alpes, Directeur de thèse

REMERCIEMENTS

Je remercie particulièrement Jean-Guillaume Dumas, pour m'avoir fait découvrir le monde de la recherche, en acceptant d'être mon directeur de thèse. Sa passion, son encadrement et son implication m'ont beaucoup appris, notamment dans les domaines scientifiques, mais aussi plus largement : il m'a poussé à donner le meilleur de moi-même tout au long de ces quatre années.

Je remercie Marie-Laure Potet, pour avoir accepté de présider le jury de cette thèse, mais aussi pour son enthousiasme durant nos collaborations. Un grand merci à Franck Leprévost et Marine Minier, tous deux rapporteurs de cette thèse, pour les diverses corrections et interrogations qu'ils ont mises en exergue. Je remercie également Pascal Lafourcade et Patrick Redon, pour leurs questions et réflexions pertinentes lors de la soutenance. Chacun des membres du jury a grandement participé à la réussite de cette soutenance, qui me restera comme un excellent souvenir.

Je remercie toutes les personnes avec lesquelles j'ai eu la chance de collaborer, notamment Pascal Lafourcade et Pascal Thoniel, pour la confiance qu'ils m'ont accordée, et tous les échanges enrichissants que nous avons eus. Je remercie aussi les membres du consortium ARAMIS, à l'origine de cette thèse, et qui en regroupant des profils industriels et académiques, m'ont transmis leur point de vue complémentaire sur les problématiques abordées dans ce manuscrit. Une pensée particulière pour les autres thésards du projet, et notamment à Maxime, qui fut l'équivalent de mon binôme de thèse.

Je remercie les membres de l'équipe CASYS, ainsi que les équipes administrative et informatique, pour leur bonne humeur et leur disponibilité.

Un immense merci aux thésards du LJK, et autres personnes qui m'ont entouré au cours de ces dernières années. Leur soutien au quotidien, et tous les moments que nous avons partagés, ont très largement contribué à la réussite de mon expérience en tant que doctorant.

Enfin, je remercie mes parents, qui m'ont soutenu et encouragé pendant toutes ces années. Chers parents, je crois que j'ai enfin fini mes études.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE I

INTRODUCTION ET ORGANISATION DE LA THÈSE

Si l'avènement de la cryptographie asymétrique permit de résoudre le problème de l'échange des clefs secrètes, il ouvrit aussi de nombreux axes de recherche. En particulier, les problématiques liées à la gestion des clefs publiques, comprenant entre autres leur diffusion, et leur association à une identité sont devenues cruciales pour permettre le déploiement de ce nouveau paradigme dans les systèmes d'information.

Ces derniers n'ont cessé d'évoluer, avec d'importantes avancées en capacité de calcul ou en moyens de communication. Pourtant, hormis pour certains domaines, leur sécurité n'a pas été mise au coeur de leur conception. Typiquement, les architectures de contrôle et de supervision déployées dans des domaines comme l'énergie ou le médical prônent majoritairement la fiabilité ou la sûreté de l'installation, souvent au détriment de sa sécurité. Récemment, de nombreuses attaques, envers des centrales nucléaires (via *Stuxnet* [FMC11] ou encore *BlackEnergy* [LC16]), ou des véhicules connectés (avec une prise de contrôle partielle de ces derniers [MV14]), ont été recensées. Elles démontrent clairement qu'en entraînant des dysfonctionnements au sein d'une architecture, des lacunes de sécurité sont aussi capables de dégrader la sûreté de ces systèmes.

Au-delà des infrastructures industrielles, les utilisateurs sont aussi directement impactés par l'évolution des systèmes d'information : la consultation d'un compte bancaire, le stockage de données en ligne ou encore l'utilisation d'objets connectés font désormais partie du quotidien. De nombreuses normes ont été mises en place dans le but de sécuriser les communications entre l'utilisateur et le serveur distant. Le meilleur exemple est probablement le protocole TLS [Eas11], dont une nouvelle version est en cours d'étude [KW16]. Durant la première phase de ce protocole, l'utilisateur (ou plutôt son navigateur) authentifie le site, en s'assurant que la correspondance entre sa clef publique et son adresse est correcte. Concrètement, cette étape correspond à la vérification du certificat du site. Récemment, des solutions comme *Let's Encrypt* [Aas14] ont permis aux serveurs d'obtenir plus facilement un certificat, et donc aux utilisateurs de les authentifier. A l'inverse, l'authentification des utilisateurs repose encore majoritairement sur l'emploi du couple identifiant/mot de passe, car l'obtention de certificats pour les utilisateurs finaux reste malaisée. Or, cette solution souffre clairement de faiblesses intrinsèques à son mode de fonctionnement : la mémorisation du mot de passe, ou encore le respect d'une politique de formatage parfois incommode ne font que favoriser les comportements à risque de la part des usagers.

Du côté des serveurs, la simplification de l'étape de certification n'a pas modifié le fonctionnement général des infrastructures de gestion de clefs : la vérification d'un certificat exige que l'utilisateur fasse appel à un tiers de confiance, dénommé autorité de certification. Cette dernière est en charge d'apposer sa signature sur le certificat, comme gage du bon contrôle des informations qu'il contient. En supposant que ce certificat soit délivré à un serveur web, l'autorité de certification garantit que

la clef publique présente sur le certificat appartient effectivement à son détenteur, dont l'identité est aussi explicitée. Lors de sa connexion, l'utilisateur récupère donc ce certificat, et s'assure de sa validité en inspectant la signature du tiers de confiance. Cette étape est généralement transparente pour l'utilisateur, car elle est effectuée par le navigateur. Elle s'apparente à un contrôle d'appartenance de l'autorité de certification dans un dépôt, dénommé magasin d'ancres de confiance. L'utilisateur considère donc le tiers signataire comme de confiance si et seulement si il appartient à ce magasin. Généralement, la gestion de ces dépôt est aussi réalisée par le navigateur, et dépend donc de ses mises à jour. Le modèle de confiance obtenu est finalement binaire, avec une confiance nulle ou absolue dans une autorité, sans forcément de corrélation avec son comportement. Typiquement, une autorité de certification délivrant des certificats à des sites frauduleux sera toujours présente dans le magasin de l'utilisateur jusqu'à l'actualisation de son navigateur.

Dans ce contexte, nous défendons une thèse dans le domaine des architectures de sécurité, et plus précisément dans la gestion des clefs asymétriques, structurée autour des trois problématiques précédemment exposées :

1. La définition d'un module de rupture protocolaire visant à améliorer la sécurité au sein de systèmes de contrôle-commandes ;
2. Une infrastructure de gestion de clefs alternative, formellement prouvée et centrée sur l'utilisateur ;
3. Un mécanisme distribué et sécurisé permettant d'obtenir une évaluation de la confiance que l'utilisateur peut avoir dans les ancres de confiance.

Le chapitre II est dédié à l'introduction des notions de cryptographie à clef publique, et des infrastructures de gestion de clefs usuelles. Nous rappelons également les principes du calcul multi-parties sécurisé, et détaillons le fonctionnement des assistants de preuves automatiques.

Dans le chapitre III, nous présentons l'architecture d'un module de sécurité dédié à l'amélioration de la sécurité des communications réseaux. L'objectif principal de ce dernier est d'effectuer de la rupture de protocoles de manière efficace et sûre. Grâce à cette méthode, l'analyse du contenu des flux réseaux est facilitée, et la diffusion des attaques est limitée. Au sein de l'architecture présentée, nous nous intéressons plus spécifiquement à la gestion des clefs. Cette dernière s'avère délicate, puisqu'un module doit s'intégrer facilement dans une infrastructure existante et doit être capable d'analyser les flux chiffrés échangés.

Dans le chapitre IV, nous exposons une infrastructure de gestion de clefs centrée sur l'utilisateur et formellement prouvée, dénommée LocalPKI. Son objectif est de démocratiser l'attribution et l'usage des certificats pour les utilisateurs. Le leitmotiv de cette nouvelle infrastructure est de fournir des garanties de sécurité équivalentes au système actuel, mais dont la mise en place est facilitée pour les utilisateurs finaux. L'idée principale est de combiner l'emploi d'un tiers de confiance

dans un système où tous les certificats sont uniquement signés par leur détenteur. Enfin, en adéquation avec les propriétés de sécurité définies pour les infrastructure de clefs, nous montrons que LocalPKI est sûre à l'aide d'un outil de vérification automatique.

Enfin, le V^e chapitre est dédié à la proposition d'un nouveau système d'évaluation privée de la confiance entre les autorités de certification. Pour cela, nous utilisons un modèle d'agrégation de confiance dans un graphe reposant sur des calculs matriciels non conventionnels. Nous développons un ensemble de protocoles permettant de calculer ces produits de manière sécurisée. L'idée est de faire participer toutes les autorités dans l'évaluation de la confiance de ces pairs, tout en préservant le secret des données qu'elles fournissent. A la fin de l'exécution du protocole, chacune des autorités obtient une estimation de la confiance qu'elle peut avoir dans les autres, en ayant pris en compte l'avis de ses collègues : ces derniers ne révèlent jamais publiquement leur opinion, et il n'est pas possible de la retrouver à partir du résultat.

CHAPITRE II

CRYPTOGRAPHIE ASYMÉTRIQUE ET INFRASTRUCTURES DE GESTION DE CLEFS

Sommaire

1 ■ NOTATIONS ET PRÉLIMINAIRES

Nous dénotons par $\mathbb{N}$ (respectivement $\mathbb{Z}$) l'ensemble des entiers naturels (respectivement relatifs). Généralement, nous dénotons par $[a,b]$ l'intervalle des entiers compris entre a et b. La notation $\{0,1\}^*$ désigne l'ensemble de toutes les chaînes de bit, c.-à-d. n'importe quelle chaîne formée d'un nombre quelconque (*) de bit. La taille d'une chaîne de bit a est dénotée par $|a|$. Les notations usuelles d'arithmétique modulaire sont employées : $a = b \mod N$ signifie que a est congru à b modulo N. Nous dénotons par $\mathbb{Z}/N\mathbb{Z}$ l'anneau (ou le corps si N est premier) contenant l'ensemble des nombres $[0,N-1]$, muni de l'addition et de la multiplication modulaire. La probabilité qu'une variable aléatoire X vaille x est exprimée sous la forme $Pr[X = x]$. Enfin, on dira qu'une fonction $\mu : \mathbb{N} \to \mathbb{R}$ est **négligeable** si pour tout polynôme positif $p(\cdot)$, il existe $N \in \mathbb{N}$ tel que $\forall n > N, |\mu(n)| \leq \frac{1}{p(n)}$.

Concernant les notations cryptgraphiques, nous dénotons par Pk_A (resp. Sk_A) la clef publique (resp. privée) d'un utilisateur A. De manière générale, la notation O_A est utilisée pour exprimer l'appartenance d'un objet O à une entité A. L'écriture $\{m\}_{Pk_A}$ (resp. $\{m\}_{Sk_A}$) représente l'action de chiffrer (resp. signer) un message m avec la clef publique Pk_A (resp. la clef privée Sk_A). Le hachage d'un message m est dénoté par $H(m)$, où H est la fonction de hachage. La concaténation de deux messages m_1 et m_2 est écrite $m_1||m_2$. Enfin, la fonction $X_{509}()$ prend les informations de l'utilisateur en entrée et les retourne dans un certificat au format X_{509} dépourvu de sa signature.

2 ■ CRYPTOGRAPHIE ASYMÉTRIQUE

La crypotologie est, étymologiquement, la science du secret. Elle s'intéresse à l'ensemble des mécanismes permettant d'assurer des propriétés en lien avec le secret de l'information. Parmi elles, les plus connues sont probablement :

- **La confidentialité** des données, qui garantie que seules les personnes autorisées ont accès aux informations ;
- **L'authentification** des interlocuteurs, qui assure que l'identité proclamée par un participant est correcte ;
- **L'intégrité** des données, qui atteste que ces dernières n'ont pas été modifiées au cours de leur échange par un tiers.

Plusieurs mécanismes ont été étudiés afin de vérifier ces propriétés selon les deux types de cryptographie : symétrique ou asymétrique.

Le paradigme de la cryptographie asymétrique repose sur un principe simple : chaque participant a une paire de clefs (Pk, Sk), dont une est publique (Pk), tandis que l'autre est secrète (ou privée) (Sk). Une clef publique peut, comme son nom

l'indique, être distribuée librement et est donc principalement utilisée pour chiffrer des messages. Afin de retrouver le message original, la clef privée (associée à la clef publique) est employée par son unique propriétaire. D'après [KL14], un cryptosystème à clef publique est défini de la manière suivante :

Définition 1. *Un cryptosystème à clef publique est défini par trois algorithmes probabilistes à temps polynomial (Generation, Encryption, Decryption)* $= (G, E, D)$ *tels que :*

1. *L'algorithme de génération de clef* G *prend en entrée un paramètre de sécurité d'une taille de* n *bits, dénoté* 1^n*, pour donner en sortie une paire de clefs publique et privée* (Pk, Sk)*.*
2. *L'algorithme de chiffrement* E *prend pour paramètre la clef publique* Pk *ainsi qu'un message* m *appartenant à l'espace des clairs. La sortie de l'algorithme est un chiffré* c *tel que* $c = E_{Pk}(m)$*.*
3. *Finalement, l'algorithme de déchiffrement* D *prend en entrée la clef privée* Sk *et un message chiffré* c*. Il génère un message* m *tel que* $m = D_{Sk}(c)$ *ou un symbole* $\perp$ *en cas d'échec.*

L'objectif de cryptosystèmes asymétriques est d'assurer la confidentialité des données puisque, en théorie, seul le propriétaire de la clef privée associée devrait être capable de retrouver les informations contenues dans le message.

Il est aussi possible d'assurer l'intégrité des données à partir d'un tel cryptosystème à partir d'un mécanisme de signatures.

Définition 2 (d'après [KL14])**.** *Un procédé de signature est un triplet d'algorithmes probabilistes à temps polynomial (Generation, Signature, Verification)* $= (G, S, V)$ *tels que :*

1. *L'algorithme de génération de clef* G *prend en entrée un paramètre de sécurité d'une taille de* n *bits, dénoté* 1^n*, pour donner en sortie une paire de clefs publique et privée* (Pk, Sk)*.*
2. *L'algorithme de signature* S *prend pour paramètre la clef privée* Sk *ainsi qu'un message* m*. La sortie de l'algorithme est la signature* σ *telle que* $\sigma = S_{Sk}(m)$*.*
3. *Finalement, l'algorithme de vérification* V *prend en entrée la clef publique* Pk*, un message* m*, et une signature* σ*. Sa sortie est un bit* b *tel que* $b = V_{Pk}(m, \sigma)$*, valant* 1 *si la signature est valide,* 0 *autrement.*

En pratique, il est courant que l'algorithme de signature soit l'algorithme de déchiffrement, et que l'algorithme de vérification de signature corresponde à l'algorithme de chiffrement. Il faut cependant être prudent sur le message que l'on donne à signer, qu'il est parfois nécessaire de transformer avant signature. Par exemple, dans le cas du célèbre algorithme *RSA* [RSA78], la signature doit se faire sur l'empreinte du message, sous peine d'avoir des attaques [dJC86].

3 ■ INFRASTRUCTURE DE GESTION DE CLEFS

3.1 | PRÉSENTATION GÉNÉRALE

La cryptographie à clef publique fournit aux entités une paire de clefs, permettant, entre autres, de chiffrer ou de signer des données. Contrairement à la clef privée, la clef publique doit être distribuée, afin que les autres entités puissent interagir avec le possesseur. C'est le rôle principal d'une Infrastructure de Gestion de Clefs, ou en anglais *Public Key Infrastructure* (PKI) : fournir un ensemble de mécanismes permettant une diffusion convenable des clefs publiques. Le terme de "convenable", volontairement imprécis, sous-entend qu'une publication naïve des clefs publiques, par exemple dans un annuaire, n'est pas suffisante. Nous pouvons mettre en exergue la problématique grâce à un exemple simple : supposons que Alice souhaite envoyer un message chiffré à Bob. Avant de pouvoir chiffrer son message, Alice doit récupérer la clef publique de Bob Pk_B sur l'annuaire en ligne. A cette étape, il est impossible pour Alice d'avoir une preuve que Pk_B est effectivement la clef publique de Bob. Avec un simple annuaire en ligne, Eve (un individu malintentionné) a pu enregistrer sa propre clef publique en prétendant être Bob. Pour Alice, cela signifie que la clef obtenue est donc Pk_E. Si Eve a accès au chiffré initialement destiné à Bob, elle pourra donc récupérer le message originel, tandis que Bob ne pourra pas déchiffrer le message correctement. Cette situation est le point de départ d'une attaque bien connue dénommée *Man-In-The-Middle* (MITM), où Eve renvoie par la suite le message rechiffré avec Pk_B à Bob. Une PKI doit donc permettre de diffuser les clefs publiques, mais doit aussi garantir que les informations contenues dans l'annuaire sont correctes. En d'autres termes, une PKI doit assurer un lien entre l'identité du possesseur et sa clef publique. Pour cela, un tiers de confiance va délivrer un **certificat** contenant l'identité et la clef publique du possesseur. Ce dernier est signé par le tiers de confiance, afin d'en assurer son intégrité. L'utilisateur doit aussi récupérer le certificat du tiers de confiance pour en extraire la clef publique lui permettant de vérifier la signature. Paradoxalement, pour vérifier un certificat quelconque, l'utilisateur doit d'abord être capable de vérifier le certificat d'un tiers de confiance. Ce problème est résolu par les certificats **auto-signés**, c'est-à-dire que le tiers de confiance va signer lui-même son certificat. Ces derniers sont alors contrôlés hors-ligne (*out-of-band*) pour s'assurer de leur exactitude. Finalement, la dernière difficulté réside donc dans le moyen de récupérer les certificats du tiers de confiance. Plusieurs implémentations de PKI, telles que PKIX ou *Pretty Good Privacy* (PGP) permettent de répondre à cette problématique.

Par la suite, nous définissons précisément les PKI, ainsi que leurs fonctionnalités. Nous détaillons ensuite un des standards actuels, PKIX, avant de s'intéresser à des solutions alternatives.

Une infrastructure de gestion de clefs fournit donc des solutions permettant de publier des clefs publiques associées à l'identité du possesseur, et de certifier que ces entrées sont correctes. De manière générale, plusieurs mécanismes sont mis en place afin de garantir la sécurité du service.

3.2 | ENTITÉS

Les principaux acteurs d'une PKI sont :

- Le propriétaire de la paire de clefs : c'est l'entité souhaitant obtenir un certificat permettant d'adjoindre son identité à sa clef publique ;
- Le tiers de confiance, aussi appelé Autorité de Certification (AC), assurant que le certificat est correct ;
- L'utilisateur, récupérant le certificat du propriétaire afin d'utiliser sa clef publique.

3.3 | FONCTIONNALITÉS ET DÉFINITION

La première étape consiste à **générer les clefs**. Une PKI permet donc à un utilisateur de générer sa paire de clefs en respectant les normes, notamment sur l'aléa utilisé et la taille des clefs. Ensuite, une PKI propose des services afin de **vérifier l'identité** du détenteur de la clef. En général, deux phases sont nécessaires : tout d'abord, le détenteur apporte des preuves de son identité (comme sa carte d'identité). Les vérifications effectuées à cette étape dépendent de la politique de sécurité mise en place. La seconde phase consiste, pour le détenteur, à prouver qu'il est bien le propriétaire de la clef publique qu'il souhaite enregistrer. Ce contrôle est nécessaire pour éviter les entrées erronées (c.-à-d. où une personne ajoute un clef publique qui n'est pas nécessairement la sienne). Une méthode classique repose sur l'auto-signature ; il est demandé au détenteur de signer un challenge, que le service de la PKI vérifie en utilisant la clef publique précédemment fournie. En particulier, ce challenge peut être composé du futur certificat, contenant donc les informations du propriétaire, ainsi que sa clef publique.

Une fois créé, il est donc nécessaire de **publier les certificats**, via un annuaire en ligne, ou par tout autre moyen de diffusion. De manière duale, une PKI propose aussi à un détenteur de **révoquer un certificat**. Dans le cas où sa clef privée est compromise ou perdue, le propriétaire doit pouvoir annuler la validité de son certificat, pour en éviter une mauvaise utilisation par les utilisateurs. Puisqu'il est possible pour un certificat de ne plus être valide, il devient alors nécessaire qu'un service de **vérification de l'état des certificats** soit présent. Ainsi, lors de la récupération du certificat, l'utilisateur emploie le service de vérification de certificat pour s'assurer que ce dernier est toujours valide.

D'autres fonctionnalités, comme la **séquestre de clefs privées** peuvent aussi être proposées, mais ne sont pas nécessaires pour assurer le bon usage d'une PKI.

Comme évoqué précédemment, le lien entre l'identité et une clef publique est réalisé par un **certificat** (initialement défini dans [Koh78]).

Définition 3. *Un certificat est un objet contenant l'ensemble des paramètres relatifs à une clef publique et son propriétaire, délivré et signé par (au moins) un tiers de confiance, généralement appelé AC.*

Une bonne pratique consiste à utiliser un certificat (et donc un jeu de clefs distinct) par type d'application, par exemple une paire de clefs pour le chiffrement, et une autre pour les signatures.

En résumé, une PKI se définit de la manière suivante :

Définition 4 (PKI, d'après [Coo08])**.** *Une Public Key Infrastructure est un ensemble de moyens matériels, logiciels, de composants cryptographiques mis en oeuvre par des personnels, combinés avec des politiques, des pratiques, et des procédures requises, qui permettent de créer, gérer, conserver, distribuer et révoquer des certificats reposant sur la cryptographie asymétrique.*

4 ■ PKIX

PKIX est une infrastructure de gestion de clefs hiérarchique développée par l'*Internet Engineering Task Force* (IETF) utilisant des certificats X.509. Cette infrastructure est largement déployée dans le monde industriel ou sur Internet. La majorité des services Internet fournissant une connexion sécurisée à leur serveur, via *HTTP Secure* (HTTPS) par exemple, repose sur PKIX. Actuellement, PKIX est en version 3 [SAM+13], et de nombreuses *Request For Comments* (RFC) décrivent les différents mécanismes intervenant dans l'infrastructure, comme les *Certificate Revocation Lists* (CRL) [Coo08], la vérification du statut des certificats en ligne [SAM+13], ou encore le protocole de gestion des certificats [KP12].

4.1 | CERTIFICAT X.509

Un certificat X.509 est un type de certificat électronique, utilisé dans PKIX, et dont le format respecte celui défini par la [Coo08]. Plus précisément, un certificat X.509 contient :

- Version du certificat ;
- Nom de l'Autorité de Certification (*Issuer Name*) ;
- Numéro de série du certificat ;
- Nom du propriétaire ;

- Clef publique ;
- Algorithme de signature utilisé par l'AC ;
- Algorithme à utiliser avec la clef publique ;
- Période de validité du certificat ;
- Extensions (optionnelles) ;
- Signature du haché (*hash*, ou encore empreinte) des informations précédentes par l'AC.

D'autres informations relatives au propriétaire du certificat peuvent aussi être contenues dans le certificat. L'ensemble des informations, sans la signature, est dénommé *To Be Signed* (TBS) Les certificats X.509 respectent la syntaxe *Abstract Syntax Notation* (ASN.1), puis sont généralement convertis via un encodage *Distinguished Encoding Rules* (DER).

4.2 | VÉRIFICATIONS DU STATUT DES CERTIFICATS

Avant d'employer la clef publique, l'utilisateur doit s'assurer que le certificat qu'il vient de récupérer est toujours valide. Au delà de la vérification de la date de validité, il faut en effet contrôler que le certificat n'a pas été révoqué. Pour cela, PKIX fournit deux mécanismes permettant de vérifier le statut d'un certificat : hors ligne, via les CRL, ou de manière interactive avec le protocole *Online Certificate Status Protocol* (OCSP).

4.2.1 • LISTE DE RÉVOCATION DES CERTIFICATS

Une *Certificate Revocation Lists*, ou **Liste de Révocation des Certificats**, est une liste contenant les numéros de série des certificats révoqués. Ainsi, l'utilisateur vérifie que le certificat qu'il vient de récupérer n'appartient pas à cette liste. Le format d'une CRL est assez proche de celui d'un certificat X.509 : en sus de la liste des certificats révoqués, complétée par la date et éventuellement la raison de la révocation de chacun, elle contient la date de sa dernière mise à jour, ainsi que celle de sa prochaine. Enfin, l'AC signe l'ensemble des informations précédentes, toujours dans le but d'en préserver l'intégrité. Autrement, un adversaire pourrait ajouter des certificats valides à cette liste pour bloquer des communications, ou pire, supprimer des certificats révoqués. Cette méthode a cependant deux inconvénients. Le premier, c'est qu'il est nécessaire pour le client de récupérer l'ensemble de la CRL. Or, le nombre de certificats révoqués peut devenir important, engendrant un coût de communication non négligeable dû au volume des données échangées. En outre, une CRL a des délais entre les mises à jour, ce qui signifie qu'entre deux, le client peut potentiellement utiliser un certificat révoqué. Afin de réduire le volume des communications, une méthode dénommée δ-CRL [Coo08] a été développée. A partir d'une première CRL dite de "base", les mises à jour vont se faire grâce

aux δ-CRL. Ces dernières contiennent uniquement la liste des nouveaux certificats révoqués relativement à la CRL de base. Ainsi, la quantité de donnée échangée est proportionnelle aux nombres de certificats nouvellement révoqués. Cette solution ne résout cependant pas le problème de l'intervalle entre les mises à jour.

4.2.2 • PROTOCOLE DE VÉRIFICATION EN LIGNE

Le protocole OCSP [SAM+13], ou **protocole de vérification du statut des certificats en ligne** permet de déléguer cette tâche à un serveur distant, a priori mis à jour continuellement. Le fonctionnement est le suivant : lors de la récupération du certificat, le client va envoyer une requête à un répondeur OCSP, contenant l'équivalent du TBS, éventuellement un *nonce* et une signature de ces informations par le client. De son côté, le répondeur va vérifier le statut du certificat dans sa base de données via le numéro de série. La réponse du serveur au client se compose donc de l'état du certificat (bon, révoqué, inconnu), du *nonce* éventuel, et d'une signature de l'ensemble par le répondeur. D'un côté, ce protocole requiert une connexion Internet pour être utilisé, et le client devient dépendant de l'état du répondeur pour débuter une communication sécurisée. D'un autre côté, OCSP allège le volume de communications, mais aussi les opérations effectuées par le client. En effet, le répondeur prend aussi en charge la vérification de la chaîne de certification.

Une solution alternative, dénommée **agrafage OCSP** (*OCSP Stapling*), permet au vérificateur de s'affranchir du contrôle de certificat. Elle a été implémentée comme extension au protocole *Transport Layer Security* (TLS) [Eas11]. L'idée est de déléguer cette étape à l'entité présentant le certificat : le détenteur du certificat demande lui-même à un répondeur OCSP de valider son certificat. Lorsqu'il envoie son certificat, il y adjoint simplement la confirmation de validité donnée par le répondeur. Du côté du vérificateur, il suffit ensuite de vérifier les signatures du certificat et de la réponse pour authentifier son possesseur. Tout d'abord, cette méthode a l'avantage de faciliter la vérification du certificat puisqu'aucune connexion au répondeur OCSP associé au certificat n'est requise. De plus, elle amoindrit le nombre de requêtes d'un répondeur, puisque ce sont majoritairement les entités certifiées par l'AC associée qui vont le contacter.

Enfin, dans le but de diminuer les coûts de calculs et de communications lors de la vérification du statut du certificat, des solutions comme *H-OCSP* [MEFP08] ont aussi été étudiées.

4.3 | CHAÎNES DE CERTIFICATION ET ANCRES DE CONFIANCE

Comme évoqué précédemment, PKIX est une PKI dite **hiérarchique**. Cette qualification est due à sa gestion de la confiance. Lors de la vérification d'un certificat, l'utilisateur va s'assurer que la signature de ce dernier a été réalisée par une AC de confiance. Par la suite, nous expliquons plus précisément comment sont vérifiées ces signatures, et comment la notion de confiance est implémentée dans PKIX.

4.3.1 • CHAÎNES DE CERTIFICATION

En pratique, il est courant que plusieurs AC cohabitent au sein d'une organisation commune. Par exemple, une PKI déployée sur une entreprise possédant plusieurs filiales, et délivrant des certificats à ses employés pour qu'ils puissent s'authentifier, peut avoir déployé une AC distincte par filiale. Cependant, il serait souhaitable que les employés d'un site aient la possibilité de s'authentifier sur un autre site. Plusieurs solutions sont possibles.

Une première idée est de déployer une AC responsable de l'ensemble des filiales, appelée **AC racine** (ou *Root Certification Authority (CA)*). Cette dernière donne ensuite la possibilité aux autres AC, dénommées **AC subalternes** ou **sous-AC**, de signer les certificats des employés. Pour cela, l'AC racine va donc générer son certificat auto-signé, et va ensuite signer le certificat des sous-AC. Cette étape, comparable à l'établissement d'une relation de confiance entre deux AC, s'appelle **la certification croisée** (**cross certification** en anglais). Ensuite, les sous-AC signent les certificats des utilisateurs finaux. La topologie obtenue est alors hiérarchique, et assimilable à un arbre. Ainsi, pour vérifier le certificat d'un utilisateur, il devient alors nécessaire de vérifier **la chaîne de certification**. Afin d'éclaircir les explications, nous dénotons par CA_{Root} le certificat racine, et par CA_1 une des AC subalterne. Le certificat utilisateur est signé par CA_1. La première étape consiste donc à s'assurer que la signature est correcte, en utilisant le certificat de CA_1. Or, un employé provenant du site 2 ne fait pas forcément confiance à cette AC. Il doit alors vérifier de manière similaire une seconde signature : celle du certificat de CA_1 par CA_{Root}. Bien évidemment, la validité (*i.e.* la date et le statut) de chacun des certificats doit être vérifiée à chaque étape. Cette solution s'appelle **la certification croisée hiérarchique**.

La seconde possibilité est de rendre totalement autonome chacune des sous-AC : cela signifie qu'il n'existe plus d'AC racine, mais que toutes les sous-AC deviennent des AC racines. En d'autres termes, tous les utilisateurs finaux font confiance à l'ensemble des CA_i précédemment définis. Dans ce cas, les AC font entre elles **une certification croisée pair-à-pair**. La vérification du chemin de

confiance s'arrête donc à la vérification de la signature par une des AC dans laquelle l'utilisateur a confiance. Bien évidemment, ces deux solutions peuvent être mélangées, pour obtenir une certification croisée dite **hybride**. Plusieurs paramètres peuvent alors être ajustés afin de gérer la confiance donnée entre les différentes configurations [DLR15] :

- la politique de certification (voir 4.4) ;
- la longueur de la chaîne de certification ;
- les contraintes sur le nom.

Finalement, il ne reste plus qu'à définir dans quelles autorités de certification les utilisateurs peuvent avoir confiance.

4.3.2 • MAGASIN D'ANCRES DE CONFIANCE

Nous avons vu que les certificats des autorités de certification étaient auto-signés. Or, dans le cas des utilisateurs, il est clair qu'un tel mécanisme n'est pas suffisant pour contrer les attaques d'impersonnification pour lesquelles les PKI ont été mises en place. Afin d'établir la confiance que l'utilisateur peut avoir dans ces AC, les **magasins d'ancres de confiance** ont été développés. Ces derniers contiennent les certificats des AC, ou **ancres de confiance**, utilisés pour terminer la vérification de la chaîne de confiance. Lors de la dernière étape de vérification, si le certificat d'une AC ayant signé celui que l'utilisateur souhaite vérifier est présent dans le magasin (et est toujours valide), la chaîne de certification est validée. Autrement, l'utilisateur n'est pas capable de remonter jusqu'à une ancre de confiance, et ne peut donc pas authentifier le détenteur du certificat. En un sens, la confiance est donc déportée à l'entité en charge de remplir ce magasin, et à sa capacité à le mettre à jour régulièrement. Les exemples d'utilisation les plus communs sont les navigateurs web, qui ont un magasin contenant les AC de confiance ayant délivré des certificats pour les serveurs Web. Dans le cadre d'une authentification via *Secure Socket Layer* (SSL), l'utilisateur s'assure donc que la clef publique fournie par le site Web pour établir une connexion sécurisée est bien celle du site qu'il souhaite contacter. En d'autres termes, il va donc vérifier que le certificat donné par le site est correct. Lors de la vérification de la signature (ou des signatures selon l'organisation), les conditions d'arrêts sont donc : une chaîne de vérification trop longue, ou l'appartenance (ou non) d'une AC signataire représentant une ancre de confiance.

4.4 | DOCUMENTS NORMATIFS

Deux documents permettent de détailler précisément les procédures employées dans une PKI : la *Politique de Certification* (PC) et la *Déclaration des Pratiques*

de Certification (DPC). Ces deux documents sont normalisés dans [KKSW12] et [KKS15].

La PC est l'équivalent du cahier des charges d'une PKI. Elle décrit l'ensemble des règles qu'elles s'engagent à respecter concernant la gestion des utilisateurs ou des AC. Par exemple, une PC définit les procédures pour :

- la gestion d'une AC : par exemple la génération des clefs, protection des clefs privées ;
- la gestion des utilisateurs : enregistrement d'un nouveau demandeur, vérification de son identité ou encore le traitement des demandes de révocation.

La DPC décrit comment sont réalisées les règles décrites par la PC. En d'autres termes, la PC contient les exigences vérifiées par la PKI, tandis que la DPC décrit les procédures mises en place pour les respecter.

5 ■ ARCHITECTURES DE CLEFS ALTERNATIVES

5.1 | PGP

PGP [Zim95] est une architecture de clefs non hiérarchique développée en 1991 pour faciliter les échanges sécurisés entre les utilisateurs via messagerie électronique. Sa spécificité se situe dans son modèle de confiance pair-à-pair. Comme nous l'avons vu précédemment, dans PKIX, la confiance repose sur les autorités de certification qui sont vues comme des tiers de confiance. Dans PGP (à partir de la version 2.0), un modèle de confiance est défini entre les usagers. Chacun d'entre eux est en mesure de signer un certificat, et donc d'ajouter de la crédibilité quant aux informations contenues dans celui-ci. Prenons l'exemple où Bob possède un certificat. De son côté, Alice a été capable de vérifier hors bandes que les informations du certificat de Bob sont correctes. Elle va alors signer le certificat de Bob pour montrer aux autres utilisateurs qu'il est correct. Imaginons maintenant que Charlie souhaite envoyer un mail chiffré à Bob. Ce dernier va récupérer son certificat, et en évaluer la confiance.

Pour ce faire, Charlie, le vérificateur, va observer toutes les signatures du certificat. A chacune d'entre elles, un niveau de confiance partielle est associé :

- **Nulle** : la signature est ignorée ;
- **Partielle** : chaque signature est prise en compte, mais seul un nombre suffisant $c_p \geq 2$ d'entre elles permet de valider le certificat ;
- **Totale** : le poids de ces signatures est plus important que les précédentes. Il suffit d'en avoir $1 \leq c_t < c_p$ pour accepter le certificat ;
- **Ultime** : le vérificateur connaît la clef privée associée à la clef publique du

certificat.

En théorie, la seule manière que Charlie accepte le certificat de Bob (où Bob et Charlie sont bien distincts) est donc qu'il ait une confiance totale en Alice. En outre, PGP fournit aussi un système pour que les signataires évaluent leur niveau de contrôle des informations contenues dans les certificats vérifiés.

5.2 | AMÉLIORATIONS DES PKI

Les PKI sont un domaine de recherche actif, où de nombreuses améliorations ont été développées au cours des vingt dernières années. Les infrastructures de clefs sont mises en place dans différents contextes, aux besoins d'efficacité et de sécurité variés. Typiquement, une PKI déployée pour un site internet favorisera les aspects liés à la sécurité, tandis que celle utilisée dans une architecture industrielle ou un réseau de capteur respectera des contraintes de déploiement, d'efficacité et de sûreté.

5.2.1 • TRANSPARENCE DES CERTIFICATS

Un autre aspect largement étudié concerne le niveau de confiance donné aux AC. Dans PKIX, une AC est vue comme un tiers de confiance. Cependant, dans le cas où ce dernier est malicieux, l'objectif principal d'authentification visé par la PKI n'est plus vérifié puisque l'AC est capable d'émettre des certificats frauduleux. Afin de limiter la confiance nécessaire dans l'AC, il est possible d'utiliser des journaux d'évènements (ou *logs*) publics où chacune de ces actions est inscrite à l'intérieur : c'est la *transparence des certificats (Certificate Transparency)* [LLK11, LLK13]. Les *logs* sont stockés dans des structures (comme des arbres de Merkle [Mer88]) où seul l'ajout d'information est possible. Par le suite, les gestionnaires des *logs* peuvent prouver qu'un certificat est présent dans cette structure (*i.e.* est donc que le certificat n'a pas été forgé), mais aussi que cette action n'est pas frauduleuse dans le sens où l'arbre des *logs* a bien été agrandi.

5.2.2 • DISTRIBUTION DE LA CONFIANCE

L'architecture dénommée *Accountable Key Infrastructure* [KHP+13] exploite cette pratique de journalisation publique des évènements, mais en distribuant la confiance donnée, en passant de la signature d'une seule AC à plusieurs. Ainsi, une AC voulant émettre un certificat frauduleux devra d'abord obtenir l'aval d'autres AC afin que son certificat soit valide. Une autre solution, décrite dans *Certificate Issuance and Revocation Transparency* [Rya14], permet une distribution de la confiance à plus grande échelle, en utilisant les navigateurs internet des utilisateurs, et est combinée avec une journalisation des évènements effectués (notamment

pour la révocation). Plus récemment, l'infrastructure dénommée *Attack Resilient Public-Key Infrastructure* (ARPKI) [BCK+14] garantit qu'il n'est pas possible de vérifier un certificat forgé dans le cas où au moins une AC n'est pas compromise. *A contrario* de *PKIX*, le client va choisir plusieurs entités de confiance. Parmi elles, au moins deux AC et un serveur de *log*, appelé *Integrity Log Server* (ILS) et chargé de la synchronisation des journaux entres ces serveurs, doivent être choisis. Chacune des entités aura pour rôle de superviser le comportement d'une autre : le serveur ILS doit prouver que le nouveau certificat émis a bien été vérifié et considéré comme correct à la seconde AC, tandis que la première AC s'assure que la seconde fait correctement ces vérifications. Les preuves du bon déroulement des opérations reposent sur des vérification successives de signatures. Toujours dans l'idée d'offrir des garanties de confiance supplémentaires, la solution *Distributed Transparent Key Infrastructure* (DTKI) [YCR16] permet d'assurer qu'un certificat forgé ne sera pas accepté même si tous les participants sont corrompus. Dans les deux cas, la sécurité de ces solutions a été montrée via l'outil de vérification automatique *Tamarin* (voir 8.2).

6 ■ OUTILS CRYPTOGRAPHIQUES

6.1 | CRYPTOSYSTÈMES HOMOMORPHES

Dans la suite, nous utilisons des cryptosystèmes à clefs publiques homomorphes afin de pouvoir effectuer des calculs (addititions et multiplications) sur des chiffrés. Ces cryptosystèmes peuvent être classés selon deux catégories : les cryptosystèmes totalement homomorphes (*fully homomorphic crypto-systems*) et ceux partiellement homomorphes (*partially homomorphic crypto-systems*). Les premiers permettent d'effectuer des calculs arbitraires sur des chiffrés, mais sont coûteux en termes de calculs. A contrario, la seconde catégorie rassemble des cryptosystèmes ne pouvant réaliser que certaines opérations sur les chiffrés, mais généralement pour un coût plus raisonnable. Une liste de cryptosystèmes aux propriétés homomorphes limitées est détaillée dans le papier [Moh11, § 3].

6.1.1 • PROPRIÉTÉS HOMOMORPHES

Dans la suite, nous dénotons par E la fonction de chiffrement (aussi écrite E_{PubX} ou E_i pour préciser la clef publique du joueur X, ou indexé par i selon le contexte) et par D la fonction de déchiffrement ou de signature D (parfois exprimée de manière similaire par D_{PubA} ou D_i). Ces deux fonctions doivent nous permettre d'effectuer des additions sur deux chiffrés c_1 et c_2, dénotées $\text{Add}(c_1;c_2)$, ainsi que des multiplications entre un chiffré et un clair m, dénotées

Mul$(c;m)$. Ces deux fonctions sont définies de la manière suivante : $\forall m_1, m_2 \in \mathbb{Z}/m\mathbb{Z}$: Add$(E(m_1);E(m_2)) = E(m_1 + m_2 \mod m)$ et Mul$(E(m_1);m_2) = E(m_1 m_2 \mod m)$. En particulier, les cryptosystèmes de Benaloh et Paillier [Pai99, Ben94, FLA11], explicités dans les paragraphes suivants (6.1.2 et 6.1.3), vérifient ces contraintes (Add$(E(m_1);E(m_2)) = E(m_1)E(m_2) \mod m$), et *via* une exponentiation pour une multiplication avec un chiffré (Mul$(E(m_1);m_2) = E(m_1)^{m_2} \mod m$), et ont les propriétés homomorphes suivantes :

$$E(m_1)E(m_2) = E(m_1 + m_2 \mod m) \tag{II.1}$$
$$E(m_1)^{m_2} = E(m_1 m_2 \mod m) \tag{II.2}$$

6.1.2 • CRYPTOSYSTÈME DE BENALOH

Le cryptosystème de Benaloh [Ben94, FLA11], dont la sécurité est réduite au problème de résiduosité d'ordre supérieur (*higher residu problem*) a le fonctionnement suivant :

Cryptosystème de Benaloh

Génération de clefs (G) :

1. Soit r la taille d'un bloc. Prendre deux nombres premiers p et q tels que $r|(p-1), \gcd(r, \frac{(p-1)}{r}) = 1, gcd(r,(q-1)) = 1$. On pose $\phi = (p-1)(q-1)$ et $N = pq$.
2. Choisir $y \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $y^{\frac{\phi}{r}} \neq 1 \mod N$. Si r est un nombre composé à la décomposition en nombres premiers telle que $r = p_1 p_2 ... p_k$, alors y doit vérifier pour tout $p_i, 1 \leq i \leq k$: $y^{\frac{\phi}{p_i}} \neq 1 \mod N$.
3. Calculer $x = y^{\frac{\phi}{r}}$
4. Clef publique : $Pk = (y, N)$
5. Clef privée : $Sk = (\phi, x)$.

Chiffrement (E) : Soient m un message tel que $0 \leq m < r$ et u un nombre aléatoire tel que $0 < u < N$. Le message chiffré c est obtenu en calculant : $c = y^m u^r \mod N$
Déchiffrement (D) : Soit $a = c^{\phi} \mod N$. Le message m est retrouvé en calculant : $m = \log_x(a)$.

L'opération de déchiffrement consiste à calculer un logarithme discret, dont la solution est le message m appartenant à $\mathbb{Z}/r\mathbb{Z}$. Par exemple, l'algorithme *Baby-Step, Giant-step* [Sha71] permet d'obtenir une solution en $O(\sqrt{r})$.

6.1.3 • CRYPTOSYSTÈME DE PAILLIER

La sécurité du cryptosystème de Paillier [Pai99] repose sur le problème de la résiduosité composé (*composite residuosity assumption*). Il est défini de la manière suivante :

Cryptosystème de Paillier

Génération de clefs (G) :

1. Prendre deux nombres premiers p et q.
2. Clef publique : $Pk = N = pq$
3. Clef privée : $Sk = (p-1)(q-1)$

Chiffrement (E) : Soient m un message tel que $0 \leq m < N$ et r un nombre aléatoire tel que $0 < r < N$. Le message chiffré c est obtenu en calculant :
$c = (1+N)^m r^N \mod N^2$

Déchiffrement (D) : Le message m est retrouvé en calculant : $m = \frac{(cr^{-N} \mod N^2)-1}{N}$

Le déchiffrement peut être calculé en utilisant une exponentiation modulaire rapide, nécessitant $O(\log N)$ opérations.

6.2 | SÉCURITÉ DES CRYPTOSYSTÈMES ASYMÉTRIQUES

Dans ce paragraphe, nous rappelons les définitions de la sécurité des cryptosystèmes à clef publique. Nous commençons par la définition de l'indistinguabilité de deux ensembles, puis nous définissons la sécurité face aux attaques à clairs choisis ainsi que celle de la sécurité sémantique.

Définition 5 (Indistinguabilité en temps polynomial (*Computational indistinguishability)* [Gol04, Lin17])**.** *Un ensemble de probabilité* $X = \{X(a,n)\}_{a\in\{0,1\}^*,\ n\in\mathbb{N}}$ *est une séquence de variables aléatoires indexées par* $a \in \{0,1\}^*$ *et* $n \in \mathbb{N}$*. Dans le cadre de la cryptographie, la valeur de* a *représente les entrées des joueurs, tandis que* n *est le paramètre de sécurité. Soient* $X = \{X(a,n)\}_{a\in\{0,1\}^*,\ n\in\mathbb{N}}$ *et* $Y = \{Y(a,n)\}_{a\in\{0,1\}^*,\ n\in\mathbb{N}}$ *deux ensembles de probabilité. On dit que* X *et* Y *sont indistinguables, si pour tout algorithme probabiliste et polynomial en temps* $\mathcal{D}$*, il existe une fonction négligeable* $\mu(\cdot)$ *telle que pour* $a \in \{0,1\}^*$ *et* $n \in \mathbb{N}$ *:*

$$|Pr[\mathcal{D}(X(a,n)) = 1] - Pr[\mathcal{D}(Y(a,n)) = 1]| \leq \mu(\cdot)$$

.

On note alors : $X \overset{c}{\simeq} Y$*.*

Grâce à la définition précédente, nous pouvons maintenant définir la sécurité des cryptosystèmes à clefs publiques à partir d'un jeu entre un adversaire et un challenger.

Définition 6 (Sécurité face aux attaques à clairs choisis (IND-CPA)). *Soient $\mathcal{PK} = (G,E,D)$ un cryptosystème à clef publique et $b \in \{0,1\}$. Le jeu dénoté $IND-CPA^b_{\mathcal{PK}}$ entre un adversaire $\mathcal{A}$ et un challenger $\mathcal{C}$ pour un paramètre de sécurité n est défini de la manière suivante :*

$$IND-CPA^b_{\mathcal{PK}}(\mathcal{A},1^n)$$

1. *$\mathcal{C}$ génère une paire de clef : $(Pk,Sk) \leftarrow G(1^n)$, puis $\mathcal{C}$ donne Pk à $\mathcal{A}$;*
2. *$\mathcal{A}$ envoie à $\mathcal{C}$ deux messages m_0, m_1 tels que $|m_0| = |m_1|$;*
3. *$\mathcal{C}$ calcule : $E_{Pk}(m_b)$, puis l'envoie à $\mathcal{A}$*
4. *$\mathcal{A}$ envoie un bit b'. $\mathcal{C}$ termine le jeu en retournant b' (ou 0 en cas d'abandon de $\mathcal{A}$)*

On dit que $\mathcal{PK}$ est sûr face aux attaques à clairs choisis (Chosen Plaintext Attack) si pour tout n et pour tout $\mathcal{A}$ probabiliste à temps polynomial, il existe une fonction négligeable $\mu(\cdot)$ telle que l'avantage de l'adversaire soit négligeable :

$$Adv(\mathcal{A}) = |Pr[IND-CPA^1_{\mathcal{PK}}(\mathcal{A},1^n)=1] - Pr[IND-CPA^0_{\mathcal{PK}}(\mathcal{A},1^n)=1]| \leq \mu(1^n)$$

Une autre manière de définir la sécurité des cryptosystèmes est la suivante :

Définition 7 (Sécurité Sémantique [GM84, Gol04]). *Un cryptosystème à clef publique (G,E,D) est sémantiquement sûr si pour tout algorithme non uniforme probabiliste à temps polynomial (n-u.p.t.p.) A, il existe un algorithme n-u.p.t.p. A' tel que pour tout ensemble de probabilité X, avec $|X_n| \leq poly(n)$, pour toute paire de fonctions $f,h : \{0,1\}^* \rightarrow \{0,1\}^*$ bornées polynomialement, pour tout polynôme positif $p(\cdot)$ et tout n suffisamment grand :*

$$\begin{aligned} \Pr_{(Pk,Sk)\leftarrow G(1^n)}&[A(1^n,k,E_{Pk}(X_n),1^{|X_n|},h(1^n,X_n)) = f(1^n,X_n)] \\ &- Pr[A'(1^n,E_{Pk}(X_n),1^{|X_n|},h(1^n,X_n)) = f(1^n,X_n)] \leq \frac{1}{p(n)} \end{aligned}$$

Ces deux définitions sont en réalité équivalentes [Gol04]. Dans [WSI02], la preuve est également étendue aux cas où l'adversaire à accès à des couples clairs/chiffrés. Par la suite, nous ferons appel aux deux notions selon le contexte.

7 ■ CALCUL MULTI-PARTIES SÉCURISÉ (MPC)

Le calcul multi-parties sécurisé (ou *Secure Multi-Party Computation*, abbrégé MPC) est un domaine de la cryptographie ayant émergé avec les travaux de [Yao82], avec la question suivante : dans un groupe de millionnaires, comment savoir qu'elle est la plus grande fortune sans que chacun des participants ne révèlent la sienne ? De manière plus générale, le MPC s'intéresse au problème suivant : comment un groupe de n participants peut calculer une fonction f telle que seul le résultat du calcul soit appris d'une partie des participants et que leurs entrées restent secrètes ? Depuis, de nombreuses techniques ont été développées, notamment en utilisant le partage de secret de Shamir (*Secret Sharing*) [Sha79, CD+15]. Dans cette section, nous définissons les paramètres utilisés par la suite, ainsi que le modèle de preuve.

7.1 | CANAUX DE COMMUNICATIONS ET ADVERSAIRES

Un des paramètres fondamentaux lors de l'évaluation de la sécurité des protocoles *Multi-Party Computation* (MPC) concerne les capacités des adversaires. Dans notre cas, l'adversaire a une puissance de calcul limitée, et est modélisé comme un algorithme probabiliste à temps polynomial. Dans le cadre du MPC, les adversaires sont vus comme des ensembles de joueurs, et plusieurs paramètres sont à définir. Tout d'abord, nous nous limitons à une corruption **statique** *i.e.* l'ensemble des adversaires corrompus est défini à priori, et reste le même durant toute l'exécution du protocole. Ensuite, il faut caractériser le comportement des adversaires : ces derniers peuvent être passifs, c.-à-d. essayer de glaner des informations durant l'exécution du protocole, mais en respecter les contraintes. Dans ce cas, on parle alors d'adversaires **semi-honnêtes**, ou encore **honnêtes-mais-curieux**. *A contrario*, les adversaires dits **actifs** ou **malicieux** ont la possibilité de ne pas respecter le protocole dans le but d'obtenir des informations. Ils peuvent effectuer des échanges supplémentaires ou encore modifier la distribution de valeurs aléatoires qu'ils fournissent. Nous nous plaçons dans un modèle classique où l'arrêt du protocole durant son exécution n'est pas considéré comme un problème de sécurité.

En termes de canaux de communication, nous ne supposons pas que les canaux sont sécurisés, ce qui signifie que l'adversaire est capable de récupérer toutes les informations circulant sur le réseau par espionnage. Cependant, ils ont à disposition une PKI leur permettant d'obtenir la clef publique associée aux autres joueurs, et donc de réaliser des canaux sécurisés en chiffrant et signant les échanges. Dans ce cas, le coût de ces opérations est explicité.

Dans ce cadre d'exécution, les bornes théoriques de sécurité sont extrêmement larges : que l'adversaire soit passif ou actif, il est possible de réaliser des protocoles

sûrs sans restriction sur le nombre d'adversaires.

7.2 | DÉFINITION DE LA SÉCURITÉ

Soit Π_ϕ un protocole multi-parties permettant à n joueurs de calculer la fonction $\phi : \{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^* \times \cdots \times \{0,1\}^*$, définie comme $\phi(x_1,\ldots,x_n) = (y_1,\ldots,y_n)$ avec $x_i \in \{0,1\}^n$ l'entrée du joueur P_i et $y_i = \phi_i(x_1,\ldots,x_n) \in \{0,1\}^n$ la valeur de sortie que P_i obtiendra à la fin de l'exécution de Π_ϕ. Pour alléger les notations, nous dénotons par $\mathcal{X}$ le domaine de définition de ϕ, et par $\mathcal{Y}$ son image, telle $\phi(\mathcal{X}) = \mathcal{Y}$. Lorsqu'un joueur ne doit pas obtenir de sortie, nous utilisons le symbole **-**.

Nous nous intéressons à la technique de preuves par simulation [Gol04, Lin17]. L'idée générale de la méthode est de montrer que l'exécution *réelle* du protocole n'apporte pas plus d'information que celle dans un monde dit *idéal*. Ce dernier définit un cadre d'exécution pour la fonction ϕ, ici appelée *fonctionnalité idéale*, et dénotée $\mathcal{F}_\phi(\mathcal{X}) \to \mathcal{Y} = \phi(\mathcal{X})$. La fonctionnalité idéale est donc définie de la même manière que la fonction ϕ : à entrées similaires, elle doit renvoyer une sortie correcte. La manière dont la fonction est exécutée n'est pas connue : en d'autres termes, $\mathcal{F}_\phi$ est un tiers de confiance qui permet de calculer la fonction ϕ en boîte noire, où le résultat est donc garanti d'être correct. Dans un monde idéal, chaque joueur P_i,$(i \in N)$ envoie ces entrées à $\mathcal{F}_\phi$. Cette dernière leur renvoie, après calcul, leur sortie respective $\phi_i(\mathcal{X})$. Dans ce cadre, les possibilités d'attaque des joueurs sont donc très limitées : d'où le nom de monde idéal.

La preuve consiste donc à montrer que les joueurs n'obtiennent pas plus d'informations dans ce monde que lors d'une exécution réelle du protocole. Une exécution réelle est définie comme une exécution du protocole entre les joueurs telle que définie par l'algorithme. A l'instar des preuves à divulgation nulle de connaissance (*Zero Knowledge*), la preuve repose donc sur l'écriture d'un algorithme appelé **simulateur**, dont l'objectif est de recréer l'ensemble de valeurs/messages appris par un ensemble de joueurs lors d'une exécution du protocole. Plus précisément, le simulateur doit envoyer la retranscription de l'exécution du protocole, en utilisant uniquement les entrées et sorties de cet ensemble de joueurs. La taille de l'ensemble est déterminée en fonction d'un seuil t (pour *threshold*) de sécurité (avec $t \in \mathbb{N}$ et $t < n$). Ainsi, le protocole est sécurisé pour un nombre t de joueurs corrompus parmi n si pour tout ensemble de t joueurs corrompus, il est possible de construire un simulateur retranscrivant une exécution réelle du protocole uniquement à partir des entrées x_i et des sorties y_i fournies par $\mathcal{F}_\phi$.

La retranscription d'un protocole s'appelle **la vue** (*view*). Soit $N_\mathcal{A}$ l'ensemble des indices des joueurs corrompus. Une vue est définie formellement comme : $View_\mathcal{A}^{\Pi_\phi}(\{\{x_i\},\ i \in N_\mathcal{A}\}) = \{r^i, \{m^i_j\}, i \in N_\mathcal{A}\}$ où r^i est la bande aléatoire interne de

chaque P_i, et m^i_j est le j^{th} message reçu par P_i.

Grâce à cette notion de vue, il est possible de définir précisément la sécurité d'un protocole MPC :

Définition 8 (Sécurité d'un protocole MPC Π_ϕ [Lin17]). *Soit Π_ϕ un protocole MPC permettant le calcul d'une fonction ϕ. On dit que Π_ϕ permet de calculer de manière sûre la fonction ϕ en présence de t adversaires semi-honnêtes si pour tout ensemble d'au plus t joueurs dénoté $\mathcal{A}$, il existe un algorithme $S_\mathcal{A}$ probabiliste à temps polynomial dénommé simulateur, tel que :*

$$S_\mathcal{A}(\{\{x_i\},\ i \in N_\mathcal{A}\}, \mathcal{F}_\phi(\{\{x_i\},\ i \in N_\mathcal{A}\}) \overset{c}{\simeq} View^{\Pi_\phi}_\mathcal{A}(\{\{x_i\},\ i \in N_\mathcal{A}\}, Output^{\Pi_\phi})$$

8 ■ ASSISTANTS DE PREUVES AUTOMATIQUES

Par la suite, nous avons parfois réalisé des preuves de sécurité en utilisant des assistants de preuve automatiques. En particulier, ce sont *Proverif* et *Tamarin* que nous avons majoritairement utilisés. Dans cette partie, nous présentons les particularités de ces deux logiciels.

8.1 | MODÈLE DE L'ADVERSAIRE ET SÉCURITÉ SYMBOLIQUE

8.1.1 • ADVERSAIRES DE DOLEV-YAO

Dans les assistants de preuve, les adversaires suivent généralement le modèle dit de Dolev-Yao [DY83]. Cette formalisation donne le contrôle du réseau à un adversaire actif : il est capable d'espionner les communications (et donc de récupérer les informations échangées), de modifier voire bloquer des messages et d'en injecter d'autres. Les messages injectés sont extraits des connaissances de l'adversaire, et de ce qu'il est capable d'en déduire.

8.1.2 • SÉCURITÉ SYMBOLIQUE

Dans le modèle symbolique, aussi appelé de Dolev-Yao [DY83] ou de Needham-Schroeder [NS78] plusieurs hypothèses sont faites :

- Les primitives sont vues comme des boîtes noires, et sont supposées parfaitement sûres ;
- Les messages sont vus comme des termes de ces primitives ;
- L'adversaire est limité au calcul de ces primitives, mais son champ d'action peut généralement être augmenté via des équations.

La relation entre la sécurité symbolique et calculatoire a été largement étudiée. Citons par exemple le papier [AR00], qui montre que dans le cas d'un adversaire passif, si la propriété de secret est vérifiée dans le modèle symbolique, alors elle l'est aussi dans le modèle calculatoire. Un résumé des résultats obtenus se trouve dans l'article [CKW11].

L'idée générale des preuves consiste à calculer les connaissances de l'adversaire. Malheureusement, le nombre d'états de ces protocoles est théoriquement infini. De plus, il n'y pas a de limitation sur la taille des messages ni sur le nombre d'exécution du protocole. Dans ce cas, le problème de la vérification est indécidable [DLMS04].

Plusieurs approches ont donc été étudiées pour contourner cette limitation. La première consiste à borner le nombre d'états possibles du protocole. Si cette méthode permet de trouver des attaques, elle ne suffit plus à démontrer la sécurité. Une seconde technique repose sur le bornage du nombre de sessions. Si le problème devient décidable, il demeure difficile (*NP-Complet*). De multiples outils ont alors été développés, en s'appuyant sur différentes méthodes. Ainsi, ProVerif utilise des abstractions reposant sur les clauses de Horn, tandis que Tamarin fournit des preuves qui peuvent potentiellement requérir des interactions avec l'utilisateur.

8.2 | TAMARIN

Tamarin [MSCB13] est un outil permettant la vérification automatique de propriétés de sécurité sur des protocoles. Comme pour ProVerif, il permet d'effectuer des vérifications sur un nombre non borné de sessions et repose sur un modèle d'adversaire de type Dolev-Yao [DY83]. Tamarin utilise aussi un modèle cryptographique parfaitement sûr, c.-à-d. que l'adversaire n'obtiendra jamais d'information sur un chiffré tant qu'il n'obtient pas la clef privée associée.

De manière plus formelle, *Tamarin* est outil permettant de vérifier la satisfaisabilité ou la validité d'une formule ϕ définissant une propriété sur des traces. Ces dernières sont déduites d'un système de réécriture de multiensembles R représentant un protocole modulo une théorie équationnelle E. L'idée est donc d'effectuer une recherche exhaustive sur les exécutions possibles, construites comme des déductions symboliques à partir de R et E. La particularité de Tamarin est d'employer les résultats de [CD05] pour réduire le raisonnement avec R modulo E à un raisonnement sur des variantes de R modulo AC (associativité-commutativité).

Les protocoles sont donc modélisés comme des règles de réécriture sur des multiensembles. Elles sont appliquées sur des faits (*facts*), qui représentent les connaissances des joueurs et leurs échanges, comme la réception ou l'envoi d'un message ou encore la génération d'un nombre aléatoire. Les faits sont ensuite consommés par les **règles** (`rule`) pour être transformés dans un autre fait. Autrement dit, les faits représentent l'état d'un joueur à un instant. S'il remplit les conditions suffisantes d'une règle, elle est alors appliquée pour le transformer dans un autre

état. Concernant la modélisation, les règles sont donc utilisées pour représenter les actions des joueurs. Prenons par exemple l'action de chiffrer un message. Pour cela, nous définissons un fait $\mathcal{F}_1$, dépendant des termes représentant un message (m) ainsi qu'une clef (k). Nous lui associons ensuite une règle de chiffrement appelée $\mathcal{C}$ définissant le terme c tel que $c \leftarrow E(k,m)$, où E est la fonction de chiffrement. Le joueur passe ensuite dans un nouvel état, modélisé par le fait $\mathcal{F}_2$ incluant c. Schématiquement, cela donne :

$$\mathcal{F}_1(k,m) \rightsquigarrow_{\mathcal{C}} \mathcal{F}_2(c)$$

Les états peuvent être persistants (ce qui est dénoté par `!` au début de son nom) ou linéaires. Une règle peut consommer un état persistant autant de fois qu'elle le désire, tandis qu'elle ne peut utiliser un fait linéaire qu'une seule fois.

La seconde partie de la modélisation consiste à définir les propriétés de sécurité que nous souhaitons démontrer à l'aide de Tamarin. Elles sont symbolisées par des lemmes (`lemmas`), qui sont des formules de logique de premier ordre appliquées aux règles composant le protocole. Ces propriétés contiennent donc des quantificateurs ($\forall$: `All`, $\exists$: `Ex`) et des liens logiques (`&`, `|`, `not`, `==>`). De plus, et c'est une des particularités de Tamarin, elles peuvent aussi contenir des marqueurs temporels, déclarés par `#` et appelés avec la syntaxe `@`. Par exemple, le lemme : `Ex m k #t. C(k,m) @t`, est la transcription de la propriété : il existe m, k et un temps t tels que la règle C soit appliquée au temps t. Ce type de lemme est particulièrement utile pour s'assurer que le protocole modélisé s'exécute correctement.

8.3 | PROVERIF

ProVerif est un vérificateur de protocoles cryptographiques [Bla14]. Il repose sur la modélisation des protocoles utilisant une syntaxe proche du π-calcul, généralement utilisée pour représenter des processus concurrents. Comme précédemment, des règles sont écrites pour modéliser des actions, et leurs dérivations sont conservées dans des traces. Les propriétés de sécurité sont modélisées comme des requêtes de dérivations (`query`). La particularité de ProVerif est d'utiliser une sur-approximation du modèle par des clauses de Horn (de premier ordre). Elle donne de bons résultats en pratique, mais reste en général indécidable. Par conséquent, il est possible que l'outil ne termine pas. Cette approximation est sûre (*sound*), mais incomplète : la sécurité de l'approximation implique celle du protocole implémenté, mais une attaque trouvée sur l'approximation n'implique pas que cette attaque soit applicable sur le protocole.

CHAPITRE III

ARCHITECTURE DE SÉCURITÉ D'UN MODULE DE RUPTURE PROTOCOLAIRE

Sommaire

RÉSUMÉ

Dans ce chapitre, nous présentons l'architecture d'un module de sécurité dédié à l'amélioration de la sécurité des communications réseaux, et particulièrement adapté aux architectures de contrôle-commande. L'exemple d'application auquel nous nous référons le plus souvent est une infrastructure contrôle-commande de type SCADA. Une telle architecture est caractérisée par l'hétérogénéité des équipements qui la composent, et a des besoins différents de ceux d'un réseau classique de type LAN et MAN. L'objectif principal du module est d'effectuer de la rupture de protocoles de manière efficace et sûre. Grâce à la rupture protocolaire, l'analyse des flux réseaux est facilitée, et la diffusion des attaques est limitée. Nous présentons ici une architecture (physique et logique) d'un tel module, en nous attardant en particulier sur la gestion des clefs et des certificats. En effet, cette dernière est particulièrement délicate, puisqu'un tel module est capable d'analyser des flux chiffrés, tout en étant aussi transparent que possible afin de s'insérer facilement dans un réseau dont la production est continue. Une partie des résultats de ce chapitre a donné lieu à une publication [BDD+17].

ORGANISATION DU CHAPITRE

Nous commençons par décrire l'organisation des réseaux industriels (1.1), ainsi que la gestion des problématiques liées à la sécurité de ces derniers (1.2). Nous mettons ensuite en exergue la nécessité de développer un module de rupture protocolaire (1.3). Nous nous intéressons ensuite à l'architecture de sécurité d'un tel module (2). Par la suite, nous analysons cette architecture afin de montrer les propriétés de sécurité qu'elle apporte (3), avant de finir par un exemple d'implémentation concret dans le cadre du projet "Architecture Robuste pour les Automates et Matériels des Infrastructures Sensibles" (ARAMIS) (4).

1 ■ SÉCURITÉ DES RÉSEAUX INDUSTRIELS

Dans cette section, nous définissons les architectures de contrôle-commande, et explicitons les équipements de protection les plus utilisés.

1.1 | SÉCURITÉ DES ARCHITECTURES DE CONTRÔLE-COMMANDE INDUSTRIELLES

1.1.1 • ARCHITECTURES DE CONTRÔLE-COMMANDE ET SCADA

Aux alentours des années 1960, les systèmes dénommés SCADA, abréviation de *Supervisory Control and Data Acquisition* se sont démocratisés dans les industries [Boy09]. Ils permettent de superviser des processus automatiques, en collectant, parfois de manière continue, des données de processus exécutés en temps réel. Ils sont généralement associés à des systèmes numériques communément appelés *Industrial Control System* (ICS), qui en plus de la supervision, permettent de donner des ordres à des machines physiques. Les architectures de contrôle-commande sont déployées dans divers domaines, tels que les industries chimiques, les applications militaires, l'énergie... Plus précisément, une architecture contrôle-commande se découpe en cinq niveaux :

- **Niveau** 0 : Les capteurs et actionneurs
- **Niveau** 1 : Traitement d'automatisme
- **Niveau** 2 : Supervision et contrôle de procédé
- **Niveau** 3 : Information de synthèse pour l'usine
- **Niveau** 4 : Système d'information à destination d'autres métiers de l'exploitation.

De manière générale, les capteurs situés au **Niveau** 0 sont reliés à des automates de **Niveau** 1 via un bus. Ils sont utilisés pour mesurer des grandeurs physiques, telles que la température ou la pression, qui sont ensuite transformées en signaux électriques à destination des automates. Ces derniers traitent les signaux envoyés par les capteurs et renvoient des ordres en fonction des données qu'ils ont reçues en entrée. L'objectif d'un automate dépend de son environnement : il peut être utilisé afin de maintenir la production, ou d'assurer la sécurité du système (en effectuant un arrêt des actionneurs en cas de problème par exemple). Les automates sont reliés au **Niveau** 2 à un ensemble de machines fournissant une *Interface Homme-Machine* (IHM). Ces machines fournissent une vision globale du processus, et permettent aussi d'assigner des consignes aux automates. Le terme SCADA est généralement associé à ces machines. Elles sont reliées au **Niveau** 3 en envoyant des informations de synthèse, utiles pour surveiller l'état des SCADA à long terme (notamment avec l'archivage des données), et assurer la sûreté de fonctionnement

du système. Les actions de commande ne sont généralement pas possible à ce niveau. Finalement, le **Niveau** 4 est associé à la gestion du système informatique dans sa globalité, et n'est pas en relation directe avec les autres niveaux. En pratique, c'est au niveau 4 que les choix techniques, en rapport avec les données récupérées au niveau précédent, tels que la manière de gérer les *logs*, sont mis en place.

De nombreuses contraintes (matérielles et logicielles) rendent la maintenance d'une telle architecture difficile. La première difficulté réside dans la durée de vie des machines, en particulier des niveaux 0 à 3, qui est souvent de plusieurs dizaines d'années. Ainsi, la rétro-compatibilité avec des technologies plus récentes n'est pas nécessairement assurée. De plus, du fait de l'environnement métier dans lequel l'architecture est déployée, l'arrêt des systèmes ne peut se faire facilement, rendant les tâches de maintenance plus ardues. En conséquence, l'architecture de contrôle-commande ne peut être mise à jour régulièrement, mettant potentiellement en péril la sécurité globale du système.

1.1.2 • SÉCURITÉ

La sécurité des architectures de type SCADA (au sens large du terme) est devenue une problématique majeure depuis les années 2000 [BFM04]. Cet intérêt grandissant envers la sécurité est en partie dû à la forte augmentation du nombre d'attaques envers les systèmes industriels [Azi13]. Dans [Tsa10], de nombreuses attaques, ciblées ou accidentelles sont exposées jusqu'en 2009. Plus récemment, des rapports exposent des attaques ciblées sur divers systèmes industriels, dont l'une des plus célèbres est dénommée *Stuxnet* [FMC11]. Le vecteur d'attaque utilisé est une faille logicielle permettant de reprogrammer les automates programmables dits *Programmable Logic Controller* (PLC). Ainsi, ces derniers ne suivaient plus la procédure normale, en ne remontant plus les informations nécessaires au bon suvi de leurs actions. De nombreuses attaques similaires ont été reportées récemment [Mun11]. Le récent exemple de l'attaque *Industroyer* [Bin17] visant une centrale électrique démontre une fois de plus la nécessité d'améliorer la sécurité dans les systèmes industriels.

Compte tenu de la complexité de ce type d'infrastructure, les vecteurs d'attaques sont nombreux. La classification suivante permet de catégoriser les attaques en fonction de leur origine [Tsa10] :

- **Humaine :** la politique de sécurité, c.-à-d. que la gestion de la sécurité n'est pas assez documentée, engendrant des comportements non désirés ;
- **Logicielle :** les vecteurs d'attaque logiciels sont nombreux, qu'ils soient liés à la qualité du code, aux mécanismes d'authentification ou encore à l'environnement d'exécution.
- **Liée à l'infrastructure :** l'absence de mise à jour ou de mécanisme de contrôle d'accès est une cause d'attaques. En termes de réseaux, l'absence

de cloisonnement entre les réseaux ou d'équipements les prédispose à la fuite d'informations et à la propagation de programmes mal intentionnés.

De manière générale, dans un système informatique, les principales propriétés de sécurité à vérifier sont l'intégrité, la confidentialité et la disponibilité. Dans le cadre d'une infrastructure de contrôle-commande, d'autres propriétés viennent s'ajouter, et l'ordre des priorités est modifié. Tout d'abord, **le temps de réponse** du système doit vérifier les contraintes métiers. En particulier, les données échangées entre les automates et les machines de supervision ne doivent pas se trouver ralenties de manière prohibitive par l'apport de sécurité. L'autre aspect du temps de réponse concerne la durée de validité des données, qui doit être bornée afin qu'elles ne puissent pas être réutilisées (*replay attacks*). En pratique, dans les systèmes continus, l'idéal est de s'approcher d'un traitement en temps réel des données. La seconde propriété est **la disponibilité**. Dans ce contexte, il est absolument nécessaire que le système reste aussi réactif que possible, notamment aux plus bas niveaux. Vérifier **l'intégrité** dans un tel système permet de s'assurer que les ordres envoyées n'ont pas été modifiés par un adversaire. Les conséquences d'attaques sur l'intégrité peuvent affecter tous les niveaux : avec l'envoi d'actions non désirées à destination des automates, comme la réception d'informations erronées concernant l'état du système pour les SCADA. Dans une moindre mesure, **la confidentialité** peut également renforcer la sécurité. Dans les niveaux inférieurs, les messages sont répétitifs par nature, et donc leur chiffrement n'est pas forcément nécessaire. Cependant, dans les niveaux supérieurs, le chiffrement des données peut éviter de divulguer des informations sensibles concernant le fonctionnement général du système (telles que celles stockées dans les logs). Par exemple, dans le cadre d'une architecture développée pour le transport d'énergie, le chiffrement des données permet d'éviter que des informations en rapport avec la consommation d'un client (qui sont donc privées) ne soient exploitables.

1.2 | PROTECTION DES RÉSEAUX INDUSTRIELS

Plusieurs solutions ont été développées afin d'ajouter de la sécurité dans les architectures de contrôle-commande. Les équipements réseaux usuels, comme les *Intrusion Detection System* (IDS), pare-feu, ou encore les diodes peuvent aussi être utilisés dans ce contexte. Ces outils permettent de répondre à différents besoins de sécurité : l'analyse et la limitation des flux échangés entre les réseaux pour les deux derniers, et la prévention par détection d'attaques pour les IDS. Dans la suite, nous allons nous intéresser plus particulièrement aux fonctionnalités offertes par les pare-feu et les diodes.

1.2.1 • PARE-FEU

Les pare-feu ont vu le jour en 1988 [DT88] dans le but de filtrer des paquets réseaux en analysant le trafic en fonction des adresses réseaux et des ports. Ces pare-feu sont dits "sans état", ce qui signifie que chaque paquet est analysé indépendamment des autres en fonction de règles prédéfinies. Les pare-feu ont cependant beaucoup évolué, pour fournir des outils d'analyse plus sophistiqués, tels que :

- L'analyse des protocoles à états : *a contrario* des pare-feu sans états, cette fonctionnalité permet d'analyser des protocoles utilisant des sessions, tels que TCP (quatrième couche du modèle OSI).
- La rupture des connexions directes entre deux entités situées de part et d'autre du pare-feu, notamment en utilisant de la translation des adresses réseaux (en anglais *Network Address Translation* (NAT)).
- L'analyse au niveau applicatif (*Proxy*). Dans ce cas le pare-feu est capable d'analyser des protocoles de la couche 7 du modèle OSI, comme *HyperText Transfer Protocol* (HTTP) et *File Transfert Protocol* (FTP). Il permet d'effectuer un filtrage strict des applications circulant sur le réseau en utilisant des listes (blanches ou noires).

1.2.2 • PARE-FEU DE NOUVELLE GÉNÉRATION

Les **pare-feu de nouvelle génération**, ou *Next Generation FireWall* (NGFW), se sont récemment développés afin de fournir des services supplémentaires comblant des lacunes des pare-feu classiques. Ils permettent de filtrer de manière plus efficace les flux entrants et sortant, en les analysant plus finement qu'un pare-feu traditionnel [Tho12]. D'après [PY09, NSS16], un pare-feu est dit NGFW s'il fournit (au moins) :

- Des services équivalents à un pare-feu classique ;
- La reconnaissance des applications (*Application Awareness*) ;
- Un contrôle d'accès fin (par utilisateur ou groupe d'utilisateurs) ;
- L'inspection de flux chiffrés (moins généralement, il est au moins capable d'analyser un flux SSL) ;
- Un système de protection d'intrusion, utilisant une base données de signatures d'attaques connues, et capable de produire des logs de l'attaque ;
- Un système de prise de décisions intelligent, capable de bloquer des sessions en utilisant un service de réputation par exemple.

La reconnaissance des applications intégrée dans les NGFW est extrêmement puissante et permet d'analyser le contenu du flux pour en détecter l'application, *a contrario* d'un pare-feu qui repose généralement sur l'utilisation des numéros de port usuels (c.-à-d. tels que défini dans les *RFC*) pour la détection du flux. En cas d'encapsulation de protocoles, par exemple à l'intérieur du protocole HTTP,

un pare-feu classique autorisera le flux s'il est configuré pour accepter *HTTP*. *A contrario*, un NGFW analysera le contenu du flux HTTP, et selon l'application détectée à l'intérieur, pourra bloquer les échanges. Les NGFW permettent donc de contrer plus d'attaques qu'un pare-feu simple, mais l'absence d'une norme définissant précisément les services qu'ils offrent engendre une efficacité variant fortement en fonction du modèle [Sky14] (de 60% à 99,2% de résistance face à un même échantillon d'attaques).

1.2.3 • DIODES

Une *diode* (*Data Diode* en anglais) est un équipement réseau permettant de rendre une communication entre deux réseaux unidirectionnelle [SP95]. En d'autres termes, lorsque deux réseaux sont connectés via une diode, un des réseaux est émetteur et l'autre récepteur, et leur rôle n'est pas interchangeable. Une diode est employée afin d'améliorer la sécurité des communications entre un réseau dit sûr (ou haut), et un autre considéré plus dangereux (réseau bas). Le choix du sens de la communication est déterminé en fonction de l'application. Par exemple, une diode peut empêcher les communications provenant d'un réseau haut en direction un réseau bas dans le but d'éviter la fuite d'information. Ainsi, le réseau sûr est capable de recevoir des données de l'extérieur, mais n'émet en retour aucune donnée. Inversement, une diode placée afin d'autoriser uniquement les données provenant d'un réseau haut vers un réseau bas peut être utile pour des actions de suivi. Par exemple, en plaçant une diode entre un niveau 2 et 3 d'une infrastructure de contrôle-commande (où le niveau 2 est considéré comme sécurisé), les SCADA sont capables d'envoyer les journaux (*log*) vers le centre de supervision. Ce dernier peut être relié à un réseau public, et dans le cas d'une compromission, l'attaque ne pourra pas être diffusée jusqu'aux SCADA. Les diodes sont donc une protection contre des attaques utilisant le réseau comme vecteur.

1.3 | MODULE DE RUPTURE PROTOCOLAIRE

Généralement, les outils présentés précédemment sont utilisés dans le cadre de la protection de réseaux, et ne répondent pas nécessairement aux contraintes des infrastructures de contrôle-commande. En particulier, l'objectif principal des NGFW est de protéger des réseaux locaux des zones démilitarisés (*DeMilitarized Zone* (DMZ)). Ils permettent de limiter les attaques entrantes, et ont aussi vocation à filtrer le trafic des utilisateurs (donc le trafic sortant). De plus, les NGFW sont capables de fournir une séparation stricte entre les réseaux, notamment avec le NAT ou l'inspection des flux SSL. Le NGFW étant souvent placé comme l'unique interface entre le réseau local et les réseaux publics, il agit comme un goulet d'étranglement sur le débit. En particulier, l'accumulation des services

offerts, et plus précisément l'analyse des flux chiffrés (SSL, TLS) font chuter les débits normalement offerts par le réseau de 75% pour du chiffrement sur 512 ou 1024 bits, à plus de 80% avec 2048 bits de sécurité [Pir13]. Le déploiement de NGFW dans une infrastructure où le temps de réponse est primordial n'est donc pas recommandé. De plus, l'architecture est souvent extrêmement étendue et ne converge pas nécessairement vers un noeud commun séparant le réseau interne de l'externe. Au contraire, elles sont souvent constituées de plusieurs branches distinctes, où chacune d'entre elles doit être sécurisée. Dans le cas des diodes, la limitation à un flux réseau unidirectionnel ne permet néanmoins pas de protéger les réseaux contre toutes les attaques. Par exemple, la propagation d'un *malware* est toujours possible, puisque ne nécessitant pas d'échanges bidirectionnels [OSH13]. Enfin, elle n'apporte aucun gain de sécurité pour le réseau émetteur, qui doit être protégé par d'autres équipements (comme des NGFW). De plus, la principale contrainte des diodes est intrinsèque à leur mode de fonctionnement : l'impossibilité d'avoir des échanges bi-directionnels. D'une part, la mise en place de protocoles à états devient complexe. Certaines diodes sont cependant capables d'émettre des segments *Transmission Control Protocol* (TCP) [BC12] en simulant elles-mêmes les acquittements à l'émetteur. Les paquets sont ensuite transformés vers des datagrammes *User Datagram Protocol* (UDP) à destination du récepteur. D'autre part, dans le cadre des *Supervisory Control and Data Acquisition* (SCADA), l'isolation d'un réseau n'est pas souhaitable puisqu'elle rendrait le déploiement de mises à jour encore plus fastidieux.

Actuellement, aucune solution ne permet donc de protéger de manière efficace les systèmes de contrôle-commande. L'utilisation des équipements réseaux classiques, éventuellement combinés, ne permet pas de répondre aux exigences de ces architectures. Notre objectif est donc de développer un nouvel équipement, permettant d'apporter de l'isolation entre les réseaux et capable d'analyser des flux réseaux. Cette analyse des flux doit pouvoir s'appliquer sur un trafic chiffré, et doit être suffisamment efficace pour répondre aux contraintes du système cible. Enfin, elle doit être adaptable à l'environnement dans lequel l'équipement est placé, c.-à-d. se conformer à la politique de sécurité en vigueur. L'isolation doit permettre de couper des communications, mimant le comportement d'une diode, mais dont le choix du sens de la communication est fait en fonction du flux. De plus, elle doit permettre l'utilisation de protocole de haut niveau, reposant sur des sessions. Dans la suite, nous proposons une solution répondant à ces besoins via un nouvel équipement réseau dénommé **module de rupture protocolaire** :

Définition 9 (Module de rupture protocolaire)**.** *Un module de rupture protocolaire est un équipement réseau orienté sécurité permettant l'isolation de réseaux ainsi que la définition et l'application d'une politique de sécurité par analyse des flux réseaux.*

De nombreuses questions sont motivées par la conception d'un module de rupture

protocolaire. Au regard de ce chapitre, nous nous intéressons aux aspects liés à la sécurité : celle offerte par le module, mais aussi celle en lien avec sa conception. En particulier, l'insertion d'un nouvel équipement, dans une infrastructure de sécurité existante, fournissant de l'analyse de flux chiffrés et de l'isolation, requiert une étude quant à la répartition des clefs et des certificats présents. La problématique à laquelle nous répondons dans ce chapitre est donc la suivante :

Comment concevoir une architecture de clefs pour un module de rupture protocolaire ?

2 ■ ARCHITECTURE DE SÉCURITÉ D'UN MODULE DE RUPTURE PROTOCOLAIRE

2.1 | ARCHITECTURE MATÉRIELLE

Un module de rupture protocolaire, tel que défini précédemment, a pour principaux objectifs de segmenter des réseaux et d'analyser le trafic. D'un point de vue matériel, l'isolation nécessite que le module soit au moins divisé en trois parties distinctes : deux d'entre elles seront utilisées pour communiquer avec les entités externes entre lesquelles le module est placé, tandis que la troisième sera en charge des tâches en rapport avec la sécurité. La figure 1 illustre cette architecture. Nous définissons les parties externes du module comme étant **les guichets**, et la partie centrale comme **le coeur**. Afin d'assurer l'isolation des réseaux entrants et sortants du module, chacune des parties doit être indépendantes et physiquement distinctes. Un exemple d'implémentation de ce principe consiste à utiliser trois systèmes autonomes, tels que des *System on Chip* (SoC), comprenant chacun au moins un processeur et de la mémoire vive. Il suffit ensuite de relier les guichets au coeur pour permettre des communications entre les différents composants.

Les guichets ont pour objectif de gérer les communications avec les entités externes : ils représentent les entrées/sorties du module. Compte-tenu de son emplacement dans le module, le coeur est donc physiquement isolé : il n'est plus en communication directe avec les entités externes au module. Cette isolation est idéale pour principalement associer le coeur aux tâches liées à la sécurité. Afin d'assurer un meilleur niveau de sécurité, ce dernier est alors être directement relié à un HSM [MG14]. Un HSM est un composant (matériel et logiciel) capable d'effectuer des opérations cryptographiques (telles que le déchiffrement ou les signatures), et de gérer un coffre fort contenant des informations sensibles, comme des clefs privées. Ainsi, le coeur fera donc appel au HSM pour utiliser ces clefs privées, qui n'auront pas à sortir de l'environnement sécurisé offert par le HSM. D'un point de

vue matériel, cette architecture fournit donc une double protection : la première par les guichets externes, qui agissent comme un premier garde-fou vis à vis des communications externes. La seconde est obtenue par l'ajout d'un composant sécurisé dédié aux calculs cryptographiques, dont les appels sont envoyés par le coeur. Cependant, comme les guichets gèrent directement les échanges externes, ils doivent aussi être capables d'effectuer des opérations cryptographiques symétriques afin d'alléger la charge de travail du coeur. Pour cela, un composant de calcul dédié, ou un processeur incluant une implémentation efficace des algorithmes de chiffrement symétrique usuels, est suffisant. D'un point de vue de la sécurité, les clefs symétriques ne représentent pas une information aussi critique qu'une clef privée puisqu'elles sont régulièrement régénérées afin d'assurer la *Perfect Forward Secrecy*.

2.2 | LA RUPTURE PROTOCOLAIRE

2.2.1 • FONCTIONNEMENT

L'objectif du module est d'effectuer de la rupture protocolaire, c.-à-d. qu'une communication point à point entre A et B sera divisée en deux communications : une première entre A et le module, puis une seconde entre le module et B. En théorie, il faudrait que l'ajout d'un module soit aussi transparent que possible pour les entités qui communiquaient auparavant. De manière générale, le module de rupture protocolaire, tel que nous le définissons, peut être relié à plusieurs entités à gauche et à droite (soient de multiples clients et serveurs), communiquant de manière bidirectionnelle. Une configuration classique consiste à le placer entre des clients et des serveurs déjà en communication afin de sécuriser leurs échanges. La majorité des configurations requises pour établir à nouveau la communication entre le client et le serveur sont effectuées sur le module, et sont détaillées dans les sections suivantes. En pratique, les modifications requises sur le client et le serveur sont équivalentes à celles effectuées lors de de l'ajout d'un serveur mandataire (c.-à-d. mettre-à-jour les adresses réseaux avec lesquelles le client et le serveur doivent communiquer).

Dans la suite, nous supposons cependant que le module est simplement placé entre un client (C) et un serveur (S), où le client souhaite communiquer de manière usuelle avec le serveur. Cette disposition nous permet de nous concentrer sur l'explication de la rupture protocolaire où le sens de la première communication est préétabli. De plus, nous supposons pour le moment que le module a correctement été intégré dans l'infrastructure existante c.-à-d. que les communications entre les trois entités sont fonctionnelles. La figure 1 schématise l'architecture générale du module ainsi que le processus de rupture dans cette configuration.

L'architecture tripartie présentée précédemment permet d'associer chacun des

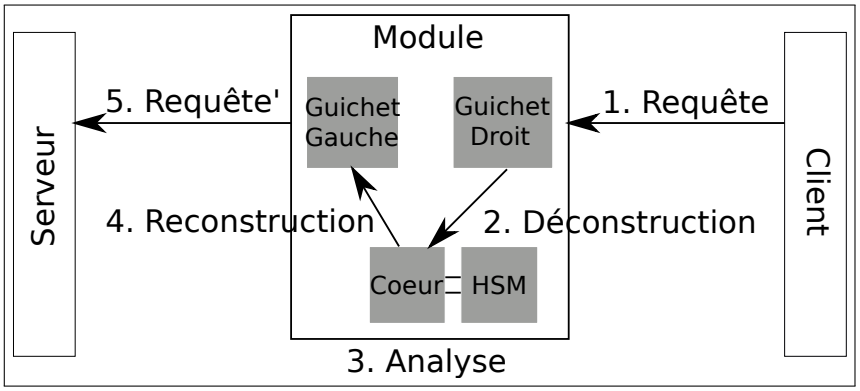

FIGURE 1 – Architecture générale et processus d'analyse d'un module de rupture de protocole entre un Client et un Serveur

composants autonomes à une tâche spécifique. Dans notre cas, nous définissons donc trois processus spécifiques caractérisant la rupture protocolaire, qui sont :

1. La déconstruction ;
2. L'analyse ;
3. La reconstruction.

Le client souhaite donc communiquer de manière usuelle avec le serveur. Puisque le module est placé entre les deux, cela signifie que le guichet droit va recevoir le flux réseau. La première étape, réalisée par le guichet récepteur, est celle de la **déconstruction**. Elle consiste à transposer le flux reçu dans un langage intermédiaire. Ce dernier doit contenir l'ensemble des informations nécessaires, c.-à-d. le contenu et les metadonnées du flux original, permettant de rediriger correctement le message. Dans notre exemple, ce langage doit donc permettre de fournir des informations sur l'origine (C), la destination (S) et le contenu de la requête. Une fois la traduction effectuée, le message est finalement envoyé au coeur via les canaux de communications internes au module.

Le coeur commence alors à **analyser** le contenu du message, en utilisant des filtres spécifiquement configurés pour ce module, c.-à-d. qu'ils sont écrits en fonction du domaine d'application dans lequel intervient le module. Ils permettent de s'assurer que les informations contenues dans le message coïncident avec la politique de sécurité mise en place. Le contenu de la requête initiale (en d'autres termes, le contenu dit applicatif) peut éventuellement avoir été modifié en par cette phase d'analyse. Une fois que ces opérations ont été effectuées, le coeur transmet le message dans le langage intermédiaire au guichet gauche.

Ce denier retranscrira le message en respectant la pile protocolaire initialement utilisée , afin de renvoyer la requête au serveur auquel elle était initialement destinée : c'est la **reconstruction** du flux réseau. A cette étape, et selon son mode

de fonctionnement, le module est aussi capable de modifier les métadonnées de la requête initiale. Dans le cas où il est placé comme un serveur mandataire, il modifie l'en-tête du paquet *Internet Protocol* (IP) en remplaçant l'adresse source du client par la sienne. Ainsi, le flux réseau entrant n'est pas nécessairement égal au flux sortant : ce phénomène est représenté par une requête sortante dénommée `REQUÊTE'` sur la figure 1, tandis que la requête initialement émise par le client est identifiée par `REQUÊTE`.

Avec un tel fonctionnement, la rupture protocolaire est directement induite, puisque la pile protocolaire est déconstruite puis reconstruite. La communication point-à-point entre un serveur et un client est effectivement subdivisée en deux communications point-à-point. Un premier gain en sécurité est donc offert, puisqu'à l'instar d'un pare-feu ou d'une diode, les attaques par le réseau sont rendues plus complexes. Cependant, la spécificité d'analyser le contenu du message par un langage intermédiaire, et d'éventuellement le modifier, constitue le second apport en sécurité de la rupture protocolaire. En effet, il est possible que l'analyse conduise à un arrêt de la communication entre C et S, s'il est considéré comme non conforme à la politique de sécurité.

2.2.2 • APPLICATION DE LA RUPTURE PROTOCOLAIRE AUX SCADA

Dans le cadre d'un dispositif SCADA visant à superviser une centrale électrique, les postes de contrôle-commande permettent de piloter des tensions. En pratique, le bon fonctionnement d'un circuit électrique est garanti selon une certaine plage de valeurs de tension, qui doivent donc être bornées.

Une configuration correcte des filtres consiste à analyser les requêtes visant à modifier cette tension. Par exemple, la règle suivante peut être définie : si une requête souhaite augmenter la tension d'un circuit et que sa valeur maximale est déjà atteinte, alors le module produit une alerte. La nature de cette dernière dépend de la politique de sécurité choisie : un message peut être envoyé aux postes de supervision, la requête peut être enregistrée dans un fichier de *log* (qui pourra être par la suite signé par le module en utilisant une clef privée dédiée). Un autre comportement pourrait être défini, en ramenant la valeur de tension à sa borne maximale (définie *a priori*). A terme, une configuration utilisant des filtres à états pourrait permettre de couper les communications, suite à un ensemble d'ordres jugés malicieux.

2.3 | ACTEURS INTERAGISSANT AVEC LE MODULE

La définition d'une architecture d'un module de rupture protocolaire est principalement orientée vers son fonctionnement. Cependant, avant de pouvoir être mis

en production, un équipement réseau doit être configuré. Cette étape est particulièrement importante, surtout pour la mise en place des clefs et certificats nécessaires à l'établissement de communications sécurisées.

Nous avons donc défini un cycle de vie du module, permettant de détailler les différentes étapes de configuration nécessaire de sa création à son utilisation, qui sont :

1. La fabrication du module ;
2. L'intégration du module dans une infrastructure ;
3. La mise en production ;
4. L'administration.

A chacune de ces étapes, différents acteurs vont interagir avec le module :

- **Le fabricant du module** vend et met à jour le module ;
- **L'intégrateur** s'occupe d'installer le module dans l'infrastructure du client. Il fournit aussi la première configuration, et peut éventuellement gérer une PKI pour le client ;
- **Le client** souhaite mettre en place le module dans son infrastructure ;
- **Le personnel de confiance** représente les personnes se connectant au module afin de réaliser des tacĥes d'administration telles que la configuration ou la mise à jour du boîtier ;
- **Les entités externes** sont celles qui communiquent avec le boîtier par le biais de la procédure de rupture de protocole, tels qu'un client et un serveur.

Les tâches administratives du boîtier requièrent des compétences et des droits spécifiques. Une hiérarchie au sein du personnel de confiance est donc nécessaire. Nous avons donc choisi d'implémenter cette hiérarchie avec un mécanisme de contrôle d'accès reposant sur les rôles (*Role-Based Access Control*) [SCFY96]. Nous définissons trois rôles :

- **Un super-administrateur** qui gère les tâches les plus critiques, telles qu'une mise à jour ;
- **Un administrateur** capable de mettre à jour le contenu du magasin d'ancres de confiance ;
- **Des techniciens** formés pour gérer la configuration du module.

2.4 | RÉPARTITION DES CLEFS ET CERTIFICATS

Par définition, un module de rupture protocolaire est un équipement voué à l'amélioration de la sécurité des réseaux. Comme nous l'avons évoqué précédemment, nous souhaitons que celui-ci s'insère dans des architectures de type contrôle-commande, ou plus généralement dans une architecture déjà déployée. En outre, l'analyse de flux éventuellement chiffrés rajoute une complexité supplémentaire lors de son intégration, puisqu'il est alors indispensable de prendre

l'architecture des clefs en compte. Comme précédemment, nous allons utiliser la configuration dans laquelle le module est mis en place entre un client et un serveur. Le modèle de gestion de clefs et certificats que nous présentons peut aisément se généraliser dans le cas où plusieurs entités sont réparties de part et d'autre du module. Dans cette section, nous détaillons les besoins en termes de clefs et de certificats d'un module de rupture protocolaire. La figure (2) synthétise l'organisation générale des PKI que nous définissons par la suite, et la table (1) résume les clefs présentes dans un module. La sécurité de notre architecture est analysée dans la section suivante (3).

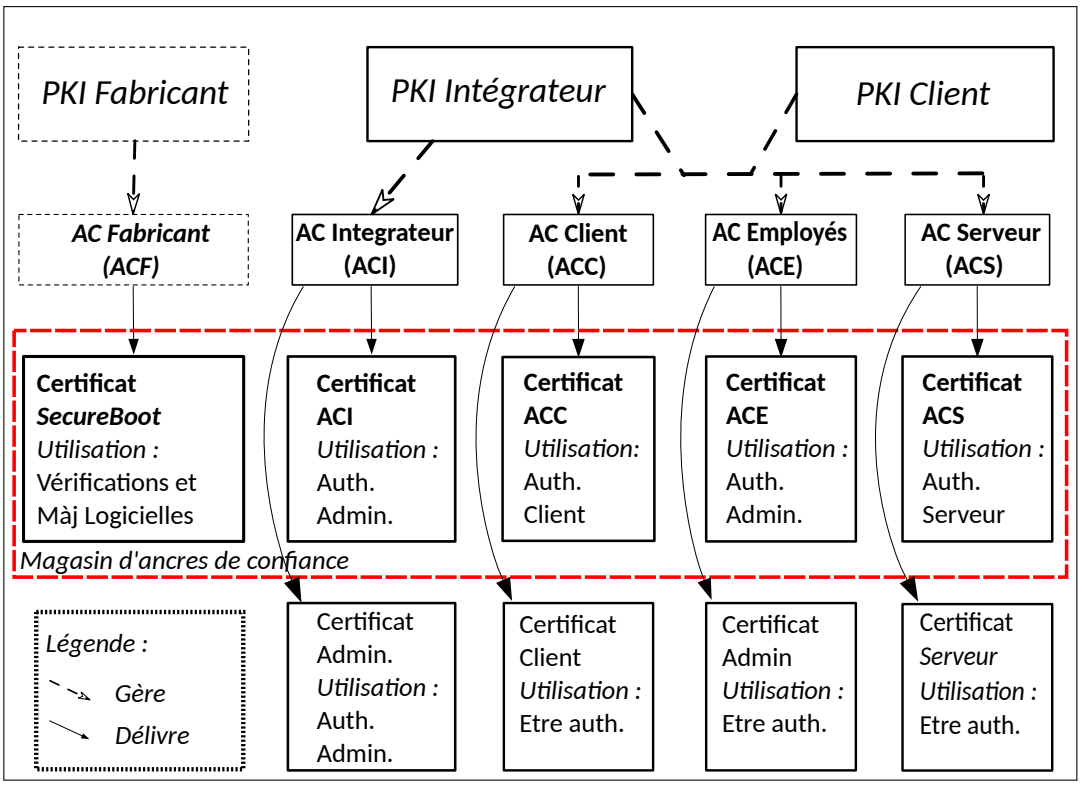

FIGURE 2 – Organisation générale des PKI, AC et Certificats

Dans un cas général, le module peut éventuellement être utilisé pour établir une connexion entre un client et un serveur n'appartenant pas au même site. Nous devons donc supposer que deux PKI distinctes peuvent gérer le client, et respectivement, le serveur. Afin de ne pas faire d'hypothèses sur l'organisation internes de ces PKI (voir le paragraphe (II.4.3.1)), nous les traitons de manière abstraite comme étant la PKI du réseau destinataire, que nous appelons la **PKI Client**. Nous identifions simplement les différentes PKI par leur autorité de certification. En pratique, les autorités de certification peuvent être racines, ou subalternes. En général, il

peut donc être nécessaire d'ajouter l'ensemble des certificats dans le magasin d'ancre de confiance pour assurer le bon fonctionnement : cette représentation est donc bien adaptée à notre problématique. Ainsi, nous obtenons donc celle du côté client (AC Client (ACC)) et celle du côté serveur par AC Serveur (ACS). L'authentification des employés peut être gérée par une AC différente, que nous dénotons AC Employés (ACE). La prise en compte des deux premières PKI est nécessaire pour permettre au module d'authentifier le client et le serveur avec lesquels ils communiquent. De plus, le module aura lui aussi besoin de paires de clefs asymétriques (dénotée **[K.4]**), et donc de certificats délivrés par l'AC correspondante, afin de s'authentifier auprès du client/serveur. Ils devront ensuite négocier des clefs symétriques, dites clefs de session (**[K.7]**), servant à chiffrer les communications. La troisième PKI permet au module d'authentifier le personnel souhaitant se connecter dessus afin d'y effectuer des tâches administratives. Parmi elles, la mise à jour du magasin d'ancres de confiance ou encore la configuration des règles de filtrage.

Un autre besoin concerne le fabricant, qui doit avoir la possibilité d'effectuer des mises à jour logicielles même une fois que le module a été installé. Il est essentiel qu'elles soient signées, pour éviter qu'un attaquant puisse modifier les programmes exécutés sur le module. Pour que le module soit à même de vérifier les signatures, il faut alors ajouter une nouvelle autorité de certification du fabricant, dénotée AC Fabricant (ACF).

Pour cela, il a besoin d'avoir une paire de clefs dédiée lui permettant de signer les différents logiciels nouvellement intégrés au module. Lors de son démarrage, le module vérifie l'intégrité des différents logiciels lancés. (comme le noyau de l'OS, voir le paragraphe sur l'implémentation d'un démarrage sécurisé 4.8). Or, lors d'une mise à jour, le programme étant modifié, il est nécessaire de réitérer le processus de signature. Afin que le module soit capable d'effectuer la vérification des différentes signatures, un certificat dénommé ***SecureBoot*** est placé dans le magasin d'ancres de confiance du module. Ce certificat peut éventuellement provenir de la PKI du fabricant, c.-à-d. qu'il représente le certificat d'une AC racine. Il peut aussi être vu comme certificat orphelin, *i.e.* il n'est pas associé à une véritable PKI (représenté par un cadre en pointillé dans la figure 2). Il est fourni par le fabricant et vu comme une ancre de confiance : il assure que seules des entités de confiance (c.-à-d. capables de réaliser une signature) sont en mesure d'effectuer des modifications logicielles sur le boîtier et donc leur intégrité. Notons qu'un démarrage sécurisé utilise une clef dite d'*ancre de confiance* (**[K.3]**), généralement stockée dans une mémoire immuable associée au matériel, et dont le but est d'assurer l'intégrité des premiers programmes lancés lors de la mise en route d'un module.

Enfin, l'intégrateur fournit aussi un paire de clefs permettant la connexion des personnes de confiance sur le boîtier. Ces derniers utilisent une authentification à

deux facteurs à l'aide d'une carte à puce. Afin que l'authentification reposant sur les certificats soit fonctionnelle, le certificat d'une ancre de confiance de l'intégrateur AC Intégrateur (ACI) est donc placé dans le magasin du module. Dans le cas où l'intégrateur gère la PKI du client, il place aussi tous les certificats racines nécessaires à l'authentification des entités communiquant avec le boîtier (soient les certificats des AC ayant signées les certificats des clients et serveurs). Les clefs publiques des ancres de confiance sont dénotées **[K.5]**.

2.5 | CYCLE DE VIE DU MODULE DES CERTIFICATS

Nous proposons le cycle de vie des certificats suivant, en adéquation avec celui du module. Ainsi, nous détaillons leur création et leur utilisation en fonction des quatre phases (la fabrication, l'intégration, la production et l'administration). Nous analyserons la sécurité afférente dans la section (3).

2.5.1 • FABRICATION

La fabrication du module est la première étape : elle consiste à physiquement réaliser les systèmes électroniques du module, ainsi qu'à fournir les briques logicielles nécessaires à sa future intégration. En termes de certificats, le fabricant fournit tout d'abord un certificat permettant la vérification de l'intégrité des programmes lancés par le module. Ce certificat peut éventuellement être délivré par l'autorité de certification du fabricant, ou vu comme un certificat auto-signé orphelin et placé comme ancre de confiance. Ce dernier est valide toute la durée de vie du module, mais peut être remplacé par le fabricant en cas de problème. Lors de son premier démarrage, le module doit être configuré pour pouvoir accepter des connexions externes provenant à la fois de clients et serveurs, mais aussi d'administrateurs. Ces derniers utilisent une authentification à double facteurs reposant sur l'utilisation de *smartcards*. L'administrateur prouve dans un premier temps qu'il est le porteur de carte légitime en entrant un code *Personal Identification Number* (PIN), puis un protocole d'authentification s'appuyant sur des certificats s'enclenche entre le module et la carte à puce. Afin que l'authentification puisse fonctionner, le certificat de l'ancre de confiance émettrice (*i.e.* le certificat ACE) doit être présent dans le magasin du module. Cependant, lors du premier démarrage, ce certificat n'est pas présent dans le module, mais la connexion de l'intégrateur doit être possible. Ce problème peut être simplement résolu par une pré-configuration du magasin réalisée lors de la fabrication du module. Dans ce cas, le certificat de l'ancre de confiance d'un administrateur temporaire est ajouté *a priori* dans le magasin et un certificat émis par ACI est alors stocké dans une carte servant au premier démarrage du module. Finalement, cette dernière contient une paire de clefs d'authentification à usage unique (dénotée **[K.2]**).

2.5.2 • INTÉGRATION

L'intégration est la phase de configuration permettant de déployer le module dans une infrastructure existante et de rendre l'ensemble fonctionnel. L'intégrateur récupère ainsi une carte lui permettant de se connecter au module en tant qu'administrateur temporaire. Il peut ensuite compléter le magasin en adéquation avec la PKI gérant les employés du client. Dans le cas où l'intégrateur fournit une PKI, il ajoute le certificat de l'autorité de certification associée, soit ACI, sinon il ajoute celui associé à la PKI du client (ACE). La seconde étape d'intégration du module consiste à ajouter l'ensemble des certificats nécessaires à l'authentification des clients et serveurs qui vont communiquer avec le module dans le magasin. Dans le cas où le client et le serveur ont des PKI distinctes, l'intégrateur ajoute donc les certificats ACC et ACS.

2.5.3 • PRODUCTION

Le module est en production lorsqu'il fournit les services le définissant dans l'infrastructure. Pendant la phase de production, le module aura de nouveaux besoins en termes de clefs : communication avec de nouvelles entités, renouvellement... Le module a alors la possibilité de générer lui-même les clefs, en utilisant son HSM. Il est alors nécessaire d'associer un certificat à ces clefs, afin que le module puisse s'authentifier auprès des entités externes. Après la génération des clefs, le module peut générer la partie *To Be Signed* de son certificat *X.509*. Il faut alors transmettre ces informations au CA afin qu'il y appose sa signature. Il est aussi possible que le CA génère lui-même les clefs ainsi que le certificat associé. Il suffit alors à un administrateur d'adjoindre l'ensemble au contenu du magasin d'ancres de confiance. De plus, le module peut aussi envoyer des informations, comme des *logs* ou des alarmes en cas de détection de comportements suspects. Pour cela, il dispose d'une paire de clefs dédiée, dénommée **[K.6]** lui permettant de les signer avant de les transmettre.

2.5.4 • ADMINISTRATION

L'administration du boîtier se compose de plusieurs tâches. Comme évoqué précédemment, il est nécessaire de mettre à jour les différents composants logiciels du boîtier. L'objectif est de réduire les possibilités d'attaques exploitant des failles découvertes sur d'anciennes versions des programmes exécutés par le module (comme le systèmes d'exploitation). Ces mises à jours sont effectuées par un administrateur du module, dont le certificat à été émis soit par le CA du client, de l'intégrateur ou du fabricant. Il est cependant possible, selon la politique de sécurité, d'exiger que les mises à jour soient signées par le fabricant afin de limiter les possibilités de corruption du boîtier.

L'administration du boîtier consiste aussi à gérer le contenu du magasin d'ancres de confiance. S'il faut ajouter des certificats d'autorités de certification pour assurer le bon fonctionnement du boîtier, il est aussi nécessaire de s'occuper du cycle de vie des clefs et certificats, c.-à-d. le renouvellement et la révocation. Comme nous l'avons vu précédemment (dans le paragraphe II.4.2), le statut d'un certificat est vérifiable de deux manières : soit en utilisant le protocole OCSP, soit en utilisant des CRL. Le protocole OCSP a l'avantage de reposer sur un gestionnaire externe (le répondeur OCSP) connaissant un statut actualisé des certificats. En contrepartie, le module doit être connecté au répondeur, afin de pouvoir lui envoyer les requêtes lors la vérification des certificats. Il peut éventuellement utiliser *l'agrafage OCSP* (*OCSP Pining*) pour éviter ces communications. La seconde solution, utilisant les CRL, ne requiert pas de communication externe pour vérifier l'état des certificats : elle repose sur l'utilisation d'une liste stockée dans le magasin d'ancres de confiance et contenant les certificats révoqués. Il est alors nécessaire qu'un administrateur mette régulièrement à jour cette liste en suivant les consignes définies dans la DPC et la PC, pour éviter que le module n'accepte des certificats récemment révoqués.

Enfin, une partie de l'administration du boîtier repose sur la configuration des filtres. Ces derniers sont dépendants de l'environnement du module, et doivent être adaptés selon la politique de sécurité du réseau. Cette tâche est déléguée aux personnels de confiance du client. Une fois terminée, la configuration peut éventuellement être chiffrée (via une clef symétrique dédiée [**K.1**]), pour être exportée afin d'être sauvegardée et permettre une restauration rapide du module en cas de problème.

Caractéristiques / *Nom des clefs*	Sym.	Asym.		Lieu de stockage	Créateur
		Pub.	Privée		
[**K.1**] Clef de chiffrement de la configuration	✓			HSM	Administrateur
[**K.2**] Clef d'authentification à usage unique		✓	✓	HSM	Intégrateur
[**K.3**] Clef "Racine de confiance"	✓			Mémoire dédiée immuable	Fabricant
[**K.4**] Clefs de chiffrement des communications		✓	✓	HSM	Guichets
[**K.5**] Clefs des Ancres de confiance		✓		HSM	AC
[**K.6**] Clefs du coeur		✓	✓	HSM	Coeur
[**K.7**] Clefs de session	✓			Guichets	Guichets, Entités externes

TABLE 1 – Récapitulatif des clefs présentes dans un module de rupture

3 ■ ANALYSE DE SÉCURITÉ

Nous souhaitons maintenant démontrer que l'architecture présentée offre un niveau de sécurité suffisant. A l'instar des profils de protection (*Protection Profile*), nous commençons par décrire un ensemble de propriétés de sécurité qu'un module de rupture de protocole devrait vérifier. Nous montrons ensuite comment l'architecture proposée permet de répondre à ces problématiques.

3.1 | PROPRIÉTÉS DE SÉCURITÉ

Dans le but de déterminer les propriétés de sécurité en adéquation avec un module de rupture protocolaire, nous nous inspirons de cibles de sécurité réalisées à destination des pare-feu [ANS15] ou NGFW [Gos16] En effet, comme certains NGFW sont capables d'effectuer de l'analyse de protocoles chiffrés, et donc de la rupture protocolaire, un module en rupture possède donc des caractéristiques communes. En particulier, il est possible d'extraire les propriétés suivantes : *support cryptographique*, *accès sécurisé au module*, *authentification et identification* et *protection interne du module*. Les propriétés spécifiques aux NGFW, en rapport avec les protocoles qu'ils doivent être capables d'analyser et sur la manière de les analyser, ont été écartées. En contrepartie, une nouvelle propriété dénommée *Isolation physique et logique* est définie.

3.1.1 • SUPPORT CRYPTOGRAPHIQUE

Le support cryptographique [ANS14, BBB+07] comprend l'ensemble des opérations cryptographiques que le module doit être capable de gérer :

- **SC.1** Génération de clefs (symétriques et asymétriques) ;
- **SC.2** Chiffrement, déchiffrement ;
- **SC.3** Signature ;
- **SC.4** Hachage ;
- **SC.5** Générateur d'aléa.

Ces propriétés sont nécessaires pour assurer des échanges sécurisés. Des propriétés plus générales doivent donc être vérifiées, notamment sur l'intégrité et la confidentialité. La première assure que les données n'ont pas été altérées. Dans le cas du module, les données qu'il faut protéger sont donc :

- **INT.1** Les *firmwares* ;
- **INT.2** Les fichiers de configuration ;
- **INT.3** Les logs ;
- **INT.4** Les alarmes ;

La confidentialité certifie que seules les personnes autorisées ont accès aux informations qui leur sont destinées. Cette propriété doit plus spécifiquement

s'appliquer au module sur :

- **CONF.1** Le stockage des données sensibles ;
- **CONF.2** Les fichiers de configuration.

3.1.2 • ROBUSTESSE

La robustesse du module comprend plusieurs critères. Elle concerne tout d'abord sa protection face aux éventuelles attaques physiques : un adversaire ne doit pas être en mesure de retrouver des informations sensibles (telles que les clefs privées) en ouvrant le boîtier. La robustesse est aussi en lien avec les capacités de filtrage déployées sur le module. A l'instar d'un NGFW, il doit être capable de détecter des paquets malformés, de vérifier la conformité de l'exécution d'un protocole ou encore d'exécuter des actions selon une politique de sécurité prédéfinie. Nous ne détaillerons pas les propriétés sur les filtres pour nous concentrer sur celles en lien avec l'architecture du module. Finalement, nous définissons donc la propriété suivante :

- **ROB.1** : Détection et protection des données face aux attaques physiques ;
- **ROB.2** : Remise à zéro (*Zéroisation*) [oST].

3.1.3 • AUTHENTIFICATION ET IDENTIFICATION

L'identification consiste à associer une identité à une entité, tandis que l'objectif de l'authentification est de prouver qu'une entité est bien celle qu'elle prétend être. Dans le cadre du module, il est nécessaire de contrôler l'identité des différents éléments communiquant avec ce dernier :

- **AUT.1** Connexions sécurisées avec des clients/serveurs externes ;
- **AUT.2** Authentification du personnel se connectant au module ;
- **AUT.3** Politique de contrôle d'accès.

L'authentification d'un utilisateur est nécessaire en amont de sa connexion au module. Elle consiste à apporter une preuve de l'identité de l'utilisateur se connectant au module. L'identité de la personne est associée à un rôle, lui conférant des droits différents.

3.1.4 • ISOLATION PHYSIQUE ET LOGIQUE

L'isolation consiste à transformer un flux, entre au moins deux entités, en deux flux distincts. En d'autres termes, le module doit séparer entre les périphériques connectés de part et d'autre. L'objectif de l'isolation est d'empêcher la propagation d'attaques. D'un point de vue logique, cela implique qu'un flux entrant n'est pas nécessairement transféré de part et d'autre du module. Physiquement, cela signifie qu'en cas de compromission d'une partie du module, l'ensemble du module ne doit

pas être compromis. En résumé, un module est capable d'isoler des réseaux s'il sépare physiquement et logiquement :

- **ISO.1** Les espaces mémoires ;
- **ISO.2** Les périphériques ;
- **ISO.3** Les unités de calculs ;
- **ISO.4** Les communications réseaux.

3.2 | VÉRIFICATION DES PROPRIÉTÉS

Dans la suite, nous montrons comment l'architecture proposée permet de vérifier les propriétés précédemment énoncées.

3.2.1 • SUPPORT CRYPTOGRAPHIQUE

L'ensemble des propriétés en lien avec le support cryptographique sont réalisées par le HSM présent sur la partie centrale. En d'autres termes, le HSM doit être capable de : générer des clefs (**SC.1**), chiffrer/déchiffrer (**SC.2**) signer (**SC.3**), hacher (**SC.4**), et générer de l'aléa (**SC.5**). Les guichets peuvent éventuellement être capables d'effectuer des opérations cryptographiques symétriques.

L'intégrité des *firmwares* est validée par la configuration initiale du boîtier : ils sont signés par une paire de clefs dont son certificat (le certificat dénommé *SecureBoot* dans la figure 2), et celui de l'AC l'ayant délivrée, sont présents dans le magasin d'ancres de confiance. Avant de lancer un programme, sa signature est vérifiée en amont, et seules les entités ayant les droits nécessaires sont à même d'effectuer une nouvelle signature d'un programme, ce qui vérifie la propriété (**INT.1**).

Dans le même ordre d'idée, les fichiers de configuration sont chiffrés à l'aide d'une clef symétrique (validant **CONF.2**), générée ou fournie par un administrateur. Aucun fichier de configuration ne peut sortir du module en clair, et l'accès à la clef ne peut se faire qu'après l'authentification de la personne. Après chaque modification, un mécanisme de signature symétrique de type *keyed-Hash Message Authentication Code* (HMAC) est utilisée, assurant leur intégrité (**INT.2**).

Pour rappel, la partie centrale du module est aussi en charge de filtrer les données. Par définition, c'est donc au niveau du coeur que les événements potentiellement malicieux seront majoritairement détectés. Le coeur a alors la possibilité de demander au HSM d'effectuer la signature d'un fichier de logs, ou encore de signer une requête déclenchant une alarme. Ces signatures permettent d'éviter que les fichiers de logs ne soient modifiés (par exemple pour supprimer les traces d'une attaque éventuelle), ou encore que de fausses alarmes soient déclenchées. Ces fichiers peuvent ensuite être transmis à un *Security Information and Event*

Management (SIEM), qui pourra vérifier qu'aucune entité autre que le module ne les a modifiées. Ce mécanisme permet d'approuver **INT.3** et **INT.4**

3.2.2 • AUTHENTIFICATION ET IDENTIFICATION

Dans l'architecture proposée, la connexion au module se fait en passant par le coeur. La solution proposée, permettant l'authentification à deux facteurs des usagers, repose sur l'utilisation de cartes à puce. Une première authentification est faite en demandant à l'utilisateur de rentrer un code PIN (démontrant qu'il est le propriétaire de la carte). Le second facteur provient d'un certificat stocké sur la carte, permettant à une personne d'authentifier à l'aide d'un protocole interactif (comme le *Simple Authentication and Security Layer* (SASL) [ZM06]) sur le module. Une fois encore, en respectant les préconisations de configuration, le certificat de l'autorité de certification de la PKI des employés sera présent dans le magasin, assurant la propriété (**AUT.2**). De plus, grâce au certificat fourni par la carte, le module est alors en mesure d'identifier l'utilisateur, de lui attribuer un rôle, et donc des droits associés à son statut (**AUT.3**). L'authentification des entités externes (**AUT.1**) se réalise principalement à l'aide des *poignées de main* (*handshake*), présentes sur les protocoles sécurisés reposant sur TLS. Les guichets redirigent les requêtes demandant l'authentification via un certificat du client/serveur, et ce dernier les retransmet au HSM. Grâce à l'étape d'intégration, les ancres de confiance associées au client/serveur permettent de vérifier la chaîne de certification, et donc de les authentifier. Par la suite, les échanges sont généralement sécurisés via des mécanismes de cryptographie symétrique, gérés par le HSM ou les guichets.

3.2.3 • PROTECTION INTERNE DU MODULE

Afin de protéger le module face aux attaques physiques, un mécanisme de détection d'intrusion doit être mis en place sur le module (**ROB.1**). Une manière plus sûre consiste à en déployer deux : un premier pour détecter une intrusion sur le module, et un second sur le HSM. Ainsi, le module se verrouille dans un premier temps, et les informations sensibles stockées sur chaque partie sont supprimées. Ensuite, HSM met à zéro l'ensemble de sa configuration enregistrée dans son coffre-fort en cas d'intrusion (**ROB.2**). Pour renforcer le module face à ce type d'attaque, il est aussi possible de chiffrer toutes les communications internes *i.e.* entre le coeur et les deux guichets. Les guichets étant capables d'effectuer du chiffrement symétrique (en utilisant une unité de calcul dédiée) une clef symétrique peut être stockée dans la mémoire dédiée ainsi que dans le HSM présent sur le coeur. En utilisant un mécanisme de dérivation de clef, les guichets et le coeur sont ainsi capables de chiffrer tous les échanges.

Isolation physique et logique Grâce à l'architecture en trois parties, le module en rupture est capable de fournir de l'isolation à plusieurs niveaux. En effet, le module est composé de trois systèmes autonomes, chacun possédant un processeur et une mémoire dédiés. Ainsi, intrinsèquement à cette architecture, l'isolation en termes de calculs et de mémoire est donc vérifiée (**ISO.1**, **ISO.2**, **ISO.3**). Il existe cependant un canal de communication entre le guichet gauche et le coeur, et un autre entre le coeur et le guichet droit. Ces canaux sont séparés d'un point de vue logiciel, puisqu'aucune des données provenant d'un des guichets ne peut être retransmise directement sur l'autre sans avoir été traitée par le coeur. Tous les flux réseaux arrivant sur le module sont déconstruits pour pouvoir être analysés, puis éventuellement reconstruits. Il est possible de configurer le module selon deux modes : en tant que serveur mandataire, ou directement relié au client. Le fonctionnement ainsi que la démonstration de l'isolation produite en fonction des couches du modèles OSI sont détaillés dans le tableau (2), validant par conséquent la propriété (**ISO.4**).

Mode / *Couche*	***Direct***	***Serveur Mandataire***
Transport	Une seule session est établie entre le serveur et le module	Deux sessions distinctes sont établies : la première entre le client et le module, et la seconde entre le module et le serveur.
Réseau	L'équivalent d'un **pont réseau** est réalisé entre le client et le module *i.e.* toutes les trames échangées par le client sont redirigées en direction du module	Le client est configuré pour communiquer avec le module, qui possède sa propre adresse IP. Le paquet est alors modifié pour émettre en utilisant comme adresse source celle du module. Une table d'équivalence est maintenue pour que les réponses soient redirigées vers le bon client
Liaison	Le client et le serveur possèdent chacun une adresse *MAC* distincte. Comme le module s'interface entre les deux en utilisant deux systèmes distincts (les guichets gauche et droit), il possède donc deux adresses MAC associées à chacun des guichets	
Physique	Le lien physique entre le client et le serveur n'est plus direct, puisque le module est ajouté entre les deux	

TABLE 2 – Description de la rupture protocolaire selon les couches du modèle OSI en fonction du mode d'utilisation du module

4 ■ ARAMIS : IMPLEMENTATION D'UN MODULE

Dans le cadre du projet ARAMIS, le consortium (présenté en 4.1) a développé une preuve de concept d'un module de rupture protocolaire. Le module ARAMIS est une implémentation d'un module de rupture protocolaire dédiée à l'amélioration de la sécurité des infrastructures de contrôle-commande.

4.1 | PROJET ARAMIS

Le projet ARAMIS, financé par la *Banque Publique d'Investissement*, a pour but de développer un module de rupture protocolaire améliorant la sécurité des communications des infrastructures industrielles. Ce projet est réalisé par un consortium composé de *ATOS Worldrid*, de *Seclab*, du *Commissariat à l'énergie atomique et aux énergies alternatives* et de l'*Université Grenoble-Alpes*. Plusieurs laboratoires de l'*UGA* sont impliqués dans le projet : l'*Institut Fourier*, *Verimag*, le *Laboratoire d'Informatique de Grenoble*, et le *Laboratoire Jean Kuntzmann*.

Le module ARAMIS permet de protéger les systèmes industriels en isolant les entités communiquant de part et d'autre du module. En particulier, cette isolation s'applique sur les communications entre les systèmes de contrôle-commande et les réseaux externes, ainsi qu'entre les communications internes au système. Le module peut effectuer des opérations de filtrage des données échangées sur des canaux de communication garantissant la confidentialité, l'intégrité et l'authentification des entités. De plus, il est aussi en mesure de vérifier des contraintes fortes sur la durée de traitement des données, afin de ne pas affecter le fonctionnement et la sûreté des infrastructures mises en place. La figure (3) est une photo d'un module ARAMIS.

FIGURE 3 – Photo du module ARAMIS

Par la suite, nous détaillons l'architecture matérielle et logicielle du module, avant de nous concentrer sur la gestion des clefs et des certificats.

4.2 | ARCHITECTURE MATÉRIELLE

L'architecture matérielle du module ARAMIS respecte l'architecture tripartie présentée au paragraphe (2.1). Elles sont chacune composés d'un processeur ARM associé à des *Field-Programmable Gate Array* (FPGA). Ce type de processeur permet de supporter un système d'exploitation classique, tandis que le FPGA est dédié aux opérations cryptographiques. Le module dispose uniquement d'interfaces réseaux filaires (de type Ethernet) évitant les attaques via les réseaux sans-fils. Un port USB est présent sur le coeur, permettant de connecter un lecteur de cartes muni d'un clavier. Un utilisateur souhaitant administrer le module insère sa carte dans ce lecteur, et tape son code PIN pour déverrouiller la carte et amorcer le protocole d'authentification utilisant le certificat présent sur la carte.

4.3 | ARCHITECTURE LOGICIELLE

Afin de limiter les vecteurs d'attaque logiciels, le module embarque sur ces guichets un système d'exploitation reposant sur un noyau sécurisé [KEH+09], permettant d'obtenir un environnement d'exécution proche d'un *Trusted Execution Environment* (TEE). Le coeur étant en charge des tâches critiques (appels aux routines cryptographiques du HSM, fonctions d'administration, filtrage des données), nous avons opté pour un système d'exploitation reposant sur un micro noyau sécurisé. Ce dernier n'implémente que les fonctions nécessaires au bon fonctionnement du coeur, et permet de limiter au maximum les failles logicielles.

Au niveau des FPGA, une bibliothèque a été spécifiquement développée pour répondre aux contraintes relatives au temps d'exécution. Elle implémente la cryptographie sur courbe elliptique en s'inspirant des travaux [Cor16]. Elle respecte évidemment les recommandations usuelles (telle que [ANS14]) concernant les algorithmes ainsi que les tailles de clefs à employer.

4.4 | ARCHITECTURE DU HSM

Le HSM relié au coeur du module ARAMIS implémente les fonctionnalités liées à la sécurité du module. Il fait office de coffre-fort pour les informations sensibles (comme les clefs ou les certificats), et réalise des opérations cryptographiques en lien avec ces dernières (comme les signatures ou du déchiffrement). Le HSM est robuste face aux attaques physiques (*tamper resistant*) et fournit des preuves d'effraction (*tamper evidence*).

De manière générale, un HSM embarqué a pour objectif de fournir un démarrage sécurisé ainsi que des applications de confiance. Dans le cas du module ARAMIS, les contraintes du HSM sont les suivantes :

- Démarrage sécurisé ;
- Stockage sécurisé ;
- Coûts de fabrication faible (moins d'une centaine d'euros) ;
- Consommation réduite ;
- Robustesse face aux attaques ;
- Opérations cryptographiques efficaces, permettant de répondre aux besoins d'un temps de réponse proche du temps réel (de l'ordre de la milliseconde) d'un système industriel.

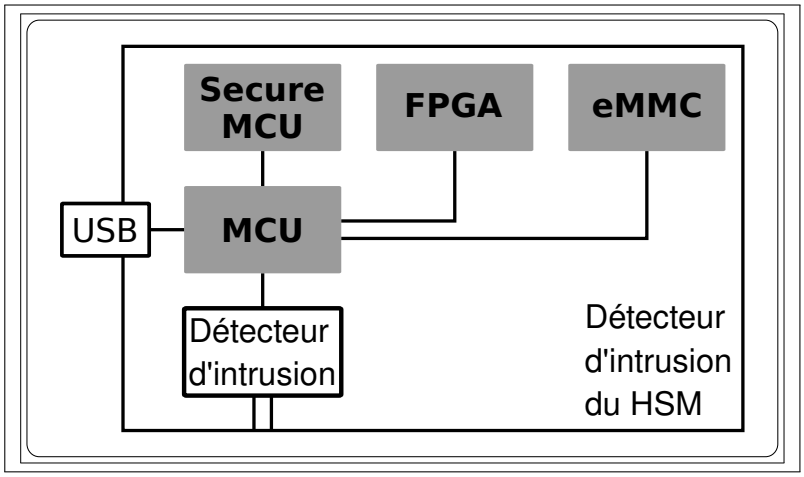

FIGURE 4 – Architecture matérielle du HSM

L'architecture matérielle du HSM (voir la figure 4) repose sur quatre composants principaux : deux microcontrôleurs (*MicroController Unit* (MCU)), un FPGA et une carte mémoire multimédia (*embedded MultiMedia Card* (eMMC)). Le premier MCU est utilisé pour coordonner l'ensemble des opérations. Il reçoit des instructions venant du coeur, puis les répartit aux autres composants selon leur nature, et renvoie finalement le résultat de l'exécution vers le coeur. Les requêtes ayant un rapport avec des opérations cryptographiques symétriques sont redirigées en flux vers le FPGA. Ce dernier permet d'effectuer ce type d'opérations efficacement, c.-à-d. avec un débit élevé et une latence faible. La cryptographie asymétrique est implémentée dans le second MCU, qui est sécurisé. En effet, comme ce composant fait aussi office de coffre-fort pour les clefs maîtres, il est donc protégé contre les attaques physiques. La carte mémoire est utilisée comme mémoire non-volatile du HSM. Toutes les informations stockées sur la carte sont chiffrées en amont, elle ne contient donc pas d'information en clair. Un détecteur d'ouverture du HSM est aussi connecté aux composants : en cas d'attaque physique, le contenu du microcontrôleur et du FPGA

est supprimé, et tous les composants sont éteints. Le contenu du MCU sécurisé est quant à lui protégé par un détecteur d'effraction dédié.

Le HSM est connecté au coeur via un port USB. La communication avec le HSM se fait par le biais de l'interface *PKCS#11* [RSA14]. Les commandes provenant du coeur sont *wrappées* puis serializées avant d'être envoyées au HSM. L'opération duale est ensuite effectuée par le MCU interne au HSM, puis la commande est exécutée par le MCU, ou redirigée vers la bonne unité de calcul si nécessaire (MCU sécurisé, FPGA).

Une des particularités du HSM réside dans l'implémentation de la gestion des rôles en utilisant *PKCS#11*. Par défaut, deux rôles seulement sont pris en compte : un responsable de la sécurité (*Security Officer* (SO)) et les autres utilisateurs (*Users*). Le SO gère les utilisateurs, en leur attribuant un code PIN, et a accès à l'ensemble des données de chacun. Un utilisateur peut stocker des objets (*e.g.*, des clefs) et demander au HSM de réaliser des opérations cryptographiques les utilisant. L'administration du module nécessite une gestion plus fine des droits où tous les utilisateurs ne peuvent pas être vus comme une seule et même entité. Le problème majeur d'une telle gestion concerne l'accès aux objets contenus dans le HSM : comme toutes les entités connectées (exceptées le SO) pointent vers le rôle *User*, cela implique que toutes les entités ont accès à tous les objets. L'extension développée permet de gérer chaque utilisateur comme une entité distincte possédant ses objets, et n'ayant pas accès aux objets privés (comme les clefs privées) des autres utilisateurs. Seul le SO conserve ce droit, comme défini originellement.

Au niveau de l'implémentation, l'idée est d'adapter la fonction de connexion `C_Login`, tout en restant compatible avec le standard *PKCS#11*. L'authentification à un *token* via *PKCS#11* se fait en entrant le code PIN de l'utilisateur, et en précisant via un *flag* si l'utilisateur souhaite se connecter en tant que responsable de sécurité (le PIN demandé est alors celui associé au SO). Dans le cadre du HSM présenté, lorsqu'un utilisateur souhaite se connecter, il ajoute un identifiant (dénommé ***login***) avant de rentrer son PIN, selon la syntaxe suivante :

```
login:PIN
```

Dans le cas où seul un PIN est rentré (et est correct), il est associé à un utilisateur *main* tel que défini dans le standard, sans avoir les droits d'accès aux objets privés qui ne lui appartiennent pas.

4.5 | CONNEXION AU HSM

Dans le module, le HSM est nécessaire à deux tâches primordiales : l'administration du boîtier et les opérations cryptographiques internes. La configuration du HSM (*i.e.* l'ajout des utilisateurs, clefs et certificats) est réalisée via le coeur

et l'interface *PKCS#11* grâce aux lignes de commandes données par *pkcs11-tools* (compris dans l'ensemble des bibliothèques *OpenSC* [Kir03]). Afin de faciliter l'utilisation des commandes *PKCS#11*, des scripts fournissant une IHM de haut niveau ont été développés. En particulier, toutes les opérations d'administration ont été simplifiées, en combinant les commandes de *pkcs11-tools* normalement nécessaires. Ainsi, les paramètres requis sont demandés explicitement, et peuvent être saisis par l'utilisateur en un bloc, soit de manière interactive.

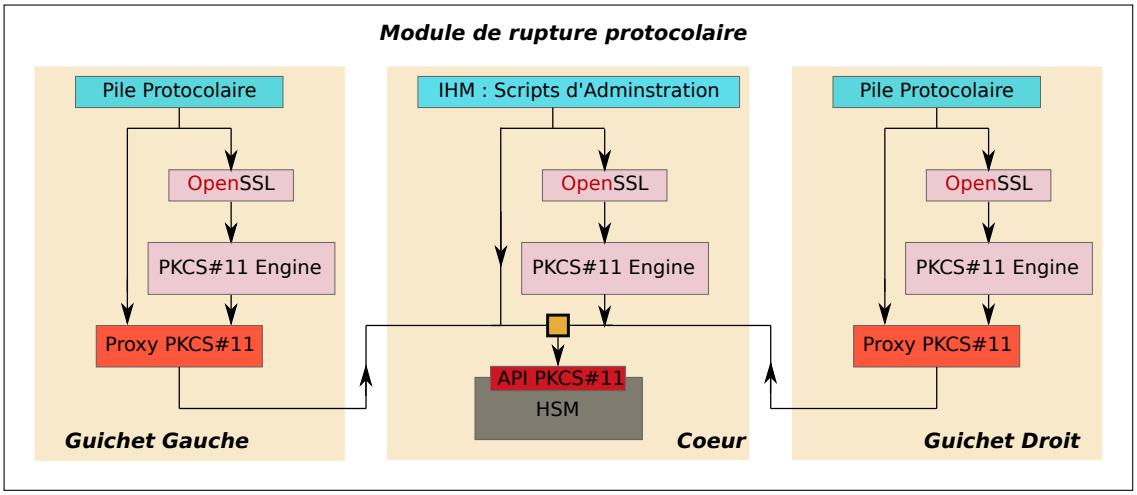

FIGURE 5 – Architecture logicielle déployée pour communiquer avec le HSM

4.6 | ETABLISSEMENT D'UNE CONNEXION SÉCURISÉE

Généralement, l'établissement d'une communication sécurisée (via un protocole comme *File Transfert Protocol Secure* (FTPS) ou *OPC Unified Architecture* (OPC-UA)) repose sur une phase dite de **poignée de main** (*handshake*). Elle permet aux deux entités de s'accorder sur les paramètres de sécurité à utiliser par la suite, et fait appel aux clefs asymétriques des participants pour procéder à la mise en place d'une clef de session commune. Lorsque le module est placé entre le client et le serveur, il doit donc effectuer deux *handshakes* : un premier entre le client et le guichet droit, et un second entre le serveur et le guichet gauche. Les guichets doivent impérativement avoir accès aux clefs asymétriques, et donc au HSM. Pour répondre à ce problème, l'idée est d'enregistrer le coeur et les guichets comme des utilisateurs du HSM. Ainsi, les guichets s'authentifient auprès du HSM, et ont ensuite accès à ces objets. Grâce à la gestion des rôles, le coeur et chacun des guichets ont accès uniquement à leurs objets. Enfin, les guichets n'ayant pas un accès direct

au HSM, une interface spécifique, appelée ***proxy PKCS#11*** est mise en place. Elle permet de transmettre les requêtes d'un guichet au HSM en passant par le canal de communication existant entre le guichet et le coeur. La pile protocolaire de chacun des guichets est configurée pour faire appel à *OpenSSL* pour effectuer les opérations cryptographiques. Ainsi, *OpenSSL* est utilisée comme une *Application Programming Interface* (API) faisant appel au HSM pour réaliser physiquement les opérations. Le lien entre l'*API PKCS#11* et les fonctions *OpenSSL* est un moteur *PKCS#11* (***PKCS#11 engine***) [Kir03]. Ce dernier redirige l'ensemble des appels fait à *OpenSSL* concernant la cryptographie asymétrique vers des appels à des fonctions de l'*API PKCS#11*. Grâce à tous ces mécanismes, les guichets peuvent effectuer l'ensemble des opérations nécessaires à la réalisation du *handshake*.

4.7 | GESTION DES CLEFS ET CERTIFICATS

La gestion des clefs et des certificats dans le cadre du projet ARAMIS reprend l'architecture décrite dans 2. Ainsi, le module est capable de s'intégrer dans une infrastructure de clefs complexe, où les serveurs, clients et employés peuvent être gérés par des PKI distinctes. Le magasin d'ancre de confiance doit donc être rempli *a priori* pour contenir tous les certificats nécessaires à l'authentification des entités externes communiquant avec le module.

Le nombre de paires de clefs contenues dans le module dépend du nombre d'entités avec lesquelles ce dernier rentre en communication. Ainsi, une paire de clef peut être générée par protocole, ou par adresse IP. Par exemple, une paire de clef dédiée à l'utilisation d'OPC-UA et une autre pour FTPS. Autrement, une paire de clefs est associée à un client, identifié par son adresse IP. Dans tous les cas, afin que le client (et de manière symétrique, le serveur) puisse identifier le module, toutes les paires de clefs doivent être associées à un certificat émis par l'AC associée, c.-à-d. AC Client ou AC Serveur (cf. 2). Dans le cas où la PKI est fournie par l'intégrateur, sa mise en place (c.-à-d. la gestion des autorités de certification) repose sur trois documents que nous avons rédigés : la *Politique de Certification*, la *Déclaration des Pratiques de Certification* et la *Procédure de renouvellement des clefs*.

La configuration du module peut être chiffrée en utilisant une clef symétrique générée par le module. Ainsi, elle peut être exportée et stockée dans une base de données quelconque. En cas de problème, elle peut être remise sur le module qui sera capable de la déchiffrer. De plus, l'utilisation d'un protocole de type HMAC permet au module de vérifier l'intégrité des données de configuration qu'il reçoit. En exportant la clef symétrique de manière sécurisée (par exemple en la chiffrant avec la clef publique d'un administrateur), une restauration complète du module est possible. Cette pratique nécessite plus de précaution, puisqu'il faut aussi stocker la clef symétrique, mais permet une restauration totale de la configuration du module

(lors d'une panne par exemple).

Enfin, la connexion au module par les employés utilise un double facteur d'authentification reposant sur l'utilisation des cartes à puce. Pour se connecter, l'utilisateur présente sa carte dans un lecteur spécifique relié au module, au niveau du coeur. Une première authentification est faite par la carte envers l'utilisateur via le code PIN. Ensuite, le module authentifie la carte à l'aide du certificat qu'elle contient. A ce stade, l'utilisateur est donc connecté sur le module, c.-à-d. qu'il est en interaction avec l'*Operating System* (OS) déployé sur le coeur par le biais d'un terminal. Si l'utilisateur souhaite effectuer des tâche d'administration nécessitant un accès ou une modification du contenu du HSM, il doit ensuite s'authentifier auprès du HSM par le biais des *Scripts d'Administration* (*cf.* la procédure décrite précédemment (4.4)).

4.8 | DÉMARRAGE SÉCURISÉ

Le démarrage sécurisé du module (en anglais *Secure Boot*) consiste à vérifier que l'ensemble des logiciels chargés en mémoire sont intègres. En d'autres termes, c'est le moyen de s'assurer que les programmes lancés n'ont pas été modifiés par un adversaire. L'architecture du module ARAMIS (voir 4.2) repose sur trois systèmes indépendants, constitués d'un processeur ARM associé à des FPGA. Lors du démarrage, les FPGA sont alimentés en premier, et charge le premier *bitstream* contenant les informations nécessaires aux configurations matérielles du FPGA. L'autre partie du *bitstream* indique au FPGA l'emplacement mémoire du prochain programme à charger, c'est le *pre-loader*. Ce dernier permet de configurer à bas niveau les autres composants de la carte, telle que la fréquence des différentes horloges présentes (par exemple celles du processeur et de la mémoire). Le *pre-loader* indique ensuite l'emplacement du *UBoot* (*Universal Bootloader* [UB13]). Ce dernier est utilisé afin de charger le noyau (*kernel*) en mémoire, suivi du montage du système de fichier (via *rootfs*.)

La sécurité du démarrage du système repose sur les vérifications effectuées par chacun des programmes sur le prochain, et sur la confiance placée sur l'origine de cette chaîne de vérification : le fondeur du FPGA. En effet, lors du chargement du *bitstream* par le FPGA, ce dernier va utiliser une clef de chiffrement symétrique, chargée de manière immuable dans un espace mémoire dédié. Le chargement du *bitstream*, et donc de cette clef pour la vérification du *pre-loader*, constitue donc la racine de confiance de la chaîne de vérification (*root of trust* [Wil14]). Cette méthode de démarrage permet donc de s'assurer que l'ensemble des programmes chargés en mémoire sont intègres, à l'exception de la racine. La protection face aux attaques physiques du module est donc essentielle, dans le but de ce prémunir face à des attaques récentes, telle que [JHZ+17]. L'implémentation du démarrage sécurisée est plus amplement détaillée dans [RBBT17].

5 ■ CONCLUSION

Dans ce chapitre, nous avons exposé une architecture de sécurité d'un module de rupture de protocole, particulièrement adapté aux architectures de contrôle-commande industrielles. Sa conception en trois parties distinctes lui permet d'associer des tâches spécifiques à chacune d'entre elles : la réception et déconstruction du flux réseau par un des guichets, l'analyse et les opérations cryptographiques par le coeur, puis la reconstruction et l'envoi du flux par l'autre guichet. Grâce à ce processus, le module est capable de vérifier des propriétés de sécurité proches d'un *Next Generation FireWall*, comme le support cryptographique, l'authentification et l'identification ainsi qu'une protection interne suffisante pour protéger les informations les plus sensibles. *A contrario* des NGFW, un module de rupture isole deux réseaux, de manière physique et logique. Il permet en outre d'effectuer de l'analyse de flux chiffrés en respectant des contraintes sur le temps de réponse (proche du temps-réel), et peut aussi être utilisé afin de sécuriser des canaux de communication non chiffrés.

Dans le cas où le module intervient dans une communication qui était déjà chiffrée, il maniera des données en clair qui sont normalement chiffrées. Le module ayant pour objectif l'analyse des flux chiffrés, le déchiffrement des données est intrinsèque à son utilisation. Cependant, si ce dernier est attaqué, des données pourraient éventuellement être récupérées.

L'architecture de sécurité détaillée précédemment présente l'avantage d'être aussi générique que possible. Les hypothèses utilisées sur les PKI permettent non seulement au module de s'intercaler entre des canaux de communications existants, mais aussi d'être placé dans le but de permettre une communication entre deux entités. Plus précisément, dans le cas où deux entités n'appartiennent pas à la même PKI, l'ajout du module entre les deux leur permet de communiquer sans avoir à faire confiance à l'ensemble des entités appartenant à l'autre PKI. A l'instar de tous les équipements de sécurité, l'apport en sécurité fournit par le module le contraint à devenir la cible principale lors des attaques.

En nous inspirant des cibles de sécurité utilisées pour évaluer les équipements réseaux usuels, nous avons analysé la sécurité d'un module en spécifiant les aspects de son architecture qui lui permettaient de répondre aux critères de sécurité. Enfin, nous avons montré que l'architecture proposée pour un module de rupture protocolaire est applicable aux architectures de contrôle-commande de type SCADA grâce à son implémentation dans le cadre du projet ARAMIS.

CHAPITRE IV

LOCALPKI : UNE PKI ALTERNATIVE CENTRÉE SUR L'UTILISATEUR

Sommaire

RÉSUMÉ

Dans ce chapitre, nous exposons une nouvelle infrastructure de gestion de clefs centrée sur l'utilisateur et formellement prouvée, dénommée LocalPKI reposant sur le paradigme *PKI2.0* [BGT11]. A l'instar de la solution fournie par *Let's Encrypt* pour la distribution de certificats aux serveurs, l'objectif est de démocratiser l'attribution et l'usage des certificats, pour les personnes physiques, les modules indépendants tels ARAMIS, ou encore les objets connectés (*Internet of Things* (IOT)). L'idée principale de LocalPKI est de remplacer la signature de l'AC d'un certificat uniquement par celle de l'utilisateur : autrement dit, tous les certificats de LocalPKI sont auto-signés. Cependant, la garantie que la clef publique est véritablement associée à l'identité du propriétaire inscrite sur le certificat est toujours donnée par un tiers de confiance, appelé notaire. Une fois que l'utilisateur s'est enregistré auprès de ce dernier, les autres seront capables de vérifier l'authenticité de son certificat par une requête au notaire, assimilable à une requête OCSP dans PKIX. La réponse du notaire est ensuite dépendante de la présence ou non de cet utilisateur dans sa base de données, contenant uniquement des informations sur les certificats valides. En outre de cette méthode dite privée, les utilisateurs ont aussi le choix d'utiliser un mode public pour effectuer la vérification : dans ce cas, le notaire fournit un sous-ensemble signé de sa base de données, à l'instar de NSEC3 [KMG12] (ou NSEC5 [VGP16]) dans DNSSEC [BM10]. Comme dans PKIX, l'enregistrement n'est pas nécessairement réalisé par l'autorité de validation, mais par une entité dédiée à cette tâche (l'autorité locale d'enregistrement). Dans LocalPKI, cette entité doit être connue du notaire, mais doit aussi être proche de l'utilisateur. Elle n'a toutefois pas forcément besoin de compétence technique dans la gestion des clefs, et donc ces fonctions peuvent être réalisées par un bureau de poste, un opérateur mobile ou encore une université. Ce chapitre s'inspire des résultats que nous avons publiés dans [DLM+17].

ORGANISATION

Nous commençons par définir les entités impliquées dans LocalPKI dans la section (2.1), avant d'expliciter les protocoles d'enregistrement (2.2), d'authentification (2.3) et de révocation (2.4). Par la suite (Section 3), nous nous intéressons aux propriétés de sécurité désirées et au modèle utilisé dans Tamarin pour réaliser les preuves de sécurité. Finalement, nous montrons en conclusion comment adapter les outils existants pour les appliquer à LocalPKI, avant de nous intéresser à son application au cas industriel.

1 ■ LES BESOINS D'UNE PKI CENTRÉE SUR L'UTILISATEUR

L'objectif d'une infrastructure à clef publique est de fournir des services permettant de relier l'identité d'un utilisateur à sa clef publique (voir (II.3) pour plus de détails). Sur Internet, le standard le plus utilisé est PKIX : l'authentification de la majorité des sites webs offrant un connexion sécurisée à leurs serveurs est faite en utilisant le protocole *TLS* (Transport Layer Security). Dans ce cadre, les utilisateurs font confiance aux autorités de certification délivrant les certificats *X.509* qui, en apposant leur signature sur ces derniers, assurent la véracité des informations qu'ils contiennent (à savoir l'identité, la clef publique et une période de validité au minimum). L'utilisateur utilise ensuite la clef publique de cette entité pour s'assurer que la signature est correcte, et en déduire que le certificat l'est aussi. Il doit ensuite vérifier que le certificat n'a pas été révoqué, il utilise une des deux méthodes les plus répandues : le protocole OCSP [SAM+13] ou les CRL [Coo08]. Dans le cas où le certificat n'a pas été révoqué, l'utilisateur réussi donc à authentifier le propriétaire du certificat.

Une difficulté de PKIX concerne sa mise en place, qui requiert des connaissances techniques importantes et augmentent son coût d'utilisation déjà non négligeable. Nous avons illustré ce problème dans le chapitre précédent, avec l'exemple des infrastructures de contrôle-commandes, où il est clair que le déploiement de la norme PKIX est relativement difficile. L'administration et la certification d'un nouveau jeu de clefs nécessitent de nombreuses manipulations. Enfin, dans le cas d'un équipement privé de connexion à un répondeur OCSP, la vérification de l'état d'un certificat repose alors sur l'utilisation des CRL, impliquant une possible authentification d'une entité ayant un certificat révoqué. Plusieurs modèles de PKI ont donc été étudiés pour mieux correspondre aux besoins des architectures industrielles [WL13]. En pratique, ils ne sont que peu appliqués, car reposant généralement sur de nouveaux outils à mettre en place.

De plus, malgré les récentes avancées dans le domaine des PKI, (voir II.5.2), telles que l'affaiblissement de la confiance nécessaire dans les autorités de certification ou la réduction des coûts de communication globaux d'une PKI en fonction de son environnement, peu de solutions alternatives facilitant le déploiement des certificats auprès des usagers n'ont réellement été mises en place. En comparaison, la solution *Let's Encrypt* [Aas14] a pourtant permis de simplifier la démarche d'acquisition de certificat pour les serveurs, mais son application pour les utilisateurs reste minoritaire.

C'est pour cela que nous souhaitons développer une alternative à PKIX centrée sur l'utilisateur. En pratique, cela signifie que nous souhaitons que la génération et la fabrication des certificats soient au maximum simplifiées. L'idée est alors de séparer

les personnes affectées aux différentes procédures : typiquement, l'enregistrement d'un nouvel utilisateur ne doit pas nécessiter de compétences techniques. De plus, elle doit pouvoir fournir à l'utilisateur un certificat qu'il pourra immédiatement distribuer pour être authentifié par la suite, sans qu'il n'ait besoin de suivre aucune démarche supplémentaire : les certificats auto-signés, comme pour PGP, nous semblent être une bonne solution. De manière générale, l'entité en charge de l'enregistrement doit simplement être capable d'effectuer une vérification d'identité. Par conséquent, une agence proche de l'utilisateur, telle qu'une banque ou un bureau de poste peut parfaitement convenir. L'utilisateur a ainsi la possibilité d'obtenir un certificat gratuitement (ou éventuellement avec des options payantes), facilement (car proche de son lieu de vie) et avec l'assurance que les informations sont correctes (puisque l'opération est faite sous ses yeux). Enfin, à l'instar de PKIX, nous souhaitons utiliser un tiers de confiance, un notaire, qui sera contacté de manière transparente par l'utilisateur pour vérifier un certificat. Enfin, il est généralement dans l'intérêt des autorités locales d'enregistrement de pouvoir fournir aisément des certificats à leurs utilisateurs/clients, en leur offrant par exemple la possibilité d'authentifier et être authentifiés pour des opérations en ligne, telles que la signature d'un contrat.

En résumé, la problématique du chapitre est la suivante :

Comment définir une PKI facilitant l'usage de certificats pour les utilisateurs finaux (à la PGP), mais reposant sur la confiance dans un tiers et fournissant des garanties de sécurité équivalentes à PKIX ?

Pour y répondre, nous avons développé une nouvelle architecture de gestion de clefs, dénommée LocalPKI. L'avantage de LocalPKI réside dans le fait qu'il n'est pas nécessaire pour ces deux entités d'avoir des compétences techniques, qui sont déléguées au notaire. Globalement, LocalPKI est donc une alternative PKIX, offrant des services similaires, mais dont l'approche est centrée sur l'utilisateur, en s'appuyant uniquement sur des certificats auto-signés. De plus, LocalPKI offre des garanties de sécurité similaires à PKIX. Cette proposition est démontrée grâce à l'outil de preuves formelles automatiques Tamarin [MSCB13, SMCB12]. Elle permet aussi de s'affranchir de certaines difficultés rencontrées lors de la mise en place d'une PKI classique dans le cadre de la sécurisation d'un réseau industriel. En particulier, nous montrons que LocalPKI, dont l'objectif originel est de cibler les utilisateurs finaux, s'adapte finalement aux besoins que nous avions avec le déploiement d'un module de rupture protocolaire. En sus, nous nous efforçons d'utiliser des outils déjà connus (ceux de PKIX), et de les adapter pour répondre à nos besoins, dans le but de faciliter la migration à partir d'une PKIX.

2 ■ LOCALPKI : DÉFINITION D'UNE PKI CENTRÉE SUR L'UTILISATEUR

LocalPKI est un ensemble de protocoles répondant aux besoins d'une infrastructure à clefs publiques, tels que :

- Enregistrement d'un nouvel utilisateur ;
- Authentification d'un utilisateur enregistré ;
- Révocation de certificats ;

Dans cette section, nous commençons par définir les acteurs intervenant dans ces protocoles, avant de détailler le fonctionnement de chacun d'entre eux.

2.1 | ENTITÉS IMPLIQUÉES DANS LOCALPKI

Trois entités distinctes interviennent dans LocalPKI : les notaires électroniques, abrégés NE (en anglais, *Electronic Notary* (EN)), les autorités locales d'enregistrement (*Local Registration Authority* (LRA)) et les utilisateurs (*Users* ou *End Entities*). Le notaire peut être vu comme une AC racine dans une architecture PKI classique. Il gère les bases de données contenant les utilisateurs enregistrés. L'autorité locale d'enregistrement représente un intermédiaire entre l'utilisateur et le notaire. C'est en quelque sorte l'autorité d'enregistrement (*Registration Authority*) d'une PKI, mais qui se veut plus proche de l'utilisateur qu'une AC. En pratique, le LRA peut être une agence de proximité de l'utilisateur, telle que sa compagnie d'assurance, sa banque ou encore son bureau de poste. Ces agences sont généralement capable d'effectuer des contrôles d'identité. Ainsi, le LRA est enregistré par un sous ensemble de notaires, et les vérifications d'identité des utilisateurs souhaitant s'enregistrer sont réalisés par le LRA. Enfin, les utilisateurs représentent les entités qui souhaitent en authentifier un autre, ou être lui-même authentifié par un autre usager.

2.2 | ENREGISTREMENT D'UN NOUVEL UTILISATEUR

Dans le but d'être connu dans l'infrastructure, et d'être authentifiable par les autres, un utilisateur doit tout d'abord être enrôlé dans le système : c'est *l'enregistrement*. Dans LocalPKI, cette étape repose principalement sur l'utilisation d'un couple contenant un identifiant (SN) et une signature (SI) réalisée par l'entité souhaitant s'enregistrer, preuve de la connaissance de la clef privée associée à la clef publique du certificat. Le couple (SN, SI) représente l'extraction et l'implémentation du certificat de propriété de clef publique (*public key ownership certificate*) décrit dans [BGT11].

L'utilisateur commence tout d'abord par générer une nouvelle paire de clefs. Il interagit ensuite avec le LRA pour débuter le processus d'enregistrement. Dans LocalPKI, le LRA est supposé proche de l'utilisateur afin de pouvoir le rencontrer en personne. Lors de ce face-à-face, l'utilisateur commence par fournir des preuves de son identité, en accord avec la politique de sécurité définie par l'agence (un contrôle visuel de la carte d'identité par exemple). L'utilisateur fournit ensuite sa clef publique au LRA. Ce dernier est alors en charge de la génération d'un champ équivalent au *ToBeSigned* (`TBSCert`) des certificats $X.509$, contenant au moins : l'identité de l'utilisateur, sa clef publique, un numéro de série (*Serial Number*, SN), une période de validité et l'URL du notaire associé au LRA. Le SN a été précédemment obtenu lors d'un échange entre le LRA et le notaire. Ce dernier doit générer en amont des intervalles de numéro de série, qui seront ensuite attribués à chaque LRA. L'utilisateur hache ensuite le $TBSCert$ obtenu précédemment, et signe l'empreinte : le résultat de la signature est dénoté SI, pour *Signature Id.* Le LRA vérifie ensuite la signature en calculant à son tour l'empreinte et en utilisant la clef publique de l'utilisateur. A cette étape, l'utilisateur a donc prouvé au LRA qu'il connaissait la clef privée associée à la clef publique fournie. Le LRA chiffre le couple en utilisant la clef publique du Notaire Electronique (NE), et l'envoie au notaire, concaténé avec sa signature (c.-à-d. $\{H(SN,SI)\}_{Sk_{LRA}}$). L'enregistrement de l'utilisateur se termine lorsque le NE enregistre l'unique couple (SN,SI), envoyé par le LRA, dans sa base de données tel que :

- SN : Numéro de série attribué au certificat ;
- SI : Signature du certificat par son détenteur, avec : $SI = \{H(TBSCert)\}_{Sk}$

A la fin de son enregistrement, l'utilisateur possède un certificat contenant le $TBSCert$ (qui comprend un SN) ainsi qu'une auto-signature SI, tandis que le notaire a ajouté le couple (SN,SI) dans sa base de données. Le processus complet est détaillé dans l'algorithme 10 et illustré dans le schéma 1.

2.3 | AUTHENTIFICATION

Une fois que le propriétaire du certificat a bien été enregistré, les autres utilisateurs doivent pouvoir l'authentifier, c.-à-d. s'assurer que la clef publique récupérée est bien celle du propriétaire attendu. Dans LocalPKI, nous avons défini deux méthodes d'authentification. Tout d'abord en utilisant **le mode privé**, où seul le notaire connaît le contenu complet de sa base de données. Dans ce cas, l'utilisateur demande au notaire si le certificat qu'il a récupéré est valide, et aucune autre information n'est révélée. Dans la seconde option, appelée **mode public** (ou *buffering mode*), le notaire partage une partie de sa base de données avec l'utilisateur, qui réalise lui-même la vérification. Dans les deux cas, l'utilisateur doit tout d'abord authentifier le NE. Pour cela, il utilise un mécanisme semblable à celui utilisé dans PKIX, à savoir les magasins d'ancres de confiance. Dans ce cas, le magasin de

Algorithme 10 Enregistrement d'Alice

Entrée(s): Le LRA possède *a priori* une liste de numéros de série, fournie par le NE de confiance.

Sortie(s): La vérification de l'identité d'Alice (par le LRA) et l'enregistrement d'Alice dans la base du NE.

1: Alice génère sa paire de clefs (Pk_A, Sk_A).
2: Alice $\rightarrow$ LRA : Pk_A
3: Le LRA vérifie les informations et l'identité d'Alice.
4: LRA $\rightarrow$ Alice : Numéro de série (SN_A), l'URL du notaire (URL_{EN}), une durée de validité.
5: Alice génère $TBSCert_A$ (similaire à celui d'un certificat $X.509$ et complété avec URL_{EN} et SN_A), et calcule $SI_A = \{H(TBSCert_A)\}_{Sk_A}$
6: Alice $\rightarrow$ LRA : $TBSCert_A||SI_A$
7: LRA vérifie SI_A [*Preuve de la connaissance d'Alice dans la clef privée*].
8: **Si** Verification OK **Alors**
9: LRA $\rightarrow$ EN : $\{SN_A||SI_A\}_{Pk_{EN}}||\{H(SN_A||SI_A)\}_{Sk_{LRA}}$
10: Le NE ajoute le couple (SN_A, SI_A) à sa base de données.

l'utilisateur contient les certificats des notaires de confiance. Ainsi, les usagers sont capables de récupérer des informations sur les notaires, dont sa clef publique. Dans la suite, nous détaillons les deux modes d'authentification.

2.3.1 • MODE PRIVÉ OU INTERACTIF

La spécificité du mode privé réside dans la base de données contenant les utilisateurs enregistrés, qui est uniquement connue par le notaire. La vérification du certificat d'un utilisateur (le propriétaire) par un autre (le vérificateur) est donc réalisée en interagissant avec le notaire. L'algorithme (11) détaille les différentes étapes.

En mode privée, le vérificateur commence contrôler l'intégrité du certificat. Pour cela, il extrait la clef publique, le SI et le $TBSCert$ du certificat. Il hache ensuite ce dernier et utilise l'algorithme de signature avec la clef publique sur le SI, et vérifie l'égalité entre les deux. Puis, le vérificateur s'assure que l'URL contenue dans le certificat du propriétaire est associé à l'un des certificats de son ancre de confiance, sinon l'authentification ne peut pas être faite. L'étape suivante consiste à générer l'*Authentication Request* (AR), composée du certificat à contrôler et d'un nonce R. Il envoie ensuite la requête au NE désigné par l'URL. Le notaire vérifie alors si ce couple apparaît dans sa base de données, et répond positivement si c'est le cas. La réponse complète comprend : la valeur précédente, le nonce R, le couple (SN, SI) et une signature de l'ensemble par sa clef privée. Le vérificateur contrôle la signature

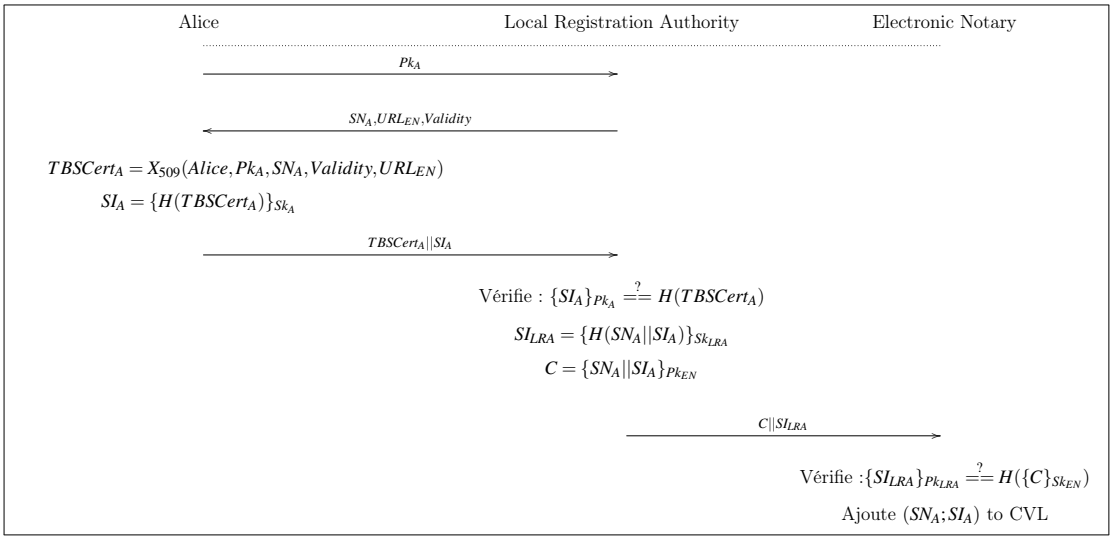

FIGURE 1 – Enregistrement d'Alice.

de la réponse du notaire en utilisant la clef publique extraite de son certificat. Un exemple d'authentification peut être trouvé dans la figure 2.

2.3.2 • MODE PUBLIC OU LISTE DE VÉRIFICATION DES CERTIFICATS

Dans le mode public, la vérification du certificat du propriétaire est réalisée par le vérificateur. Dans cette configuration, l'utilisateur n'a besoin d'échanger avec le notaire que de manière partielle. Nous définissons la procédure suivante : après avoir récupéré le certificat, le vérificateur commence par vérifier que la signature (soit SI) est cohérente avec les informations contenues dans le $TBSCert$. Après s'être assuré que le notaire associé au certificat fait partie de son ancre de confiance, le vérificateur lui envoie une requête pour obtenir la liste de vérification des certificats (*Certificate Verification List*, *Certificate Verification List* (CVL)). Cette liste contient l'ensemble des couples (SN, SI) des utilisateurs qui ont été enregistrés. Pour s'assurer de l'intégrité des données qui sont envoyées à l'utilisateur, le NE joint une signature de la liste en plus de son contenu dans la réponse. Le vérificateur contrôle ensuite l'intégrité des données reçues, et examine la CVL afin de voir si le couple (SN, SI) est inclus. A l'instar des CRL dans PKIX, une CVL peut être utilisée pendant une certaine durée, par exemple quelques jours, évitant ainsi un nouvel échange de données. Notons que les deux phases du protocole peuvent être inversées : l'utilisateur commence alors par récupérer les CVL auprès du notaire. Durant la durée de vie de la CVL, il pourra vérifier tous les certificats enregistrés chez ce notaire, sans aucune interaction supplémentaire. Le schéma 3 synthétise le

Algorithme 11 Vérification du certificat dans le mode privé (avec auto-vérification de la signature)

Entrée(s): Alice récupère le certificat de Bob. Elle souhaite se renseigner sur sa validité.

Sortie(s): L'authentification de Bob par Alice dans le cas où le certificat est correct, échec sinon.

1: **Si** $\{SI_B\}_{Pk_B} \stackrel{?}{=} H(TBSCert_B)$ **Alors**
2: Alice : $R_A \leftarrow^{\$}$
3: Alice $\rightarrow URL_{EN} : AR = SN_{Bob}||SI_{Bob}||R_A$ (*Authentication Request*)
4: **Si** $(SN_{Bob}; SI_{Bob}) \in$ Base de données **Alors**
5: $Rep = "OK"||AR$
6: **Sinon**
7: $Rep = "Unknown"||AR$
8: $URL_{EN} \rightarrow$ Alice : $Rep||\{H(Rep)\}_{Sk_{EN}}$
9: Alice vérifie la signature de la réponse, et authentifie (ou non) Bob.

processus d'authentification en mode public.

Même si le mode public permet de limiter le nombre d'opérations réalisé par le NE, il implique un coût non négligeable de communication. Afin de les réduire, l'idée est de diviser la base de données en sous-domaines. En effet, comme le notaire est en charge de la génération des numéros de série *SN* fournis au LRA, sa base de données est intrinsèquement divisée en sous-domaine. Ainsi, le vérificateur peut récupérer la CVL associée au sous-domaine, et donc réduire les coûts à la taille du sous-domaine.

2.4 | RÉVOCATION

Une infrastructure à clef publique doit permettre de révoquer les certificats émis avant la fin de leur durée de validité, notamment pour gérer la perte, la compromission de clefs mais les changements des droits d'accès par exemple. Nous expliquons ici comment procéder dans LocalPKI. Tout d'abord, la révocation peut être demandée par le propriétaire du certificat ou son LRA, via une requête envoyée au NE. Dans les deux cas, la demande est signée et contient le certificat ainsi que le message `Revoke`. Dans le cas où le propriétaire est à l'origine de la demande, la signature est une preuve de connaissance de la clef privée. Pour la vérification de la signature, le notaire authentifie l'entité via le certificat ajouté à la requête. Lorsque le LRA demande la révocation, le notaire accepte la demande à condition que le *SN* appartienne à l'intervalle dédié à cette agence locale. Lors de la réception de la requête, le NE vérifie la signature ainsi que la présence du couple

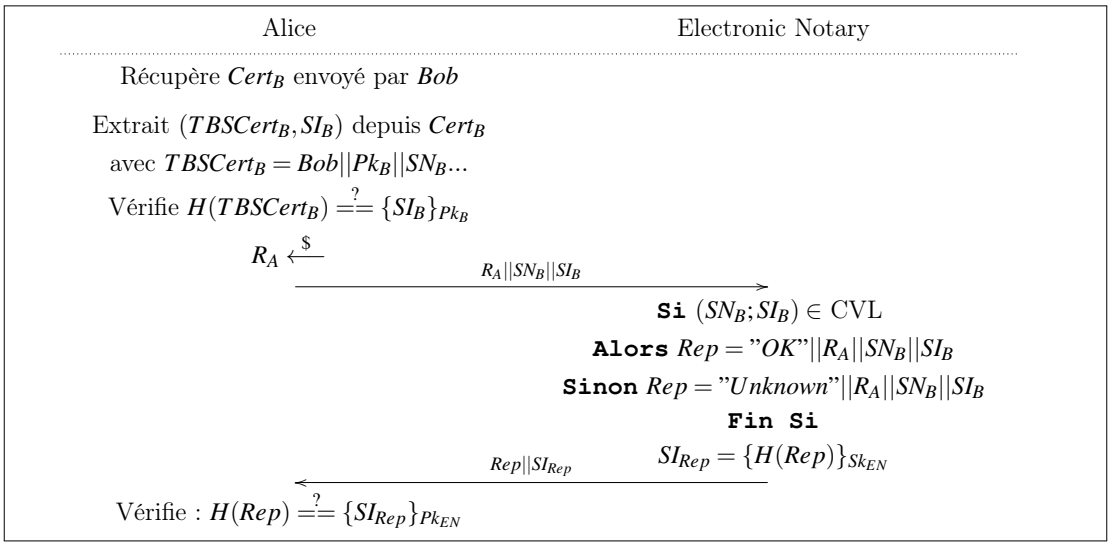

FIGURE 2 – Authentification de Bob par Alice dans le mode privé.

(SN, SI) dans sa base de données : si tel est cas, il le supprime simplement. Par la suite, les demandes d'authentification avec ce couple seront donc refusées, puisqu'il a été supprimé de la base de données. Cependant, un utilisateur dont le certificat a été révoqué peut le renouveler en procédant à un nouvel enregistrement auprès de son LRA.

Algorithme 12 Révocation de certificat

Entrée(s): Alice est correctement enregistrée dans la base de données du NE. $X \in \{\text{LRA}, Alice\}$

Sortie(s): La révocation du certificat de l'utilisateur.

1: $X \rightarrow EN : SI_{Rev} = Cert_X||\{H("Revoke"||(SN_{Alice}; SI_{Alice}))\}_{Sk_X}$
2: NE vérifie la signature
3: **Si** La vérification est correcte et $(SN_X; SI_X) \in$ Base de données **Alors**
4: NE supprime $(SN_X; SI_X)$ de la base de données.

3 ■ ANALYSE DE SÉCURITÉ DE LOCALPKI

L'objectif principal de LOCALPKI est de permettre l'authentification d'utilisateurs utilisant la cryptographie asymétrique à l'aide d'un certificat auto-signé associé. La principale propriété à vérifier est donc l'authentification. Nous utilisons un outil de vérification automatique dénommé *Tamarin Prover* (voir II.8.2) afin de démontrer la sécurité de nos protocoles. Dans ce contexte, d'autres propriétés

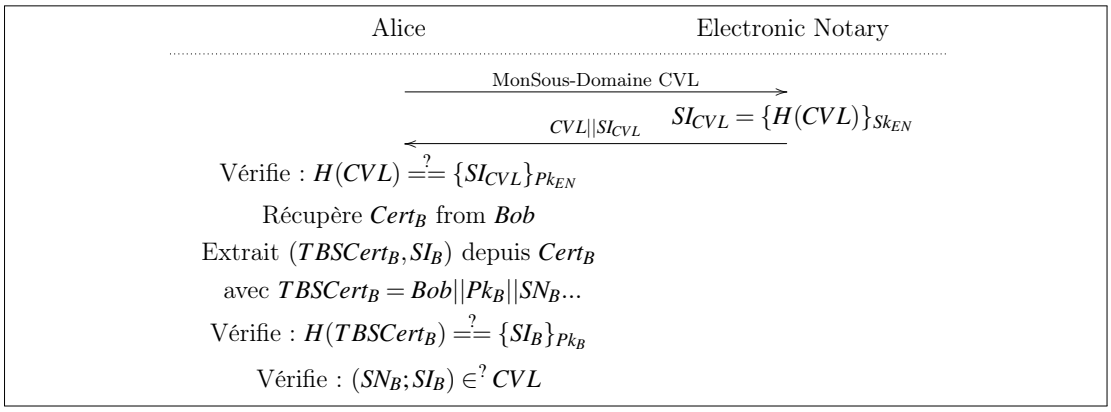

FIGURE 3 – Authentification de Bob par Alice dans le mode public

sont aussi démontrées, afin de prouver que le protocole s'exécute correctement par exemple. Dans la suite, nous définissons les propriétés de sécurité désirées, puis nous décrivons ensuite le modèle que nous avons utilisé pour implémenter nos protocoles.

3.1 | PROPRIÉTÉS DE SÉCURITÉ

La première propriété requise est l'exactitude (*correctness*) du protocole, c.-à-d. que si une personne est correctement enregistrée dans la base de données et que son certificat est valide, alors il est possible de vérifier son certificat (et donc de l'authentifier). La propriété sous-jacente montre que nos protocoles et modèles fonctionnent tels que nous le souhaitons.

Vice versa, la sûreté (*soundness*) du protocole peut être définie de la manière suivante : un adversaire qui n'a pas été enregistré ne peut pas valider la procédure de vérification de certificat. Cette propriété implique qu'un adversaire n'est pas capable de forger un certificat qui est considéré comme valide et ne peut pas en compromettre un qui est valide. La sûreté peut être reformulée ainsi : une procédure de validation d'un certificat (non révoqué) à un temps i_2 implique l'enregistrement de ce certificat à un temps i_1 et $i_1 < i_2$. Notre définition de la sûreté correspond à celle de l'authentification dans le modèle symbolique ([Bla04]).

Dans [BCK+14], les auteurs définissent l'intégrité de connexion (*Connection Integrity*) de la sorte : si un utilisateur établit une connexion avec un autre, alors cet utilisateur est propriétaire légitime de la clef privée. En d'autres termes, après l'enregistrement et la vérification du certificat, l'adversaire ne connaît la clef privée associée à la clef publique inscrite sur le certificat (*i.e.* la clef privée d'Alice).

Enfin, le protocole doit assurer la confidentialité (*Confidentiality*) des données. En particulier, lorsque qu'un utilisateur a été authentifié, le message qui lui est

envoyé doit rester secret, c.-à-d. que l'adversaire ne peut pas connaître le message.

En résumé, nous souhaitons donc que LocalPKI vérifie les propriétés suivantes :

1. **Exactitude :** si un utilisateur est correctement enregistré et que son certificat n'est pas révoqué, alors il est possible de l'authentifier ;
2. **Sûreté :** si un utilisateur a été authentifié, alors il a du être enregistré auparavant et son certificat n'a pas été révoqué ;
3. **Intégrité de connexion :** si un utilisateur a été correctement enregistré et son certificat vérifié, alors l'adversaire ne connaît pas la clef privée ;
4. **Confidentialité :** lorsqu'un utilisateur a été authentifié, les messages qui lui sont adressés ne peuvent pas être connus par l'adversaire.

3.2 | MODÈLE TAMARIN DE LOCALPKI

Nous présentons maintenant le modèle formel en Tamarin de LocalPKI. Nous considérons une exécution du protocole où seuls un notaire, un LRA et un utilisateur enregistré (Alice) et un vérificateur (Bob) sont représentés. D'après les résultats théoriques de [CC04], une instance de chaque rôle (honnête ou adversaire) est suffisante pour prouver les propriétés d'authentification et de confidentialité. Nous avons modélisé l'enregistrement d'Alice, la vérification de son certificat par Bob suivi par l'envoi d'un message, et la procédure de révocation de son certificat.

Dans les modélisations classiques de PKIX [Bla04, BCD+], la génération des paires de clefs est faite en utilisant des faits persistants employés à plusieurs reprises avec différents termes. Ces faits relient l'identité d'une entité avec sa clef publique, afin de représenter les ancres de confiance. Nous employons cette méthode donc uniquement pour modéliser les ancres de confiance du LRA et du NE. Dans le cas d'Alice, comme nous souhaitons démontrer que les certificats de LocalPKI permettent d'avoir un lien sûr entre l'identité et la clef, nous explicitons donc son certificat auto-signé décrit dans l'algorithme 10. Nous supposons ensuite que ce certificat est public.

Nous utilisons deux types de canaux de communication, en fonction de la situation : un pour modéliser les échanges face-à-face, et un pour les échanges à distance. Le premier canal, qui intervient notamment lors de l'enregistrement, n'est pas sujet aux attaques d'un intrus : cela modélise le fait que lorsque Alice et le LRA communiquent, l'adversaire n'est pas capable d'apprendre ou de modifier les informations échangées. Toutes les autres communications peuvent être mises sous écoute et éventuellement modifiées par l'adversaire. L'enregistrement de (SN_A, SI_A) dans la base de données est représenté à l'aide d'un fait associant le couple à l'identité du notaire.

Enfin, pour la révocation, comme Tamarin utilise des traces d'exécution (c.-à-d. l'enchaînement des différents états) du protocole, la modélisation de la suppression

d'un certificat de la base de données est exprimée comme l'association d'une étiquette spécifique (aussi appelée *tag*) au couple dans un fait persistant. Cela nous permet d'exprimer l'aspect définitif d'une révocation.

Le code source du modèle en Tamarin peut être récupéré à l'adresse : https://jgdumas.gricad-pages.univ-grenoble-alpes.fr/localpki/.

3.3 | LEMMES

Nous commençons par décrire l'implémentation des propriétés de sécurité en tant que lemmes dans Tamarin. En adéquation avec la partie précédente 3.1, nous avons des lemmes sur l'exactitude, la sûreté, l'intégrité de connexion et la confidentialité de LocalPKI. Par la suite, nous utilisons parfois le terme d'authentification pour désigner une vérification correcte du certificat d'Alice par Bob.

3.3.1 • SÛRETÉ (SOUNDNESS)

Le premier lemme est à propos de la sûreté (*soundness*) de LocalPKI. Le couple `<sna,sia>` représente le numéro de série (*SN*) et l'identifiant de signature (*SI*) d'Alice. Les labels utilisés dans les lemmes représentent le moment où l'évènement se produit. L'idée est de prouver que pour les couples (*sna*, *sia*) possibles, associés à la clef privée d'Alice `ltkA`, si Bob a authentifié correctement Alice au temps `i` (`Bob_Auth_Alice()`), cela signifie qu'Alice a été enregistrée auparavant (au temps `j` par le notaire `EN_Reg_Alice()`) et que si le certificat a été révoqué, alors cette action s'est déroulée à un temps postérieur `k`.

Soundness

```
lemma soundness:
   all-traces
   "All EN B sna sia #i.
    Bob_Auth_Alice(B, <sna,sia>, ltkA) @i
    ==>
    (Ex #j. EN_Reg_Alice(EN, <sna,sia>, ltkA)@j
     & (j<i))
     & (All #k.
        Cert_Is_Revoked(EN, <sna,sia>, ltkA)@k ==> i<k)"
```

3.3.2 • EXACTITUDE (CORRECTNESS)

Le second lemme (`exactitude`) permet de vérifier que le modèle est correct, c.-à-d. qu'il existe une exécution telle que Alice s'est enregistrée, Bob l'authentifie,

et le certificat n'est pas révoqué. Grâce à ce lemme, nous nous assurons qu'une exécution normale du protocole est possible, et que le résultat est celui attendu : Bob obtient effectivement la clef publique associée à Alice.

Correctness

```
lemma correctness:
   exists-trace
   "(Ex EN B sna sia ltkA #i #j.
     EN_Reg_Alice_(EN, <sna,sia>, ltkA) @j
     & Bob_Auth_Alice(B,<sna,sia>, ltkA) @i
     & not(Ex EN #l. Cert_Is_Revoked(EN, <sna,sia>, ltkA) @l
     & j<l))"
```

3.3.3 • CONFIDENTIALITÉ (SECRECY)

Le lemme de confidentialité (`secrecy global`) nous permet de montrer que les messages échangés après une authentification utilisant les mécanismes de LocalPKI ne peuvent pas être connus par l'adversaire. Nous prouvons en Tamarin qu'un message dénoté `x` ne doit jamais être connu de l'adversaire (`K()`).

Secrecy

```
lemma secrecy_global:
all-traces
"All x #i. Secret(x) @i ==> not(Ex #j. K(x)@j)"
```

3.3.4 • INTÉGRITÉ DE CONNEXION (CONNECTION INTEGRITY)

De manière similaire, avec le lemme d'intégrité de connexion, nous nous assurons que la clef secrète d'Alice `ltkA` n'est pas connue de l'adversaire dans le cas où Bob authentifie Alice (qui est donc enregistrée si la sûreté est vérifiée) ou que le certificat a été révoqué.

Connection Integrity

```
lemma connection:
   all-traces
   "(All EN sna sia ltkA #i.
      Bob_Auth_Alice(B, <sna,sia>, ltkA)@i
      ==> not(Ex #l. K(ltkA)  @l))
     & (All B sna sia ltkA #i.
        Cert_Is_Revoked(EN, <sna,sia>, ltkA) @i
        ==> not(Ex #l. K(ltkA)  @l))"
```

3.4 | VÉRIFICATION DES LEMMES ET CORRUPTIONS

Notre modélisation en *Tamarin* nous permet de voir si les lemmes précédemment définis sont vérifiés, selon certaines hypothèses de corruption. Dans un premier temps, nous supposons que l'adversaire est uniquement externe : ce dernier repose sur le modèle de Dolev-Yao. Il est donc capable de contrôler le réseau et d'extraire des informations des messages échangés. Grâce à ce modèle, nous pouvons ainsi observer si LocalPKI est capable de fournir des services résistants face à des adversaires n'étant pas acteurs de l'infrastructure.

Nous nous intéressons aussi aux hypothèses faites sur la confiance accordée aux LRA et NE dans le but de préserver la sécurité du protocole (comme c'est le cas dans PKIX). En d'autres termes, nous voulons exposer les possibilités d'attaques du LRA et le NE lorsqu'ils sont malicieux. Dans notre modèle, la corruption d'une entité est réalisée en partageant sa clef privée avec l'adversaire.

Confiance envers le NE Dans le cas où le notaire est corrompu, la confidentialité n'est pas vérifiée car ce dernier est capable d'ajouter une fausse clef à sa base de données. Ainsi, après la prétendue authentification d'Alice par Bob en utilisant cette paire de clef, une simple mise sur écoute des communications permet au notaire de retrouver les messages supposés secrets. Le lemme sur la sûreté est aussi invalidé : comme l'adversaire a accès à la clef privée du NE, il agit comme si Alice avait été enregistrée auparavant bien que cela ne soit pas le cas. En pratique, cette attaque montre la possibilité du notaire à fournir l'authentification aux entités de son choix, sans respecter le protocole. Concernant la propriété de l'intégrité de connexion, Tamarin trouve aussi une attaque dès que la clef du NE est révélée à l'adversaire. Elle fonctionne de la manière suivante : après son initialisation, le notaire s'enregistre lui-même en tant qu'Alice. En d'autres termes, cela signifie que l'adversaire forge un certificat en utilisant l'identité d'Alice, mais lui associe sa clef publique, et signe en utilisant sa clef privée. A l'étape suivante, Bob authentifie

Alice en utilisant le certificat forgé, et l'adversaire connaît la clef privée associée : la connexion n'est plus intègre. Cette attaque souligne tout d'abord la capacité du notaire à émettre des contrefaçons de certificats, mais montre aussi que le lemme d'intégrité de connexion est bien défini. En effet, l'intégrité de connexion est falsifiée, alors que le notaire ne connaît pas réellement la clef privée d'Alice, mais uniquement celle qui lui est associée dans le certificat. Ces attaques nous montrent que le NE doit être considéré comme une entité de confiance.

Confiance envers LRA Lorsque la clef du LRA est révélée, Tamarin ne trouve pas d'attaque à propos de la sûreté. D'un point de vue pratique, ce résultat est cohérent : comme le LRA participe uniquement à l'enregistrement, sa seule possibilité est de fournir de fausses informations au notaire. L'enregistrement est donc nécessaire même si le couple (SN_A, SI_A) n'est pas celui associé à Alice. Cependant, en supprimant l'unicité d'enregistrement des informations précédentes, l'outil est capable de trouver une attaque en fournissant à nouveau un certificat qui a été révoqué au notaire. Cette attaque est facilement contrée, en ajoutant une simple vérification de la part du NE sur les couples déjà utilisés. Le lemme sur la confidentialité n'est plus vérifié, puisque l'adversaire se fait passer pour le LRA durant la dernière étape de l'enregistrement, en signant un faux certificat contenant l'identité d'Alice associée à la mauvaise paire de clefs. Ainsi, lorsque Bob utilise les informations contenues dans le faux certificat, le LRA connaît donc la clef privée associée, et peut retrouver les messages échangés. L'intégrité de connexion n'est plus valide lorsque la clef du LRA est donnée à l'adversaire. L'attaque trouvée est assez proche de la précédente : lors de l'enregistrement d'Alice, le LRA va modifier son certificat pour y ajouter sa clef publique, et le signer en utilisant sa clef privée. La contrefaçon du certificat est envoyée au NE. Ce certificat est ensuite révoqué par le notaire, et l'adversaire connaît la clef privée associée : l'intégrité de connexion n'est plus vérifiée. Globalement, cette attaque montre la possibilité d'un LRA à forger des certificats : il doit donc être considéré comme une entité de confiance.

Confiance envers Alice Lorsque Alice est corrompue, les répercussions sur la sécurité du protocole sont limitées. En particulier, la sûreté n'est pas influencée. Ce résultat est cohérent, même si la clef privée d'Alice est donnée à l'adversaire, il ne peut s'identifier sans être enregistré. En pratique, Alice pourrait essayer de duper le LRA lors des contrôles d'identité. Cependant, le contrôle de l'identité d'Alice, reposant sur la présentation et l'examen de ces documents d'identité, est un processus réalisé hors-ligne. Il n'est donc pas possible de le modéliser en Tamarin, et les attaques relatives à la falsification des documents d'Alice ne sont donc pas prises en compte dans cette analyse. Concernant le lemme de confidentialité, il n'est évidemment plus vérifié puisque la clef privée d'Alice est donnée à l'adversaire.

Cette attaque est l'équivalent en pratique d'un vol de clef, où Alice et l'attaquant sont capables de retrouver les informations contenues dans des chiffrés. De même, l'intégrité de connexion est intrinsèquement cassée, puisque l'adversaire connaît la clef secrète d'Alice par définition. Globalement, nous pouvons conclure que Alice ne doit pas être un tiers de confiance.

3.5 | LA SÉCURITÉ DE LOCALPKI

En résumé, grâce à notre analyse de sécurité réalisée avec Tamarin, nous avons donc prouvé le théorème suivant :

Théorème 13. *En supposant que le notaire et l'autorité locale d'enregistrement ne sont pas corrompus, alors l'architecture de* LocalPKI *est exacte, sûre, et préserve la confidentialité et l'intégrité.*

Notre implémentation comprend environ 350 lignes de code, décrivant 20 règles et 4 lemmes principaux. Dans le but de réduire les temps de preuves et obtenir des traces cohérentes, chaque entité a été associée à un `Role` prédéfini. De plus, nous avons exigé l'unicité de certaines actions par la contrainte `Only_One`. Ces contraintes sont dénommées `axioms` dans le code. Elles permettent d'associer une chaîne de caractères fixe à une variable (comme le nom du rôle), ou encore d'avoir des égalités sur les moments d'exécution de certaines actions possiblement répétées. Par conséquent, en utilisant un unique coeur d'un *i5 4590@3.50Ghz* avec *8Go RAM*, nous obtenons de bonnes performances sur les différentes preuves des lemmes (voir le tableau 1).

Lemmes	Temps `CPU`
Correctness :	4.7 s
Soundness :	4.4 s
Connection integrity :	10.9 s
Secrecy :	6.6 s

Table 1 – Temps de calculs nécessaires à Tamarin pour prouver les lemmes.

4 ■ DÉPLOIEMENT DE LOCALPKI

4.1 | DÉPLOIEMENT DE LOCALPKI À PARTIR D'UNE PKIX

LocalPKI présente l'avantage d'être facilement mise en place à partir d'une

installation de PKIX. En effet, toutes les prérequis de LocalPKI peuvent être vérifiés en adaptant les standards actuels de PKIX. Dans la suite, nous présentons comment les réaliser en pratique.

- Tout d'abord, les certificats utilisés dans LocalPKI s'approprient le format des certificats X.509v3 [Coo08]. En effet, les deux PKI (LocalPKI et PKIX) partagent le même type de données d'authentification (TBSCert). La signature de l'AC est remplacée par celle de l'utilisateur, et le SN peut être enregistré dans le champ `serialNumber`.
- Les communications avec le notaire pendant l'authentification dans le mode privé (figure 2) peuvent être mises en place en utilisant la norme OCSP [SAM+13]. Dans une requête OCSP, le SN remplace l'actuel `serialNumber` dans la séquence `CertID`, et le SI est stocké dans le champ optionnel de `Signature`. Les autres champs dans `CertID`, tels que `issuerNameHash` et `issuerKeyHash` peuvent être laissés vides puisque les informations qu'ils contiennent sont redondantes, ou peu pertinentes avec le SI. Le nonce R_A (voir la figure (2), et l'algorithme (11)) peut être stocké dans le champ `requestExtensions`.
- De manière similaire, la réponse du notaire est identique à une réponse OCSP, où tous les champs sont communs.
- Dans le mode public, les *Certificate Verification Lists* peuvent être gérées comme les *Certificate Revocation Lists*. Les SN et SI seraient alors respectivement stockés dans le champ `CertificateSerialNumber`, et dans une extension (`crlEntryExtensions` ([Coo08]). Par exemple, une CVL peut être postée sur le site du NE, et enregistrée dans un répertoire local. Dans le cas où une gestion des sous-domaines est mise en place, c.-à-d. que chaque intervalle de SN représente un sous-domaine, l'indicateur des δ-CRL peut être utilisé [Coo08], comme le champ `BaseCRLNumber` pourrait représenter un équivalent du *Base CVL Number Field*.
- Les requêtes de révocation de LocalPKI et PKIX sont quasi identiques. Par conséquent, LocalPKI peut directement s'appuyer sur le protocole *Certificate Management Protocol* [KP12], et en particulier le champ `RevReqContent`, pour les révocations.
- Finalement, PKIX requiert l'utilisation d'ancres de confiance [RW10], par exemple dans un magasin du système d'exploitation ou du navigateur de l'utilisateur. Les communications entre le LRA et le notaire reposent aussi sur l'utilisation des ancres de confiance, où un mécanisme comme PKIX peut directement être employé.

Ainsi, des outils existants tel que *OpenSSL* peuvent permettre à toutes les entités de générer leurs clefs, certificats, requêtes et réponses de révocation. La mise en place technique de LocalPKI est principalement restreinte à une gestion de

base de données. Cette tâche est déléguée aux notaires, qui sont alors en charge de maintenir la disponibilité de la PKI. Les LRA sont responsables des communications avec le notaire, c.-à-d. d'un échange de numéros de série, ainsi que de la vérification d'identité des nouveaux utilisateurs souhaitant s'enregistrer.

Selon le niveau de connaissance des utilisateurs, ils ont plusieurs possibilités :

- **Utilisateurs Experts :** Ils génèrent eux-mêmes leur paire de clefs. Ainsi, ils font la demande au LRA d'un numéro de série, puis créent et signent leur certificat. Ils fournissent ensuite le *SI* associé avant de le faire parvenir au LRA.
- **Utilisateurs Intermédiaires :** Les utilisateurs génèrent eux-mêmes leur paire de clefs. Le LRA se charge de la création du certificat, puis leurs fournit ensuite un moyen simple de le signer (par exemple un port USB et un clavier leur permettant de taper le mot de passe pour déchiffrer la clef privée stockée sur une clef USB).
- **Utilisateurs Novices :** Le LRA génère la paire de clef pour l'utilisateur, et lui fournit l'ensemble des produits (clefs et certificats).

Une partie ou l'ensemble de ces possibilités peuvent être réalisées à l'aide d'un site web dédié. Le seul prérequis technique pour le LRA est l'utilisation d'un outil comme *OpenSSL* et une création d'une interface conviviale, afin d'aider les utilisateurs dans leur enregistrement.

4.2 | COMPARAISON DES DEUX INFRASTRUCTURES

4.2.1 • DIFFÉRENCES FONCTIONNELLES

Les principales différences entre PKIX et LocalPKI sont les suivantes :

- Dans LocalPKI, les autorités d'enregistrement n'ont pas besoin d'être expertes en sécurité. Ainsi, elles peuvent être proches des utilisateurs et faciliter l'utilisation des certificats dans leur quotidien (voir § 2.2) ;
- La création des certificats est réalisée par l'AC dans PKIX ; dans LocalPKI cette action est réalisée par le LRA ou l'utilisateur, tandis que le notaire le stocke et le rend disponible (voir § 2.2) ;
- Le mode d'authentification par défaut de PKIX utilise les CRL, tandis que celui de LocalPKI est intéractif, avec un protocole semblable à OCSP (voir § 2.3.1).
- Le mode alternatif d'authentification de PKIX est OCSP, tandis que LocalPKI propose un mécanisme alternatif dénommé *Certificate Verification Lists* (décrit dans le paragraphe 2.3.2)

En observant l'exécution des deux protocoles, la principale différence se situe dans la signature du certificat utilisateur : dans PKIX, la signature d'une AC est présente alors que dans LocalPKI seule la signature de l'utilisateur est présente.

Comme nous pouvons le voir dans la figure 5, la création de certificat est réalisée en **deux** étapes par son propriétaire, au lieu de **quatre** par l'AC dans PKIX (figure 4). De plus, les autorités d'enregistrement transmettent le certificat complet au notaire alors que le LRA fournit simplement un haché du certificat. Ainsi, dans LocalPKI les certificats ne sont pas directement publiés puisque la base de données du notaire contient uniquement des empreintes.

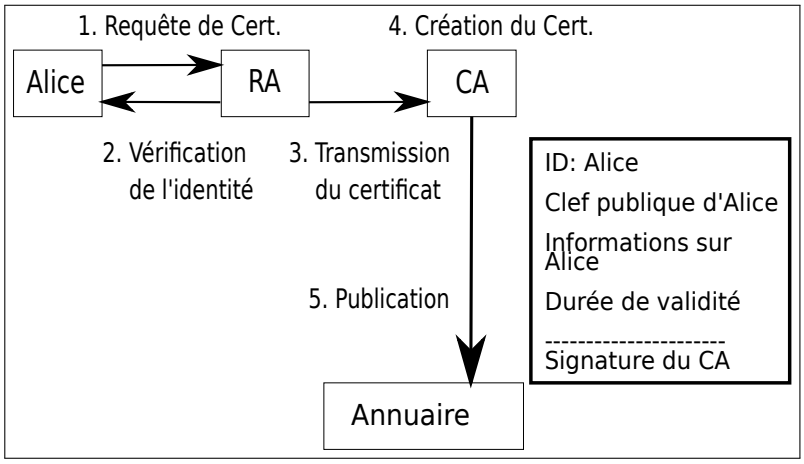

FIGURE 4 – Procédure d'enregistrement de PKIX.

Le mécanisme d'authentification est aussi différent, comme les figures (6) et (7) le montrent : dans LocalPKI, les propriétaires doivent fournir leur certificat au préalable, et les utilisateurs interagissent directement avec le notaire pour être convaincus de la validité du certificat. Dans PKIX, les utilisateurs récupèrent le certificat dans un dépôt local et interagissent ensuite avec un répondeur OCSP pour effectuer la vérification.

4.2.2 • COMPARAISON ENTRE LES MODES PUBLIC, PRIVÉ ET LES MÉCANISMES DE PKIX

LocalPKI est conçue pour utiliser le mode privé par défaut. Il permet aux utilisateurs d'avoir accès à une base de données à jour, et son coût de communication dépend du nombre de requête au lieu d'être périodique comme dans le mode public. Cependant, les CVL sont intéressantes dans le cas où la communication avec un notaire n'est pas toujours possible. Il est possible de faire un parallèle de ces deux modes avec les mécanismes de PKIX, où le mode privé peut être associé à OCSP et les CVL avec les CRL.

Initialement, OCSP n'a pas été étudié pour résister aux *replay attacks*. C'est dans une version plus récente, voir [SAM+13, § 4.4.1], que le champ `responseExtensions` a été ajouté comme contre-mesure. Cependant, comme précisé dans [SAM+13, § 5],

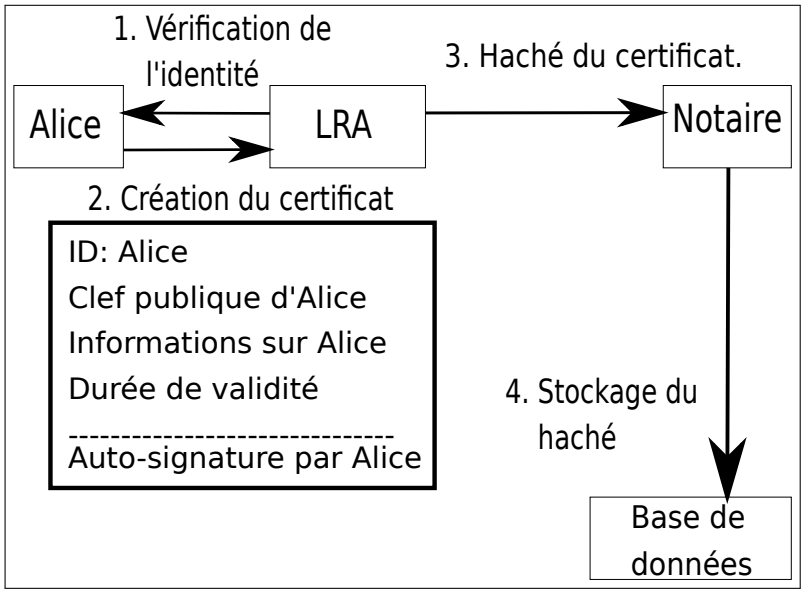

FIGURE 5 – Procédure d'enregistrement de LOCALPKI.

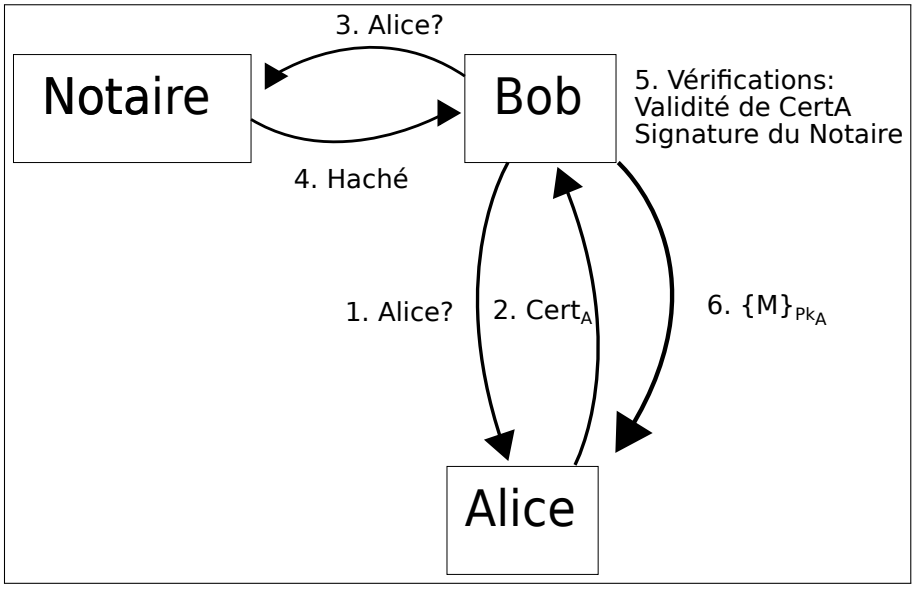

FIGURE 6 – Procédure interactive de vérification du statut des certificats de LOCALPKI.

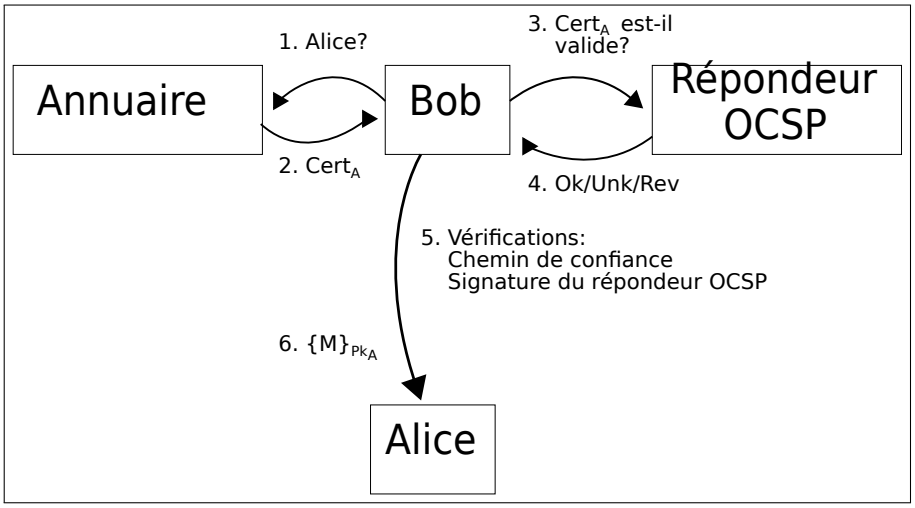

FIGURE 7 – Procédure interactive de vérification du statut des certificats de PKIX

cette solution n'est pas nécessaire pour respecter la norme, *a contrario* du mode privé de LOCALPKI, où les attaques par rejeu sont contrées par défaut grâce à un *Number used Once* (nonce).

La comparaison des CRL de PKIX et des CVL de LOCALPKI est analogue à celle entre les mécanismes de listes noire et blanche. Avec les CRL, qui fonctionnent comme une liste noire contenant les certificats révoqués, la liste complète doit être échangée, impliquant un coût de communication important. Une solution, nommée δ-CRL, permet de réduire ce coût (voir la section II.4.2). De manière similaire, nous proposons d'utiliser les sous-domaines dans LOCALPKI. Ainsi, seule une partie de la liste doit être partagée pour obtenir une authentification correcte. Par exemple, un utilisateur peut décider de récupérer uniquement la CVL dont le sous-domaine est un intervalle des numéros de série qui l'intéresse.

A la différence des δ-CRL, les sous-domaines des CRL ont l'avantage d'offrir une meilleure tolérance face à une récupération partielle des listes. En effet, si utilisateur n'obtient qu'une CRL partielle, il peut alors accepter un certificat révoqué. Au contraire, un certificat non présent dans un CVL n'est simplement pas accepté même s'il avait pu être valide. Notons enfin qu'un mécanisme équivalent aux δ-CRL peut aussi être mis en place : une CVL récupérée à un instant t_1 peut être complétée uniquement par les entrées des nouveaux certificats validés en $t_2 > t_1$. Cette méthode peut aussi être associée aux CVL de sous-domaines, où les seules mises à jour sont faites si et seulement si un nouveau *SN* appartenant à son intervalle de définition est ajouté.

De plus, une solution reposant sur les CRL expose les utilisateurs à des authentifications dites fausse-positives. Comme une CRL n'est pas mise à jour continuellement, un certificat récemment révoqué pourra toujours être considéré comme

valide. En utilisant une stratégie reposant sur une liste blanche comme avec les CVL, les vérificateurs ne pourront peut-être pas réussir à authentifier un utilisateur nouvellement enregistré. Une fois encore, obtenir des faux négatifs pour l'authentification est généralement plus sûr que des faux positifs. Enfin, sur une période donnée, la taille d'une CVL n'est pas condamnée à augmenter, elle ne dépend que du nombre d'entités puisqu'un certificat révoqué est supprimé. Dans le cas des CRL, une entité peut révoquer plusieurs certificats : le nombre de certificats n'est donc borné que par leur date d'expiration.

4.3 | APPLICATION DE LOCALPKI À UN MODULE DE RUPTURE PROTOCOLAIRE

Pour conclure la comparaison entre les deux types de PKI, nous allons montrer comment gérer les clefs lors d'un déploiement d'un module de rupture protocolaire avec une instance de LocalPKI. Cette partie est donc à mettre en parallèle avec le chapitre précédent.

Dans un premier temps, nous devons attribuer chaque rôle à une entité prenant part au cycle de vie du module. Comme nous l'avons vu précédemment, le module est utilisé afin de sécuriser des échanges au sein d'un réseau. Le client, tel que défini en (III.2.3), est celui qui souhaite installer le module dans son infrastructure. De manière générale, nous ne pouvons donc pas supposer qu'il est expert en PKI, ou encore qu'une PKI est déployée. Par contre, il est capable de se connecter au module afin d'y effectuer les tâches d'administration. De son côté, le fabricant, ou éventuellement l'intégrateur, a la capacité d'en fournir une. A partir de ces besoins, nous pouvons donc définir une hiérarchie (voir la figure 8), analogue à celle définie dans LocalPKI :

- Le fabricant (ou l'intégrateur) est en charge de la gestion du notaire électronique ;
- Le client, ou plus précisément l'administrateur du module, jouera le rôle du LRA ;
- Le module est finalement vu comme l'utilisateur final, qui souhaite certifier ses clefs.

Lors de l'installation du module, le fabricant donne un ensemble de numéros de série (*Serial Number* (SN)) à l'administrateur. Par la suite, l'administrateur les répartit simplement lorsqu'il se connecte sur les modules. En tant que *LRA*, l'administrateur va utiliser ses droits pour faire générer des clefs au module, suivi de son certificat LocalPKI. Cette méthode est en fait un équivalent de l'enregistrement dans LocalPKI, puisqu'en procédant ainsi, il vérifie la clef publique associée à un module. Il récupère finalement le couple composé du numéro de série et de la signature. Dans le cas où de multiples modules doivent être configurés,

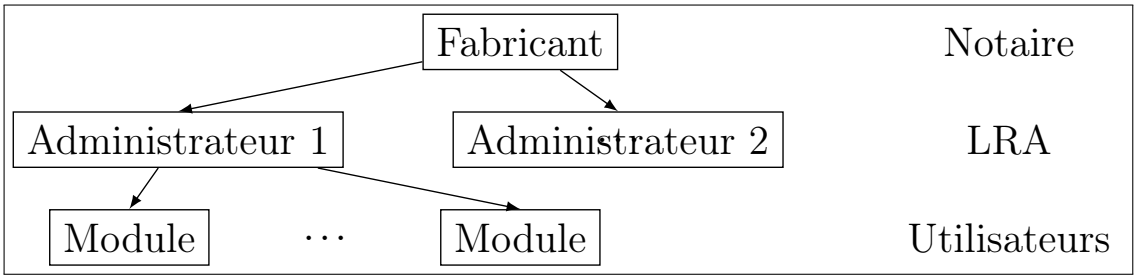

FIGURE 8 – LOCALPKI appliquée au déploiement d'un module de rupture protocolaire

remarquons que LOCALPKI offre à l'administrateur l'avantage de pouvoir procéder par blocs : il peut configurer l'ensemble des modules, et envoyer ensuite un lot contenant tous les nouveaux couples à stocker dans la base de données du notaire. Il suffit ensuite au fabricant de vérifier que les numéros de série contenus dans le lot sont conformes à ceux qu'il a distribués au client, avant de les ajouter à sa base de données. Précisons que dans le cas où le client a les capacités, il peut bien évidemment mettre en place son propre notaire. Finalement, les modules n'ont plus qu'à échanger leur certificat, puis vérifier leur statut pour établir des communications sécurisées. La vérification du statut se fait en utilisant l'un des deux modes (privé ou public) décrit précédemment. Dans ce cadre, le découpage par sous-domaine devient particulièrement utile. Supposons que le fabricant (donc le notaire) ait un client dont l'infrastructure est composée de plusieurs sites. Il aura donc une base de données contenant des certificats délivrés pour des entités localisées sur des sites différents. En partageant uniquement la CVL associée à un des emplacements, le volume des données échangées est restreint. Il reste cependant aisé de pouvoir authentifier deux éléments appartenant à deux emplacements différents : il suffit de partager la CVL des deux sites.

En comparaison avec PKIX, la mise en place des modules est plus rapide : une fois les certificats générés par le module, il suffit que l'administrateur envoie les couples au notaire pour que l'ensemble soit fonctionnel. Avec PKIX, une fois les certificats générés, il faut attendre que l'autorité de certification les signe. Avec LOCALPKI, aucun échange supplémentaire n'est requis de la part des modules. L'aspect asynchrone de LOCALPKI est donc particulièrement intéressant dans ce type d'architecture. Enfin, entre deux mises à jour des listes de certificats (c.-à-d. les CVL ou CRL), notre PKI offre une meilleure robustesse. D'un côté, l'utilisation de listes blanches nécessite une mise à jour de la base de données pour établir de nouvelles communications sécurisées. D'un autre côté, cela permet d'avoir un contrôle d'accès plus fin sur les appareils présents sur le réseau, alors que dans le cas de PKIX, tous les nouveaux certificats sont valides par défaut.

5 ■ CONCLUSION

Dans ce chapitre nous avons donc proposé un modèle alternatif pour les infrastructures à clefs publiques. Ce dernier a ensuite été analysé et sa sécurité formellement prouvée à l'aide de Tamarin. Pour rappel, la principale différence avec PKIX se situe dans l'utilisation de certificats auto-signés exclusivement, où le lien entre l'identité et le certificat peut être vérifié via une requête à un tiers de confiance, le notaire. L'utilisation globale des certificats par les utilisateurs finaux est facilitée : d'une part lors de l'enregistrement, réalisée dans une agence de proximité, et par la simplification de la vérification d'autre part. De plus, comme la phase d'enrôlement des nouveaux utilisateurs ne requiert pas de connaissance technique du LRA, LocalPKI pourrait participer à un accroissement global de l'utilisation des certificats par des utilisateurs ordinaires. De plus, LocalPKI pourrait s'inscrire dans le modèle économique actuel, puisque les notaires sont majoritairement en charge de la partie technique. Ils pourraient donc, à l'instar des AC actuelles, facturer les services assurant la gestion et la disponibilité de la base de données d'empreintes. D'un point de vue de la sécurité, LocalPKI offre des garanties similaires à celles offertes par PKIX, à savoir que les entités hors utilisateurs sont supposées de confiance. Enfin, comme nous l'avons vu dans la section (4.1), le déploiement de LocalPKI peut être effectué en adaptant simplement les outils déjà développés pour PKIX.

Dans certains cas, une gestion des clefs à la manière de LocalPKI peut être plus adaptée que PKIX. Par exemple, les clients d'une banque ont souvent la possibilité de gérer leur compte en ligne, parfois en utilisant une authentification à base de certificat. La gestion d'une PKI classique nécessite un département spécialisé ou une délégation complète de la confiance envers une entité externe. En déployant LocalPKI, la banque préserve la gestion de l'enregistrement de ses clients via ses agences locales, soit la confiance et la responsabilité de ces clients. La partie technique assurant l'authentification et la disponibilité est déportée sur le notaire. Ainsi, le coût global et les prérequis techniques sont réduits pour l'entreprise. Au-delà de l'exemple de la banque, LocalPKI pourrait permettre à de petites agences de mettre en place des services nécessitant une authentification en participant à l'enregistrement de leurs utilisateurs. Le coût de ce nouveau service serait raisonnable en comparaison de celui nécessaire pour la mise en place d'une PKI classique.

Dans le cadre d'un module de sécurité comme ARAMIS, LocalPKI correspond mieux aux besoins qu'une PKI classique. En assimilant le boîtier au rôle d'un utilisateur générant lui-même son certificat, il suffirait qu'il se fasse ensuite enregistrer par le client (c.-à-d. l'entreprise recevant le module) à l'aide d'outils fournis par le fabriquant du module. Lors d'une connexion au boîtier, les utilisateurs vérifient

eux-mêmes la validité du certificat en envoyant une requête au notaire. Le notaire peut être géré de manière externe, ou la base de données peut être partagée afin d'être distribuée au client. Inversement, plutôt qu'intégrer au module la liste des certificats révoqués, seule la liste des certificats validés pourrait être mise en place dans le module. Ainsi, le contrôle sur les authentifications de la part du module est complet, en évitant une acceptation par défaut d'un certificat non présent dans la liste des certificats révoqués (comme c'est le cas dans PKIX).

En guise de perspectives, l'aspect légal sur le partage des responsabilités entre le NE et le LRA devrait être étudié, puisque les deux entités jouent un rôle important dans les processus d'enregistrement et d'authentification. A titre de première ébauche, le NE pourrait enregistrer l'identité du LRA en envoyant les informations à propos du nouvel utilisateur, et le stocker dans sa base de données. Ainsi, si cet utilisateur est ensuite prouvé comme malfaisant, le NE pourrait blâmer le LRA.

Une autre perspective pourrait être à propos de l'adaptation de ce modèle pour les authentifications utilisant l'identité (telles que les PKI sans certificat [Gen03, BSS05]). Enfin, comme *Let's Encrypt* permet d'établir une connexion sécurisée pour les sites internet, il serait intéressant de développer un site permettant de générer un certificat auto-signé pour n'importe quel utilisateur.

CHAPITRE V

APPLICATION DES PRODUITS SCALAIRE ET MATRICIEL DISTRIBUÉS ET SÛRS À L'AGRÉGATION DE CONFIANCE

Sommaire

RÉSUMÉ

Dans ce chapitre, nous nous intéressons aux calculs multi-parties privés appliqués à l'agrégation de confiance. Cette dernière se réduit à un calcul de produits matriciels, mais dont la distribution des données est peu conventionnelle : chaque joueur ne connaît qu'une ligne de chacune des matrices. Dans ces conditions, chaque joueur souhaite apprendre une ligne du résultat (*i.e.* du produit), sans révéler ses entrées aux autre joueurs. Nous commençons par améliorer un protocole existant, calculant une moyenne pondérée, pour calculer un produit scalaire ayant un volume quadratique de communication et nombre de tours linéaire. Nous proposons ensuite deux protocoles duaux nécessitant cinq tours de communication, et qui utilisent le cryptosystème de Paillier. A l'aide d'outils de vérification automatique, tels que ProVerif, nous exposons des contre-mesures pour chacune des attaques trouvée par l'outil. Nous obtenons donc des protocoles privées contre des adversaires semi-honnêtes et malicieux. Nous donnons aussi une méthode générique permettant d'éviter des attaques en cas de coalition. Enfin, nous montrons comment appliquer ces protocoles à l'agrégation de confiance. Les publications associées à ce chapitre sont [DLOP16, DLOP17].

ORGANISATION

Nous commençons par rappeler en détails le fonctionnement de l'agrégation de confiance dans un réseau (2), avant d'expliciter la réduction de notre problématique à un produit scalaire multi-parties. Nous poursuivons par les descriptions et analyses de sécurité de trois protocoles dans les sections (3), (4), (5) qui permettent de calculer des produits scalaires de manière privée. Nous comparons ensuite leurs approches et efficacités. Enfin, nous montrons comment utiliser ces protocoles dans le cadre de l'agrégation de la confiance.

1 ■ EVALUATION DE LA CONFIANCE DES AUTORITÉS DE CERTIFICATION

Comme nous l'avons vu précédemment avec PKIX ou LocalPKI, la vérification d'un certificat repose en partie sur l'utilisation *d'ancres de confiance* (en anglais, *trust anchor*). Lors de l'authentification d'une entité employant un certificat, il faut s'assurer que le certificat de l'autorité de confiance est présent dans le magasin d'ancres de confiance, et que ce certificat est valide. Ce dernier est **auto-signé** par son propriétaire, et sa vérification consiste à utiliser la clef publique contenue à l'intérieur du certificat pour s'assurer de l'intégrité des données qu'il contient. Si la signature est correcte, et que le certificat du propriétaire est valide, alors l'entité est authentifiée. Ainsi, la gestion du magasin d'ancres de confiance est critique : si le certificat d'une autorité malhonnête est ajouté à l'ancre de confiance, tous les propriétaires de certificats délivrés par cette autorité pourront être authentifiés. Pourtant, les ancres de confiance sont d'ordinaire employées de manière transparente, et la responsabilité de l'administration du contenu du magasin incombe rarement à l'utilisateur. Parmi les exemples d'application reposant sur l'utilisation de ce mécanismes, les plus connus sont probablement la vérification que l'éditeur d'une application est de confiance pour les systèmes d'exploitation, ou encore l'authentification d'un site web lors d'une connexion sécurisée par le navigateur. Dans les deux cas, l'utilisateur fait alors confiance à l'éditeur du programme pour gérer les certificats contenus dans l'ancre.

D'après [DKBH13] ou [cen], il existe entre 500 et 700 autorités de certification racines, qui peuvent donc potentiellement être reconnues comme ancres de confiance. Chacune d'entre elles émet, par définition, un grand nombre de certificats. Or, lorsqu'un certificat est émis à une entité reconnue par la suite comme frauduleuse, seul ce dernier est révoqué, alors que le certificat de l'autorité reste toujours valide. Malgré des pratiques démontrant des faiblesses dans le mode opératoire de l'AC, les utilisateurs l'ayant comme ancre de confiance continueront d'authentifier les entités qu'elle a certifiées. Ce fonctionnement s'est récemment révélé problématique, comme l'exemple entre l'AC *WoSign* et le navigateur *Mozilla Firefox* [Moz].

Nous souhaitons donc considérer une nouvelle méthode, permettant d'évaluer la confiance que l'utilisateur peut avoir dans une autorité de certification en fonction de ses actes. L'approche que nous avons choisie est d'utiliser un modèle de confiance plus complexe que celui du magasin d'ancres de confiance. Actuellement, ce dernier est binaire : une AC est de confiance si et seulement si elle est dans le magasin. En d'autres termes, nous souhaitons réaliser une évaluation de la confiance au sens d'un réseau, composé des AC. Pour cela, des méthodes reposant sur la transitivité d'une mesure simple, symbolisant la confiance, ont étés explorées [FAO10], et parfois même appliquées aux PKI [HN10]. D'autres solutions employant une évaluation

de la confiance ont été étudiées [GKRT04] : dans ce cas, la valeur de la confiance est considérée comme la probabilité qu'un évènement survienne, et est associée à une seconde valeur, l'incertitude. Cette dernière représente la probabilité que l'évènement opposé à celui attendu se produise. D'autres approches offrent une mesure encore plus fine, **la logique subjective** [Jøs07]. Cette dernière est composée de trois composantes, qui sont la confiance, la méfiance et l'incertitude. De plus, dans [DH12], les auteurs ont montré que l'évaluation de la confiance dans un réseau via ce modèle peut se réduire à un calcul de produits matriciels (défini sur deux monoïdes) entre les entités du réseau.

Cependant, les informations possédées par chacun des noeuds sont extrêmement sensibles : elles représentent la confiance qu'ils ont pour les autres noeuds et doivent donc rester secrètes. Nous sommes donc dans une situation où plusieurs entités souhaitent participer à un calcul, celui de l'agrégation de la confiance dans le réseau qu'elles forment, mais où aucune d'entre elles ne peut révéler ses entrées aux autres. Cette formulation est exactement celle d'un problème de calcul multi-parties (II.7). Finalement, notre problématique est la suivante :

A partir du modèle de confiance de [Jøs07] et de sa réduction à un produit matriciel [DLR15], comment calculer l'agrégation de confiance dans un réseau de manière distribuée et privée ?

2 ■ AGRÉGATION DE CONFIANCE ET PRODUIT MATRICIEL

Dans cette partie, nous rappelons des résultats obtenus dans [DH12]. Nous commençons par détailler le modèle de confiance utilisé, avant de montrer que le calcul de la confiance dans un graphe peut être effectué via un produit matriciel.

2.1 | MODÈLE DE CONFIANCE

Nous allons utiliser un modèle où la confiance est représentée par un triplet (*Confiance*, *Méfiance*, *Incertitude*), à l'instar de celui utilisé dans [Jøs07,HN10,DH12]. Les quantités sont évaluées de la manière suivante :

- $\mathcal{T}$ = *Confiance* (*Trust*) : Proportion d'expériences prouvées (ou espérées) positives ;
- $\mathcal{D}$ = *Méfiance* (*Distrust*) : Proportion d'expériences prouvées négatives ;
- $\mathcal{U}$ = *Incertitude* (*Uncertainty*) : Proportion d'expériences au caractère indéterminé. Sa valeur dépend des deux précédentes, *i.e.* $\mathcal{U} = 1 - \mathcal{T} - \mathcal{D}$.

Comme l'*Incertitude* est entièrement déterminée par la *Confiance* et la *Méfiance*, il est donc possible d'exprimer la confiance à l'aide du couple de ces deux derniers coefficients. Dans la suite, nous allons donc considérer que la confiance est donnée sous la forme d'une paire $\langle a,\ b\rangle \in \mathbb{D}^2$, où $\mathbb{D}$ est un anneau intègre.

2.1.1 • AGRÉGATION SÉQUENTIELLE DE LA CONFIANCE

Soient trois joueurs, Alice, Bob et Charlie, tels que Alice ait un certain degré de confiance envers Bob, et que Bob ait un certain degré de confiance envers Charlie. L'agrégation séquentielle de la confiance permet de formaliser la transitivité de la confiance. A l'issue de ce calcul, Alice obtiendra une opinion à propos de Charlie en utilisant celle de Bob.

Définition 14 (Agrégation séquentielle de la confiance)**.** *Soient trois joueurs P_1, P_2 et P_3, tels que la confiance de P_1 en P_2 est égale à $\langle a,\ b\rangle \in \mathbb{D}^2$, et celle de P_2 envers P_3 vaut $\langle c,\ d\rangle \in \mathbb{D}^2$. L'agrégation séquentielle de la confiance est une fonction $\bigstar : \mathbb{D}^2 \times \mathbb{D}^2 \to \mathbb{D}^2$ donnant la valeur de la confiance obtenue pour le chemin $P_1 \overset{\langle a,\ b\rangle}{\to} P_2 \overset{\langle c,\ d\rangle}{\to} P_3$ comme :*

$$\langle a,\ b\rangle \bigstar \langle c,\ d\rangle = \langle ac+bd,\ ad+bc\rangle.$$

2.1.2 • AGRÉGATION PARALLÈLE DE LA CONFIANCE

Il est aussi possible qu'il existe plusieurs chemin de confiance entre Alice et Charlie. Dans ce cas, nous pouvons définir une agrégation parallèle :

Définition 15 (Agrégation parallèle de la confiance)**.** *Soient deux chemins disjoints $P_1 \overset{\langle a,\ b\rangle}{\to} P_3$ et $P_1 \overset{\langle c,\ d\rangle}{\to} P_3$. L'agrégation parallèle de la confiance de ces deux chemins est une fonction $\maltese : \mathbb{D}^2 \times \mathbb{D}^2 \to \mathbb{D}^2$ calculant la valeur de la confiance obtenue comme*

$$\langle a,\ b\rangle \maltese \langle c,\ d\rangle = \langle a+c-ac,\ bd\rangle.$$

L'opérateur $\maltese$ vérifie les propriétés suivantes :

i. $\maltese$ est commutatif, car $\langle a,\ b\rangle \maltese \langle c,\ d\rangle = \langle a+c-ac,\ bd\rangle = \langle c+a-ca,\ db\rangle = \langle c,\ d\rangle \maltese \langle a,\ b\rangle$;

ii. L'élément neutre est le couple $\langle 0,\ 1\rangle$, car $\langle a+0-a.0,\ b.1\rangle = \langle a,\ b\rangle$.

Nous avons le lemme suivant :

Lemme 16. *Le couple $\langle a,\ b\rangle$ est inversible si et seulement si b est inversible dans $\mathbb{D}$ et ($a = 0$ ou $a-1$ est inversible).*

Démonstration. Commençons par remarquer que le couple $\langle 0,\ 1\rangle$ représente l'élément neutre pour ✠ (car $\langle a+0-a.0,\ b.1\rangle = \langle a,\ b\rangle$). Alors, pour b inversible, si $a=0$, alors le couple $\langle 0,\ b^{-1}\rangle$ est un inverse pour le couple $\langle 0,\ b\rangle$. Dans le cas où $a-1$ est inversible :

$$\begin{aligned}\langle a(a-1)^{-1},\ b^{-1}\rangle ✠ \langle a,\ b\rangle &= \langle a,\ b\rangle ✠ \langle a(a-1)^{-1},\ b^{-1}\rangle \\ &= \langle a+a(a-1)^{-1}-a^2(a-1)^{-1},\ bb^{-1}\rangle \\ &= \langle 0,\ 1\rangle.\end{aligned}$$

De même, si $\langle a,\ b\rangle ✠ \langle c,\ d\rangle = \langle 0,\ 1\rangle$, alors $bd=1$ et b est inversible. On a alors $(a-1)c=a$. Finalement, si $a\neq 0$ et $a-1$ est un diviseur de zéro, il existe $\lambda\neq 0$ tel que $\lambda(a-1)=0$, et donc $\lambda(a-1)c=0=\lambda a$. On a alors $\lambda(a-1)-\lambda a=-\lambda=0$. Par contradiction, nous obtenons donc que $a=0$ ou $a-1$ est inversible. □

2.2 | AGRÉGATION DE CONFIANCE PAR PRODUIT MATRICIEL

Dans [DH12], les auteurs montrent que l'agrégation de confiance dans un graphe peut être vue comme un produit matriciel sur $\mathbb{D}$, avec les deux opérations précédemment définies. Soit n le nombre de joueur. Soit $M\in(\mathbb{D}^2)^{n\times n}$ une matrice de confiance telle que chaque ligne i contienne les valeurs de confiance de P_i envers les autres joueurs.

P_1	$\langle 1,\ 0\rangle$	$\langle \mathcal{T}_2,\ \mathcal{D}_2\rangle_1$	$\cdots$	$\langle \mathcal{T}_n,\ \mathcal{D}_n\rangle_1$
P_2	$\langle \mathcal{T}_1,\ \mathcal{D}_1\rangle_2$	$\langle 1,\ 0\rangle$	$\cdots$	$\langle \mathcal{T}_n,\ \mathcal{D}_n\rangle_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
P_n	$\langle \mathcal{T}_1,\ \mathcal{D}_1\rangle_n$	$\langle \mathcal{T}_2,\ \mathcal{D}_2\rangle_n$	$\cdots$	$\langle 1,\ 0\rangle$

TABLE 1 – Matrice de confiance.

Nous supposons que chacun des joueurs a une confiance totale en lui-même, ce qui se caractérise par le couple $\langle 1,\ 0\rangle$ dès que $i=j$.

Définition 17 (Produit de matrices de confiance [DH12])**.** *Soient A et B deux matrices de confiance construites sur un ensemble $(\mathbb{D}^2)^{n\times n}$. Le produit de matrices est défini tel que :*

$$C_{ij} = ✠_{k=1}^{n} A_{ik} ★ B_{kj}$$

Ce produit de matrices permet en fait de calculer l'agrégation de confiance dans un graphe. En supposant que $M\in(\mathbb{D}^2)^{n\times n}$ est une matrice de confiance, le calcul de k puissances successives permet d'obtenir une agrégation de la confiance

à une distance k de chacun des noeuds. Les auteurs précisent qu'en cas de cycle, il existe une limite finie d telle que l'agrégation complète du graphe converge après d itérations du produit.

Grâce à cette modélisation, nous pouvons donc réduire le problème de l'agrégation de confiance aux calculs de puissances d'une matrice carrée. Pour effectuer le produit d'une matrice, nous devons être capable de calculer n^2 produits scalaires, c.-à-d. n pour chaque ligne de la matrice. Or, comme nous pouvons le voir sur la table (1), chaque joueur possède une ligne d'une matrice. Le produit scalaire étant effectué entre une ligne et une colonne, le problème se reformule ainsi : n joueurs souhaitent calculer un produit scalaire $S = \vec{u}^T\vec{v}$, où le vecteur $\vec{u} \in \mathbb{D}^2$ est détenu par joueur P_i, tandis que le vecteur $\vec{v} \in \mathbb{D}^2$ est distribué entre les joueurs tel que chaque composante v_i est possédée par le joueur P_i, $i \in [1,n]$.

Afin de simplifier le problème, nous nous intéressons dans un premier temps au calcul d'un produit scalaire classique *i.e.* $\vec{u},\vec{v} \in \mathbb{Z}$. Le problème du produit scalaire multi-parties a déjà été largement étudié [DA01, AEC07, WhSsH+08]. Cependant, dans ces papiers, la répartition des données est différente : chaque joueur possède un vecteur complet, et le produit scalaire est calculé entre deux joueurs uniquement. Enfin, les protocoles génériques, reposant généralement sur l'évaluation de circuits, requièrent des coûts de calculs et de communications de l'ordre de $O(n^3)$ par produit scalaire [BDOZ11, DPSZ12].

Par la suite, nous présentons donc trois protocoles permettant de répondre à cette problématique de manière plus efficace, ainsi que les méthodes pour les appliquer à un produit de matrices classiques. Nous explicitons ensuite comment adapter ces protocoles à un produit de matrices de confiance.

3 ■ DU PROTOCOLE MPWP À P-MPWP

Comme nous l'avons vu précédemment, nous voulons calculer un produit scalaire multi-parties, avec une répartition des données telle qu'un joueur possède un vecteur complet et que les composantes de l'autre vecteur soient distribuées entre tous les joueurs. Le protocole MPWP [DGK10] semble répondre à cette problématique, au prix de complexités en calculs et en communications importantes. Nous commençons par présenter le protocole MPWP, puis nous explicitons une version allégée dénommée P-MPWP.

3.1 | MPWP : UN PROTOCOLE DE CALCUL D'UNE MOYENNE PONDÉRÉE DISTRIBUÉ

Le protocole *Multiple Private Keys Weighted Protocol* (MPWP) défini dans [DGK10], est utilisé pour calculer la confiance de manière sécurisée dans un système de réputa-

tion additif comprenant n joueurs. Tous les joueurs P_i (sauf le joueur maître P_1) ont une entrée secrète v_i. P_1 possède un vecteur $\vec{u}$, où chaque composant u_i est le poids affecté à chacun des autres joueurs. Le but est de calculer la confiance moyenne pondérée, soit $\sum_{i=2}^{n} u_i * v_i$. MPWP fonctionne de la manière suivante : le premier joueur crée un vecteur $\vec{t}$ contenant ses entrées secrètes chiffrées à l'aide du cryptosystème de Benaloh, soit $\vec{t} = [E_1(u_2), \ldots, E_1(u_n)]$. P_1 initialise aussi l'ensemble des coefficients à 1 d'une matrice M de taille $(n-1) \times (n-1)$, ainsi qu'une variable d'accumulation a initialisée à 1. Le premier tour du protocole suit une topologie circulaire ; P_1 débute le tour en envoyant le triplet $(M, \vec{t}, a)$ au joueur P_2 qui, après modifications, le renverra à P_3... Après réception, un joueur P_i calcule $a = a * t_i^{v_i} * E_1(z_i)$, où t_i est la i^{me} coordonnée de $\vec{t}$ (soit $t_i = E_1(u_i)$) et z_i est une valeur aléatoire précédemment tirée par ce joueur. Ensuite tous les joueurs, excepté P_1, vont découper leur aléa z_i en $n-1$ parts à valeur positive, telles que $z_i = \sum_{j=2}^{n} z_{i,j}$. Chaque part $z_{i,j}$ est ensuite chiffrée avec la clef publique de P_j avant d'être stockée dans la i^{e} colonne de M. A la fin du premier tour, P_1 obtient donc $D_1(a) = \sum_{i=2}^{n} u_i v_i + z_i$, ainsi que la matrice M contenant les parts de tous les joueurs chiffrées, $M = (m_{i,j}) = (E_j(z_{i,j}))$.

La seconde partie du protocole est dédiée à l'élimination de ces valeurs aléatoires. A partir de M, les joueurs P_j vont calculer $PSS_j = \sum_{i=2}^{n} D_j(m_{i,j}) = \sum_{i=2}^{n} z_{i,j}$ en déchiffrant les valeurs contenues dans la j^{e} ligne de M. Ils envoient ensuite $\gamma_j = E_1(PSS_j)$ à P_1, ainsi que PSS_i chiffré avec la clef publique de P_1. Finalement, P_1 est capable de retrouver le bon résultat en calculant : $Confiance = D_1(a) - \sum_{j=2}^{n} D_1(\gamma_j) = D_1(a) - \sum_{j=2}^{n} PSS_j = D_1(a) - \sum_{j=2}^{n} \sum_{i=2}^{n} z_{i,j} = D_1(a) - \sum_{i=2}^{n} z_i = \sum_{i=2}^{n} u_i v_i$.

3.2 | P-MPWP : UNE VERSION ALLÉGÉE DE MPWP

Nous souhaitons maintenant modifier le protocole MPWP afin de réduire ses coûts de calcul et de communication. Par la suite, nous présentons tout d'abord le protocole *Paillier-MPWP* (P-MPWP), dérivé du protocole MPWP, suivi de son analyse de sécurité.

3.2.1 • PRÉSENTATION DU PROTOCOLE

P-MPWP est une variante de MPWP, dont les deux principales différences sont : l'utilisation du cryptosystème de Pailler plutôt que celui de Benaloh, et un coût global de communication réduit de $O(n^3)$ à $O(n^2)$ en ne partageant qu'une partie de la matrice M. Ainsi, l'algorithme de P-MPWP reprend celui MPWP, à l'exception des modifications que nous présentons maintenant.

Contrairement au cryptosystème de Benaloh, celui de Paillier ne permet pas aux joueurs de bénéficier d'un espace des clairs commun. Ainsi, il devient nécessaire de

définir la borne B suivante sur les coefficients des entrées secrètes des participants :

$$\forall i, 0 \leq u_i \leq B, 0 \leq v_i \leq B \tag{V.1}$$

En utilisant Benaloh, il suffit que l'espace des clairs $\mathbb{Z}/\sigma\mathbb{Z}$ soit plus grand que le produit scalaire calculé pour que ce dernier soit correct (*i.e.* dans $\mathbb{Z}$), c.-à-d. :

$$(n-1)B^2 < \sigma. \tag{V.2}$$

Avec Paillier, chaque joueur P_i possède donc un modulo différent dénoté N_i. Ainsi, en suivant les étapes du protocole MPWP, à la fin du premier tour, le joueur maître P_1 obtient $A = \prod_{i=2}^{n} E_1(u_i)^{v_i} * E_1(z_i)$. En supposant que la borne (V.1) est satisfaite, si P_1 souhaite déchiffrer correctement, son modulo doit être plus grand que $(n-1)(B^2+B)$. Les autres joueurs doivent aussi effectuer un déchiffrement au cours de l'exécution du protocole. Ils reçoivent $(n-1)$ parts, qui sont toutes bornées par B. Par conséquent, leur modulo N_i doit être plus grand que $(n-1)B$. Les conditions sur les modulos des joueurs sont synthétisées dans le résultat suivant :

Lemme 18. *Soit $n > 3$ le nombre de joueurs. Si les trois conditions suivantes sont satisfaites :*

- *Les entrées sont conformes à la borne (V.1) ;*
- *$\forall i \in [1,n], 0 \leq z_i \leq B$;*
- *les modulos sont tels que $(n-1)(B^2+B) < N_1$ et $(n-1)B < N_i$, $\forall i \in [2,n]$*

Alors, à la fin de l'exécution de P-MPWP, P_1 obtient $S_n = \sum_{i=2}^{n} u_i * v_i \in \mathbb{Z}$.

Afin de réduire les coûts de communication, l'idée est de supprimer l'échange de la matrice M entre tous les joueurs. Lors du calcul des parts $z_{i,j}$, chaque P_i envoie directement le j^{e} coefficient au joueur P_j, et ne le stocke pas dans M. A la fin, chaque P_i a ainsi reçu $(n-1)$ valeurs chiffrées avec sa clef publique, et est capable de calculer son PSS_i simplement en déchiffrant puis en ajoutant toutes les valeurs reçues, à l'instar de MPWP. Par conséquent, dans P-MPWP chaque joueur n'envoie plus que $O(n)$ éléments, contre $O(n^2)$ dans MPWP. Les dernières étapes du protocole sont communes aux deux protocoles.

3.2.2 • SÉCURITÉ DU PROTOCOLE

Intuitivement, comme les cryptosystèmes de Paillier et Benaloh [Pai99, FLA11, Ben94] sont prouvés sémantiquement sûrs, le passage de l'un à l'autre ne devrait pas altérer la sécurité globale du protocole. Il est cependant nécessaire de borner les entrées des joueurs (V.1) *a priori*, afin d'avoir une exécution du protocole se rapprochant de celle définie dans MPWP avec Benaloh. De cette manière, il est possible d'effectuer une preuve par réduction de la sécurité de MPWP avec Paillier sur celle de MPWP initialement définie avec Benaloh. Au niveau de la modification

des informations échangées, l'intuition est la suivante : l'ensemble des informations échangées entre les joueurs dans P-MPWP est un sous-ensemble de celui de MPWP. En conséquence, si les valeurs connues par les joueurs dans P-MPWP permettent de casser le protocole, cela signifie qu'ils sont aussi capables de casser MPWP. Nous obtenons donc le lemme suivant :

Lemme 19. *Le protocole P-MPWP est aussi sûr que MPWP.*

4 ■ DSDP : UN PROTOCOLE POUR LE PRODUIT SCALAIRE DISTRIBUÉ ET SÛR

Nous présentons dans la suite un protocole encore plus économe en mémoire (et en calculs) permettant de calculer un produit scalaire distribué de manière sécurisé. Nous commençons par décrire le protocole à différentes échelles (un exemple à trois joueurs, puis le cas général avec n joueurs). Dans la seconde partie, nous étudions la sécurité du protocole selon le modèle de l'adversaire, en démontrant la sécurité, ou en explicitant des attaques et les contre-mesures associées.

4.1 | LE PROTOCOLE DSDP

Cette partie a pour but de présenter le premier protocole permettant le calcul d'un produit scalaire distribué sûr, dénommé *Distributed Secure Dot Protocol* (DSDP).

4.1.1 • PREMIÈRE APPROCHE AVEC 3 JOUEURS

Dans un premier temps, intéressons nous à une instanciation à trois joueurs du protocole, représentée dans la Figure 1. Alice souhaite calculer le produit scalaire $S = U^T \cdot V$ de dimension 3 avec Bob et Charlie. Dans cette configuration, Alice est le joueur maître, c'est-à-dire qu'elle possède le vecteur complet U, tandis que les coefficients de V sont distribués entre les trois participants (Alice connaît v_1, Bob v_2 et Charlie v_3). Tout d'abord, Bob et Charlie doivent partager leurs entrées privées avec Alice. Afin de ne pas lui révéler directement, ils vont commencer par les chiffrer en utilisant leur propre clef publique. Ainsi, Bob (respectivement Charlie) obtient $c_2 = E_{pubB}(v_2)$ (resp. $c_3 = E_{pubC}(v_3)$). Les deux joueurs envoient ensuite leur valeur à Alice. Cette dernière calcule les produits de ses entrées u_2, u_3 avec c_2 et c_3, en utilisant la propriété homomorphe (II.2) pour obtenir $c_2^{u_2} = E_{pubB}(v_2 u_2)$ et $c_3^{u_3} = E_{pubC}(v_3 u_3)$. Alice tire ensuite deux nombres r_1 et r_2 au hasard. Ces deux nombres vont lui permettre de cacher ces entrées secrètes u_2 et u_3 aux autres joueurs. Elle applique ensuite l'autre propriété homomorphe du cryptosystème (II.1) pour

obtenir $\alpha_2 = E_{pubB}(v_2u_2 + r_2) = E_{pubB}(v_2u_2)E_{pubB}(r_2)$ et $\alpha_3 = E_{pubC}(v_3u_3 + r_3)$. Alice termine ces instructions en envoyant α_2 et α_3 à Bob. Après réception, Bob déchiffre α_2 et obtient $\Delta_2 = v_2u_2 + r_2$. Il utilise ensuite la clef publique de Charlie pour chiffrer Δ_2 et multiplier le résulat avec α_3 pour obtenir $\beta_3 = E_{pubC}(v_3u_3 + r_3 + v_2u_2 + r_2)$. Il envoie finalement β_3 à Charlie. Ce dernier va simplement le déchiffrer pour obtenir $\Delta_3 = v_3u_3 + r_3 + v_2u_2 + r_2$. Charlie le rechiffre en utilisant la clef publique d'Alice, et lui envoie le $\gamma = E_{pubA}(v_3u_3 + r_3 + v_2u_2 + r_2)$ obtenu. Après déchiffrement, elle enlève alors les valeurs aléatoires ajoutées au début du protocole, et complète avec le terme manquant u_1v_1 pour obtenir le produit scalaire $S = \sum_{i=1}^{3} u_iv_i$ initialement voulu.

La sécurité du protocole repose sur l'utilisation d'un cryptosystème homomorphe et de valeurs aléatoires. Ces deux outils permettent de cacher les valeurs des entrées aux autres joueurs. La sécurité du protocole est étudiée dans la la partie (4.2). Intuitivement, les arguments sont les suivants :

- Au début du protocole, Alice reçoit uniquement des valeurs chiffrées, l'empêchant de récupérer des informations ;
- Par la suite, les autres joueurs déchiffrent des valeurs intermédiaires (α_i, β_i). Ils obtiennent des équations contenant au moins trois termes (de la forme $u_iv_i + r_i$) et deux inconnues u_i and r_i. Ils ne sont donc pas capables de retrouver ces valeurs, ni celles des autres v_i potentiellement présentes ;
- Une fois que chacun des joueurs à ajouter ses informations à celles qu'il a reçues, le dernier joueur reboucle en envoyant le résultat chiffré à Alice. Après suppression des r_i, elle obtient $u_2v_2 + u_3v_3$. Il y a donc deux inconnues pour une seule équation, empêchant toute fuite d'information.

Après plusieurs chiffrements, déchiffrements et suppressions des valeurs aléatoires r_i, Alice obtient $S = \sum u_iv_i$. Les propriétés homomorphes (II.1)et (II.2) garantissent seulement que $D(\text{Add}(\text{Mul}(E(v_i);u_i);r_i)) = v_iu_i + r_i \mod N_i$, avec N_i le modulo utilisé par le joueur P_i. Cependant, à chaque étape ces valeurs doivent être rechiffrées, potentiellement avec un autre cryptosystème ou avec un modulo différent. A la fin, Alice doit être capable d'enlever correctement l'aléa et retrouver S dans $\mathbb{Z}$. D'une part, si les joueurs utilisent un cryptosystème permettant de partager le même modulo $M = N_i$, les opérations de déchiffrements et rechiffrements sont naturellement compatibles. Une telle mise en place est possible en utilisant le cryptosystème de Benaloh. D'autre part, en se basant sur un cryptosystème comme celui de Paillier, à la fin du protocole, Alice obtient : $S_4 = ((u_2v_2 + r_2) \mod N_2 + u_3v_3 + r_3) \mod N_3$. Elle peut enlever r_3, *via* $S_3 = S_4 - r_3 \mod N_3$, et elle obtient $S_3 = ((u_2v_2 + r_2) \mod N_2 + u_3v_3) \mod N_3$. En supposant que les coefficients des vecteurs U et V soient bornés par B, et que le troisième modulo est plus grand que le second, de sorte que $N_3 > N_2 + B^2$, elle obtient alors la valeur exacte de S_3 sur les entiers naturels : $S_3 = (u_2v_2 + r_2) \mod N_2 + u_3v_3$. Alice peut alors suppri-

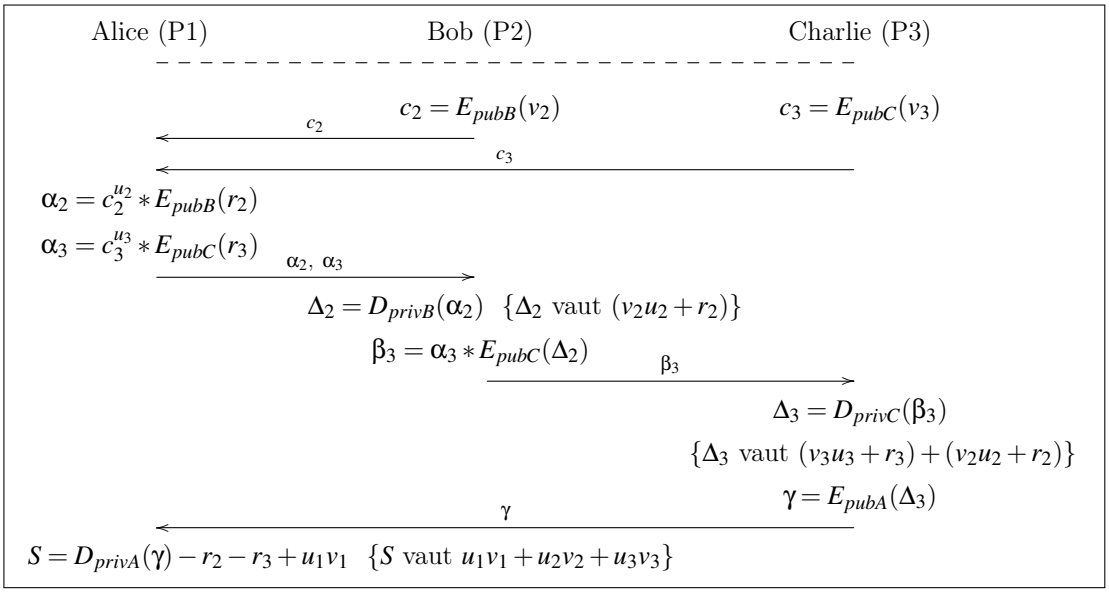

FIGURE 1 – *DSDP*$_3$: Calcul distribué et sûr d'un produit scalaire entre 3 participants reposant sur un cryptosystème à la Paillier

mer la seconde valeur aléatoire, modulo N_2 : $S_2 = (u_2v_2 + u_3v_3) \mod N_2$. Comme précédemment, il suffit de supposer que $N_2 > 2B^2$ pour qu'elle obtienne un résultat correct sur les entiers, soit $S = S_2 \in \mathbb{N}$. Montrons dans la partie suivante qu'il est possible de généraliser ces résultats à n joueurs.

4.1.2 • GÉNÉRALISATION À N JOUEURS

Dans l'algorithme (20), nous présentons la version plus générale avec n joueurs. Dans un premier temps, nous énonçons les hypothèses requises sur les entrées des joueurs (V.1) pour que le protocole s'exécute correctement.

Selon le cryptosystème utilisé, il existe deux bornes distinctes à respecter pour que le protocole s'exécute correctement avec n joueurs :

- Cryptosystème de Benaloh : Dans ce cas, tous les joueurs partagent un modulo commun pour l'espace des clairs σ pour tout i. Alice utilise alors le même modulo σ pour retirer les r_i. Ainsi, pour obtenir un résultat S correct, il suffit de vérifier la borne (V.2).
- Cryptosystème de Paillier : Dans ces conditions, les joueurs ne peuvent pas partager un modulo commun. Pour obtenir un résultat correct, ils doivent vérifier le lemme (21).

Lemme 21. *Soient $u_i, v_i, r_i \in \mathbb{Z}/B\mathbb{Z}$ (i.e. vérifiant la borne V.1), $M_2 = (u_2v_2 + r_2) \mod N_2$ et $M_i = (M_{i-1} + u_iv_i + r_i) \mod N_i$, avec $i = 2 \ldots n-1$. Si les modulos N_i sont*

Algorithme 20 $DSDP_n$ Protocole : Produit Scalaire Distribué et Sûr de taille n (Distributed and Secure Dot-Product)

Entrée(s): $n \geq 3$ joueurs, deux vecteurs U et V tels que P_1 connaisse le vecteur complet U, et que chaque joueur P_i connaisse la coordonnée v_i de V, avec $i = 1 \dots n$;

Entrée(s): E_i (resp. D_i), la fonction de chiffrement (resp. déchiffrement) de P_i, pour $i = 2 \dots n$.

Sortie(s): P_1 apprend le produit scalaire $S = U^T V$.

1: **Pour** $i = 2 \dots n$ **Faire**
2: $P_i : c_i = E_i(v_i)$
3: $P_i \xrightarrow{c_i} P_1$
4: **Pour** $i = 2 \dots n$ **Faire**
5: $P_1 : r_i \xleftarrow{\$} \mathbb{Z}/N_i\mathbb{Z}$
6: $P_1 : \alpha_i = c_i^{u_i} * E_i(r_i)$ tel que $\alpha_i = E_i(u_i v_i + r_i)$
7: $P_1 \xrightarrow{\alpha_2} P_2$
8: **Pour** $i = 2 \dots n-1$ **Faire** $P_1 : \xrightarrow{\alpha_{i+1}} P_i$
9: $P_2 : \Delta_2 = D_2(\alpha_2)$ tel que $\Delta_2 = u_2 v_2 + r_2$
10: $P_2 : \beta_3 = \alpha_3 * E_3(\Delta_2)$ tel que $\beta_3 = E_3(u_3 v_3 + r_3 + \Delta_2)$; $P_2 \xrightarrow{\beta_3} P_3$
11: **Pour** $i = 3 \dots n-1$ **Faire**
12: $P_i : \Delta_i = D_i(\beta_i)$ donc $\Delta_i = \sum_{k=2}^{i} u_k v_k + r_k$
13: $P_i : \beta_{i+1} = \alpha_{i+1} * E_{i+1}(\Delta_i)$ tel que $\beta_{i+1} = E_{i+1}(u_{i+1} v_{i+1} + r_{i+1} + \Delta_i)$; $P_i \xrightarrow{\beta_{i+1}} P_{i+1}$
14: $P_n : \Delta_n = D_n(\beta_n)$; $P_n : \gamma = E_1(\Delta_n)$; $P_n \xrightarrow{\gamma} P_1$
15: **Renvoyer** $P_1 : S = D_1(\gamma) - \sum_{i=1}^{n-1} r_i + u_1 v_1$.

tels que :

$$\begin{cases} N_{i-1} + (n-i+1)B^2 < N_i, & \text{pour tout } i = 3 \cdots n \\ (n-1)B^2 < N_2 \end{cases} \tag{V.3}$$

alors $S_2 = \sum_{i=2}^{n} u_i v_i \in \mathbb{N}$.

Démonstration. Par récurrence, commençons par montrer que $S_i = M_{i-1} + \sum_{j=i}^{n} u_j v_j$, pour $i = n..3$. On a $S_n = (M_n - r_n) \mod N_n = (M_{n-1} + u_n v_n) \mod N_n$. Or M_{n-1} est modulo N_{n-1}, donc $(M_{n-1} + u_n v_n) < N_{n-1} + B^2$, et (V.3) avec $i = n$ nous assure que $N_{n-1} + B^2 < N_n$ et $S_n = M_{n-1} + u_n v_n \in \mathbb{N}$. Alors, pour $3 \leq i < n$, $S_i = (S_{i+1} - r_i) \mod N_i = (M_i + \sum_{j=i+1}^{n} u_j v_j - r_i) \mod N_i = (M_{i-1} + u_i v_i + r_i + \sum_{j=i+1}^{n} u_j v_j - r_i) \mod N_i = (M_{i-1} + \sum_{j=i}^{n} u_j v_j) \mod N_i$, par récurrence. Or, la borne (V.1) nous donne $M_{i-1} + \sum_{j=i}^{n} u_j v_j < N_{i-1} + (n-i+1)B^2$ et (V.3) nous assure que ce dernier est plus petit que N_i. En conséquence, $S_i = M_{i-1} + \sum_{j=i}^{n} u_j v_j$ et la récurrence est prouvée. Finalement, $S_2 = (S_3 - r_2) \mod N_2 = (M_2 + \sum_{j=3}^{n} u_j v_j - r_2) \mod N_2 = (\sum_{j=2}^{n} u_j v_j)$

$\bmod N_2$. Comme $\sum_{j=2}^{n} u_j v_j < (n-1)B^2$, par (V.3) pour $i = 2$, nous avons $S_2 = \sum_{j=2}^{n} u_j v_j \in \mathbb{N}$. □

Ce résultat nous montre que le protocole $DSDP_n$ défini dans l'algorithme 20 peut être implémenté en utilisant le cryptosystème de Paillier. Pour cela, les joueurs successifs doivent avoir des modulos incrémentaux. Cependant, la taille de la borne B peut être petite en comparaison de celle des modulos : la taille des modulos n'augmente donc que légèrement.

Théorème 22. *Lorsque la borne (V.1) et les hypothèses d'un cryptosystème à la Benaloh (V.2) ou Paillier (V.3) sont respectées, le protocole $DSDP_n$ defini par l'algorithme (20) est correct. Son coût en communication est de $O(n)$, et il nécessite $O(n)$ opérations de chiffrement et déchiffrement.*

Démonstration. Tout d'abord, chaque joueur envoie son entrée chiffrée à P_1, qui ajoute homomorphiquement les valeurs aléatoires r_i. Ensuite, chaque P_i $(i \geq 2)$ déchiffre le message envoyé par P_{i-1}, et obtient Δ_i. Par récurrence, on a $\Delta_i = \sum_{k=2}^{i} u_k v_k + r_k$. Cette valeur est alors rechiffrée en utilisant la clef publique du joueur suivant, avant d'ajouter homomorphiquement la part du joueur suivant. A la fin, P_1 supprime simplement l'aléa et obtient $S = \Delta_n - \sum_{i=2}^{n} r_i + u_1 v_1 = \sum_{i=1}^{n} u_i v_i$. Au niveau de la complexité, le protocole requiert $n-1$ chiffrements et communications pour les c_i, $2(n-1)$ opérations homomorphes sur les chiffrés et $n-1$ communications pour les α_i ; $n-1$ déchiffrements pour les Δ_i ; $n-1$ chiffrements, opérations homomorphes et communications pour les β_i ; et finalement un chiffrement et une communication pour γ. En conclusion, P_1 a donc besoin de $O(n)$ opérations pour calculer S.

□

4.2 | SÉCURITÉ DU PROTOCOLE DSDP

La définition standard de la sécurité dans le modèle MPC [Lin09] permet de couvrir de multiples problématiques, telles que la correction d'un protocole, l'indépendance des données d'entrées ou encore le caractère privé (ou confidentiel) des données. Elle définit aussi plusieurs types d'adversaires, honnêtes-mais-curieux ou encore malicieux. En pratique, il est difficile d'obtenir un protocole prouvé sûr face à une équipe d'adversaires malicieux et ayant une complexité raisonnable. Pour cette raison, nous montrons que le protocole est sûr dans ce modèle contre un adversaire honnête-mais-curieux, puis nous affaiblissons le modèle face aux adversaires malicieux. En effet, parmi les propriétés précédemment énoncées, le protocole doit absolument garantir le secret des entrées privées des participants honnêtes. Ainsi, nous montrons ensuite qu'en utilisant des techniques classiques telles que les signatures ou les preuves à divulgation nulle de connaissance, ainsi

qu'une nouvelle méthode de placement aléatoire des joueurs, il est possible de garantir un niveau de sécurité dépendant d'un paramètre ε choisi a priori.

4.2.1 • ADVERSAIRE HONNÊTE MAIS CURIEUX

Dans un premier temps, intéressons nous à une propriété qui n'est pas prise en compte dans le modèle MPC : *la safety*. Cette dernière permet de s'assurer que le calcul effectué est "sain", c.-à-d. qu'à la fin de l'exécution du protocole, le résultat du calcul de la fonction f ne permet pas de retrouver les entrées des autres joueurs. Par exemple, le calcul d'une moyenne entre deux participants peut être prouvé sûr au sens du MPC, alors que le joueur obtenant le résultat pourra toujours retrouver la valeur utilisée par l'autre joueur [Lin09]. Dans notre cas, la fonction f est le calcul d'un produit scalaire, et nous obtenons donc le résultat suivant :

Lemme 23. *Soit $n \geq 3$ le nombre de participants, soient U et V deux vecteurs tels qu'un joueur connaisse U et que les coefficient de V soient distribués entre les joueurs tels que P_i connaisse v_i. Alors, le calcul du produit scalaire $U^T V$ est sain.*

Démonstration. A la fin de l'exécution d'un protocole de calcul de produit scalaire à la $DSDP_n$ avec $n \geq 3$, P_1 reçoit une unique équation contenant $(n-1)$ inconnues (les coefficients de v_2 à v_n). En conclusion, il n'est donc pas capable d'en déduire les entrées privées des autres joueurs. □

Regardons maintenant la sécurité du protocole face à des adversaires honnêtes-mais-curieux, sans coalition. Nous obtenons le résultat suivant :

Lemme 24. *Soit E un cryptosystème supposé sémantiquement sûr. Soit $n \geq 3$, alors $DSDP_n$ est sûr (au sens du MPC) contre un adversaire honnête-mais-curieux.*

Démonstration. Soit E un cryptosystème supposé sémantiquement sûr. Prouvons que le protocole $DSDP_n$ est sûr contre un adversaire honnête-mais-curieux si $n \geq 3$. La preuve est découpée en deux parties : tout d'abord nous nous intéressons au cas où le joueur maître P_1 est corrompu, puis au cas d'un autre joueur P_i. Dans chaque partie, nous construisons un algorithme simulant de la vue du joueur corrompu dans le cadre d'une exécution dans un monde idéal du protocole, et montrons que la vue générée par le simulateur est indistinguable (au sens *computationally indistinguishable*) de celle du joueur corrompu. Dans toute la preuve, une variable dénotée x dans le cas réel, est noté x' dans le cas idéal.

P_1 est corrompu. La vue de P_1 lors d'une exécution du protocole dans le cas réel est $View_{P_1} = \{U, R, \gamma, S, A, B, C\}$, où $U = \{u_i\}_{1 \leq i \leq n}$, $R = \{r_i\}_{1 \leq i \leq n}$, $A = \{\alpha_i\}_{2 \leq i \leq n}$, $B = \{\beta_i\}_{3 \leq i \leq n-1}$ et $C = \{c_i\}_{2 \leq i \leq n}$. Soit Sim_1, le simulateur du joueur P_1 dans le cas idéal. Par définition, les entrées secrètes du joueur P_1 (soient le vecteur U

et les valeurs aléatoires comprises dans R) sont directement accessibles par le simulateur. De même, ce dernier a aussi accès à la sortie S du protocole, qui est fournie par le tiers de confiance dans le monde idéal. Sim_1 débute par générer $n-2$ valeurs aléatoires qu'il chiffre en utilisant la clef publique correspondante (c.-à-d. que la i^{me} valeur aléatoire sera chiffrée avec la clef publique du joueur indexé $i+1$). Il obtient ainsi les valeurs c'_i, simulation des c_i dans le cas réel. Il effectue ensuite les calculs homomorphes tels qu'ils sont définis dans le protocole en combinant les c'_i avec les r_i et u_i pour obtenir $\alpha'_i = c_i'^{u_i} * E_i(r_i)$, avec $2 \leq i \leq n$. Les valeurs contenues dans B' sont obtenues en chiffrant de nouvelles valeurs aléatoires avec les clefs publiques appropriées. Enfin, γ' est calculé en utilisant les sortie S du protocole : $\gamma' = E_1(S + \sum_i^{n-2} r_i + u_1 v_1)$. La vue du simulateur est donc $View_{Sim_1} = \{U, R, \gamma', S, A', B', C'\}$. Montrons que les deux vues sont indistinguables. S'il existe un adversaire capable de distinguer C' de C (et donc A' de A), cela signifie qu'il est capable de casser la sécurité sémantique du cryptosystème. Comme le calcul de γ' utilise la même sortie qu'une exécution réelle, il n'est pas possible de les distinguer. De même, toutes les autres valeurs sont calculées en suivant les étapes du protocole, les rendant indistinguables du cas réel.

$P_i, i \geq 2$ est corrompu. On dénote par Sim_i le simulateur du joueur P_i corrompu. Compte tenu du rôle générique des joueurs dans le protocole (hormis pour P_n, ne différant des autres joueurs que par le calcul de γ à la place de β_{n+1}), l'algorithme à exécuter par le simulateur peut être aisément adapté. Lors d'une exécution réelle, la vue d'un joueur P_i est $View_{P_i} = \{v_i, A, B, C, \gamma, \Delta_i\}$. Les variables communes avec P_1 sont simulées de la même manière, en utilisant les clefs adéquates : A', B', γ', C' (exceptée c_i) sont donc obtenues en chiffrant des valeurs aléatoires. La simulation de c_i est réalisée en utilisant l'entrée secrète v_i et la clef publique du joueur P_i. Δ'_i est générée en utilisant les valeurs aléatoires précédemment tirées, puis chiffrées, pour simuler α'_i. L'indistinguabilité des deux vues est montrée en utilisant les deux arguments suivants : la sécurité sémantique de E (pour les valeurs A, B, C et γ), et l'utilisation d'aléas r_i par P_1, inconnus des autres joueurs. Ainsi, Δ'_i est indistinguable de Δ_i. Les vues obtenues lors d'une exécution réelle et idéale sont donc indistinguables. Le protocole $DSDP_n$ est donc prouvé sûr contre un adversaire honnête-mais-curieux. □

Nous avons prouvé la sécurité pour une seule exécution du protocole. En utilisant la transformation de Arapinis et al. dans [ADK14], il est possible de la généraliser à plusieurs exécutions. Cette transformation nécessite que tous les participants (honnêtes ou malicieux) partagent un nonce et leur identité au début du protocole. Par la suite, ces informations seront ajoutées à tous les messages nécessitant un appel aux primitives cryptographiques. Finalement, nous obtenons un protocole sûr pour un nombre quelconque d'exécutions.

4.2.2 • ADVERSAIRE MALICIEUX ISOLÉ (AVEC PROVERIF)

Intéressons-nous à la sécurité du protocole face à des adversaires dits malicieux. Dans un premier temps, nous supposons que les attaques sont menées par un seul adversaire. La modélisation du protocole avec des outils de preuves automatiques nous a permis de trouver deux attaques :

- La clef d'un des autres joueurs est compromise : une solution utilisant le mécanisme des signatures permet de contrer l'attaque.
- La clef du joueur maître est compromise (ou le joueur maître est malicieux) : cette attaque peut être déjouée à l'aide de *preuves de connaissances à divulgation nulle de connaissance* ;

Par la suite, nous détaillons tout d'abord la modélisation du protocole en ProVerif, puis les attaques découvertes associées à leur contre-mesure. Les sources sont disponibles sur :
https://casys.gricad-pages.univ-grenoble-alpes.fr/matmuldistrib/.

Modélisation en ProVerif. Nous utilisons le logiciel ProVerif pour modéliser notre protocole. Ce dernier nous permet de définir notre propre théorie équationnelle, et plus particulièrement les propriétés homomorphes de notre cryptosystème. Cependant, comme il a été prouvé dans [Del06], la vérification de protocoles reposant sur des théories impliquant des fonctions homomorphes sur des groupes abéliens est un problème indécidable. Nous avons donc adapté notre théorie équationnelle afin d'obtenir des résultats. En particulier, le morphisme additif est représenté par l'équation suivante :

(i). $\forall u,v,r,k,\ bob(E_k(r),u,E_k(v)) = E_k(uv+r)$

Cette propriété permet à Bob d'obtenir $u_2v_2+r_2$ à partir α_2. De plus, elle donne aussi la possibilité à l'adversaire d'effectuer cette déduction, et donc de modéliser le cas où Bob est corrompu. De plus, nous avons les équations suivantes :

(ii). $\forall u,v,r,x,y,z,k,\ charlie(E_k(uv+r),E_k(xy+z)) = E_k(uv+xy+r+z)$ (soit β_3)

(iii). $\forall u,v,r,x,y,z,a,b,c,k,\ dave(E_k(uv+xy+r+z),E_k(ab+c)) = E_k(uv+xy+ab+r+z+c)$ (soit β_4)

Nous utilisons ensuite ProVerif pour prouver la sécurité de notre protocole à l'aide de nos abstractions précédentes. L'outil trouve des attaques, pour lesquelles nous proposons des contre-mesures. Malheureusement, nous atteignons les limites de ProVerif lorsque nous les implémentons. Ce résultat n'est pas surprenant, puisque le problème lié à la décision quand à l'aspect secret ou non d'un terme dans un protocole cryptographique est indécidable [ALV03], et NP-Complet dans le cas des abstractions utilisées par ProVerif [RT03]. Le simple ajout de signatures dans le modèle est déjà trop exigeant pour l'outil. Une technique classique consiste à aider l'outil dans son analyse, en limitant les boucles possibles, où en supprimant

des chemins possibles dans l'arbre de vérification. Cependant, cette manipulation risque de supprimer la propriété de complétude de la preuve. Par la suite, les contre-mesures sont donc prouvées de manière usuelle.

Remarque 25. *Nous avons utilisé notre modèle dans le cas d'adversaire semi-honnête, où la corruption des participants est symbolisée par le partage de sa clef privée à l'adversaire contrôlant le réseau. Dans tous les cas de compromission (i.e. Alice, Bob ou Charlie), l'outil est en adéquation avec la preuve par simulation : aucune attaque n'est découverte.*

Signatures. Le logiciel ProVerif trouve une attaque lorsque Charlie est corrompu. Dans ce cas, l'adversaire bloque toutes communications émises ou reçues par Alice, supposée honnête. Il génère lui-même les valeurs composant les α_i à la place d'Alice. Ainsi, lorsque Charlie obtient $\Delta_3 = (v_2u_2 + r_2) + (v_3u_3 + r_3)$, et connaissant u_2, u_3, r_2, r_3 (puisque précédemment générés), il est capable de retrouver v_2.

D'un point de vue général, cette attaque est assimilable à un vol d'identité : Charlie est capable d'agir en tant qu'Alice sans que les autres joueurs ne puissent s'en rendre compte. Elle est donc facile à contrer, en utilisant des signatures cryptographiques. Ces dernières vont permettre de s'assurer que le seul le joueur connaissant la clef privée associée est capable d'envoyer des messages.

Si le joueur maître est compromis, ProVerif trouve une attaque à destination d'un autre participant P_i, choisi a priori par P_1, permettant de retrouver la valeur secrète v_i. Explicitons l'attaque dans le cas où P_2 est désigné comme cible. P_1 va remplacer le calcul des α_i, excepté pour α_2, par des chiffrés $E_i(x_i)$, où les x_i sont des valeurs qu'il aura choisies auparavant. Il peut par exemple prendre $x_i = 0$, ou encore $x_i = r_i$ (c.-à-d. qu'il pose $u_i = 0$). A la fin de l'exécution du protocole, P_1 obtient : $u_2v_2 + r_2 + \sum_{i=3}^{n} x_i$. Sachant qu'il connaît u_2, r_2 et tous les x_i, il peut calculer v_2.

Preuves à divulgation nulle de connaissance. Commençons par expliciter une attaque lorsque la clef du joueur maître est compromise. Pour simplifier, nous nous plaçons dans une exécution du protocole avec trois joueurs ($n = 3$), Alice, Bob et Charlie, voir la Figure (2). Nous supposons donc que Alice est malicieuse.

Lors d'une exécution normale du protocole, Bob chiffre la valeur secrète v_2 pour obtenir c_2, et l'envoie à Alice. Cette dernière étant corrompue, elle génère de fausses valeurs r_i et v_i pour tous les participants : cela lui permet de calculer des valeurs α_i alternatives pour tous les participants. La suite du protocole se passe de manière usuelle. A la fin, Alice reçoit de la part de Charlie la valeur γ. Comme toutes les valeurs contenues ont été remplacées par des valeurs choisies par Alice, il ne reste donc que la valeur v_2 qui est inconnue : Alice est donc capable de la retrouver.

Cette attaque, et plus généralement les attaques sur la modification des α_i peuvent être contrées en utilisant des preuves de connaissances à divulgation de

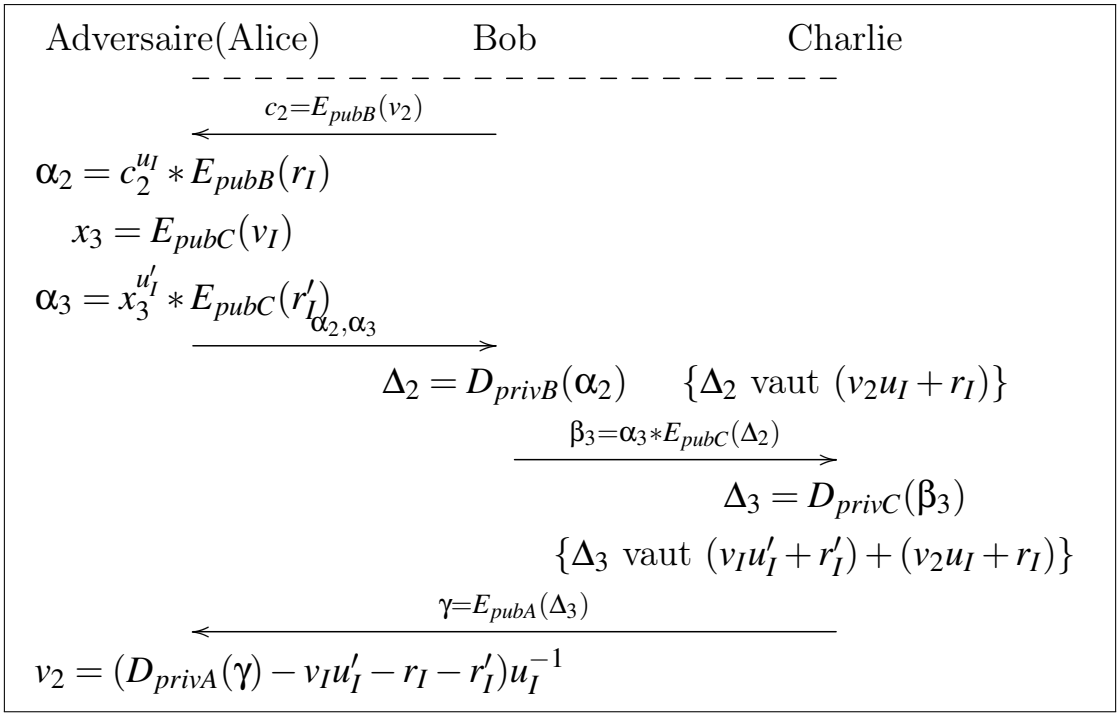

FIGURE 2 – Attaque d'Alice sur Bob révélant v_2

connaissances nulles (en anglais *zero-knowledge proofs of knowledge*, abrégé *ZK-PoK*). P_1 doit prouver aux autres joueurs que les α_i utilisés sont des transformations affines non triviales de leur secret v_i. Pour cela, il emploie une variante des *PoK* d'un logarithme discret, détaillée dans le schéma (3).

Dans le protocole (20), cette preuve de transformation affine non triviale s'applique directement sur α_2, avec $\mu_2 = g^{u_2}$ et $\rho_2 = g^{r_2}$, de manière à ce que P_2 vérifie $\delta_2 = g^{\Delta_2} \stackrel{?}{=} \mu_2^{v_2} \rho_2$. Durant la phase circulaire du protocole, la valeur $\delta_{i-1} = g^{\Delta_{i-1}}$ utilisée pour tester l'égalité doit être transférée au joueur suivant. Ainsi, lorsque le joueur effectue l'instruction (12), il doit aussi s'assurer que $\Delta_i = u_i v_i + r_i + \Delta_{i-1}$. Puisque P_1 envoie $\mu_i = g^{u_i}$, $\rho_i = g^{r_i}$ et que P_{i-1} fournit δ_{i-1}, il vérifie que $\delta_i = g^{\Delta_i} \stackrel{?}{=} \mu_i^{v_i} \rho_i \delta_{i-1}$.

La *PoK* des informations envoyées par transformée affine repose sur le problème du logarithme discret. Ce problème étant considéré comme difficile, elles ne révèlent aucune information sur les valeurs mises en exposant. En termes de coût en communications pour le protocole, on obtient selon les messages échangés entre :

- P_1 et P_i : α_i devient $(\alpha_i, \mu_i, \rho_i)$, soit à peu près le triple du volume ;
- P_i et P_{i+1} : β_i devient (β_i, δ_i), soit le double du volume.

D'un point de vue de la complexité, les opérations à effectuer sont principalement des exponentiations modulaires, et n'ont aussi qu'un coût global linéaire négligeable.

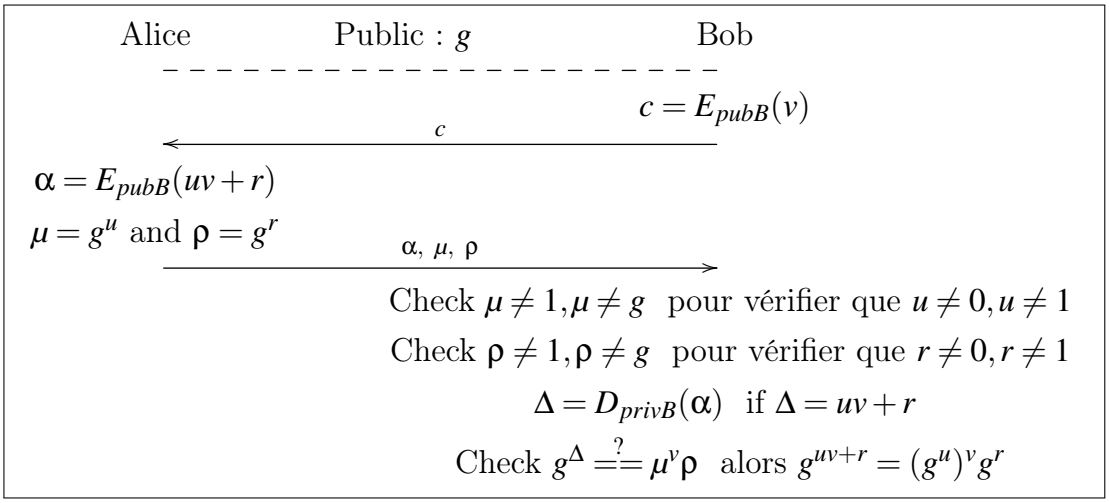

FIGURE 3 – ZK-PoK d'une transformation affine non triviale

4.2.3 • RANDOMRING : UNE CONTREMESURE CONTRE LES COALITIONS

Attaques en cas de coalition. Cette attaque peut être généralisée à un nombre quelconque de participants. Pour cela, il est nécessaire que le joueur maître coopère avec deux joueurs P_{i-1} et P_{i+1}. Ils partagent les valeurs suivantes :

- P_1 connaît u_i et r_i, avec $i \in [1, n]$;
- P_{i-1} connaît $\Delta_{i-1} = (u_2 v_2 + ... + u_{i-1} v_{i-1} + r_2 + ... + r_{i-1})$;
- P_{i+1} connaît $\Delta_{i+1} = (u_2 v_2 + ... + u_{i-1} v_{i-1} + u_i v_i + u_{i+1} v_{i+1} + r_2 + ... + r_{i-1} + r_i + r_{i+1})$.

En calculant la différence entre Δ_{i+1} et Δ_{i-1}, P_1 obtient : $\Delta_{i+1} - \Delta_{i+1} = u_i v_i + u_{i+1} v_{i+1} + r_i + r_{i+1}$. Puisque P_1 connaît tous les coefficients à l'exception de v_i, il est donc capable de le retrouver.

Présentation de l'anneau aléatoire. Nous avons vu précédemment qu'une coalition entre le joueur maître et le troisième joueur leur permet de retrouver les valeurs secrètes du second joueur. Plus généralement, lorsque trois joueurs P_1, P_{i-1} et P_{i+1} coopèrent, ils sont capables de retrouver la valeur secrète v_i du joueur P_i. Or, comme dans un produit matriciel, tous les joueurs deviennent à leur tour joueur maître, les chances d'une attaque envers un joueur honnête sont importantes. Afin de contrecarrer ce problème, l'idée est donc de répéter plusieurs fois le protocole du produit scalaire, en plaçant aléatoirement tous les joueurs à chaque tour, sauf le premier. Une proposition similaire a été proposée dans [CKV+02, § 2.1], avec l'hypothèse que la majorité des joueurs est honnête. Nous étendons donc ces résultats au cas où il est suffisant que seuls deux joueurs soient honnêtes. Dans

le même ordre d'idée, les auteurs de [SDKD10] proposent de répéter de manière déterministe le protocole $O(n)$ fois. Nous montrons ici que ce nombre peut être réduit dans le cas moyen.

Afin que l'attaque fonctionne, les adversaires malicieux (hormis P_1) doivent entourer leur cible. Donc, si le protocole doit être répété plusieurs fois avant d'obtenir le résultat, une attaque est réussie si et seulement si les adversaires ont entouré le joueur honnête à tous les tours. En conséquence, si l'emplacement des joueurs est choisi de manière aléatoire à chaque tour, cela signifie que la probabilité de succès d'une attaque est réduite à chaque nouvelle itération. D'un point de vue global, le *random ring order* apporte de la sécurité face à des adversaires malicieux car leur placement est choisi aléatoirement de manière uniforme : il est donc indépendant de la volonté des joueurs. La probabilité qu'un adversaire se retrouve bien placé est donc très réduite.

D'un point de vue pratique, le placement des joueurs doit donc être publiquement connu de tous. Pour cela, il est possible d'utiliser une fonction de hachage prenant la clef publique de tous les joueurs ainsi que l'heure de départ du protocole. Cette dernière peut facilement être partagée entre les joueurs, en utilisant un site web ou en la diffusant au départ. Cela permet d'obtenir un ordre aléatoire à chaque session du protocole. Notons que si cette méthode peut directement être employée avec le cryptosystème de Benaloh, nous devons adapter celui de Paillier pour qu'elle soit fonctionnelle. En particulier, le placement aléatoire ne nous permet plus d'utiliser le lemme (V.3).

La procédure pour un produit scalaire utilisant le protocole $PDSMM_n$ est détaillée ci-dessous. Tous les joueurs sauf le premier masquent leur coefficient secret v_i à la manière d'un canal sur écoute (*wiretap channel*) [OW84]. L'algorithme suivant (26) décrit la procédure :

Théorème 27. *L'algorithme (26) permet au premier joueur de calculer un produit scalaire de manière correcte.*

Démonstration. Commençons par le cas où le modulo est partagé entre les joueurs. Après le premier tour, Alice (P_1) obtient $S_1 = \sum_{j=2}^{n} u_j \left(v_j - \sum_{i=2}^{d} \lambda_{j,i}\right)$. Lors des tours suivants, elle reçoit $S_i = \sum_{j=2}^{n} u_j \lambda_{j,i}$. Finalement, elle calcule $\sum_{i=1}^{d} S_i = \sum_{j=2}^{n} u_j v_j$.

Avec un cryptosystème utilisant une configuration comme celui de Paillier, après la première occurrence, Alice obtient $S_1 = \sum_{j=2}^{n} u_j \left(v_j + \sum_{i=2}^{d} (B - \lambda_{j,i})\right)$. Lors des répétitions suivantes, elle réceptionne $S_i = \sum_{j=2}^{n} u_j \lambda_{j,i}$. Finalement, elle calcule $\sum_{i=1}^{d} S_i - (d-1)B(\sum_{j=2}^{n} u_j) = \sum_{j=2}^{n} u_j(v_j + (d-1)B) - (d-1)Bu_j = \sum_{j=2}^{n} u_j v_j$. □

Intéressons nous maintenant à la probabilité qu'une attaque soit évitée en utilisant le placement aléatoire. Afin de simplifier les calculs, nous supposons que le nombre de joueur est impair, soit $n = 2t + 1$.

Algorithme 26 Répétition du produit scalaire avec placement aléatoire

1: Les joueurs s'accordent sur le nombre d de répétitions
2: Chacun des joueurs calcule son emplacement pour chacune des d répétitions en utilisant une fonction de hachage cryptographique.
3a : Dans le cas d'un cryptosystème avec un modulo commun, les joueurs doivent partager M selon l'hypothèse (V.2) précédemment définie. Lors de la première occurrence du protocole, chaque joueur P_j masque alors son entrée secrète v_j avec $d-1$ valeurs aléatoires $\lambda_{j,i} \in \mathbb{Z}/M\mathbb{Z} : v_j - \sum_{i=2}^{d} \lambda_{j,i}.v_j - \sum_{i=2}^{d} \lambda_{j,i}$.
3b : Avec un cryptosystème à la Paillier, les joueurs choisissent leur modulo selon l'hypothèse (V.3), où B^2 est remplacé par dB^2. Dans les groupes de taille $n=4$, les conditions nécessaires de (V.3) sur les modulos sont en quelque sorte séquentielles, mais peuvent être satisfaites indépendamment si chaque modulo est choisi dans un intervalle distinct plus grand que $3dB^2$. Durant le premier tour, chaque P_j masque son entrée secrète v_j avec $d-1$ valeurs aléatoires $0 \leq \lambda_{j,i} < B : v_j + \sum_{i=2}^{d}(B - \lambda_{j,i}) < dB$.
4 : Pour chacune des occurrences suivantes, chaque joueur remplace son coefficient par un des $\lambda_{j,i}$.
5 : A la fin, le premier joueur a collecté d produits scalaires. Il lui suffit de tous les sommer pour obtenir le bon résultat.

Théorème 28. *Supposons un nombre de joueurs n impair ($n = 2t+1$), groupé par trois, parmi lesquels $k \leq n-2$ sont malicieux (incluant nécessairement le joueur maître Alice) et coopérants. Alors, il est en moyenne suffisant d'exécuter $d \leq 2\ln\left(\min\{k-1, n-k, \frac{n-1}{2}\}\right)\left(1+\frac{k-1}{n-k-1}\right)$ fois l'algorithme (26) pour éviter que les joueurs malicieux puissent retrouver la valeur secrète de ceux qui sont honnêtes.*

Démonstration. Pour que la valeur secrète soit révélée à Alice, il faut que Bob soit entouré de joueurs malicieux coopérants à chaque exécution du protocole. De plus, dans le cas où il n'y qu'un seul joueur de non corrompu, il n'est pas possible de le protéger. Dans le cas où deux joueurs sont honnêtes, il suffit qu'ils soient ensemble durant un tour : cette situation se produit avec une probabilité $\frac{1}{n-2}$, soit après $n-2$ répétitions en moyenne. Le protocole $PDSMM_n$ utilise $t = \frac{n-1}{2}$ groupes de trois joueurs, incluant Alice. En conséquence, à chaque fois qu'un groupe est formé avec un joueur malicieux et un autre honnête, Alice peut apprendre la valeur privée du joueur non malicieux. Après chaque tour, le nombre a de joueurs attaqués est plus petit que le nombre de joueurs malicieux moins un (pour Alice), et est évidemment inférieur au nombre de joueurs honnêtes : $0 \leq a < \min\{k-1, n-k\}$. Posons $b = k-1-a$ et $c = n-k-a$. Durant la prochaine répétition, la probabilité de conserver au moins un joueur non malicieux est $\frac{a(a-1+c)(n-3)!}{(n-1)!}\frac{n-1}{2} = \frac{a(a-1+c)}{2(n-2)} = \frac{a(n-k-1)}{2(n-2)}$. Cela implique donc que le nombre d'occurrences moyen pour conserver tous les joueurs honnêtes vaut

$\mathbb{E}_{n,k}(a) = \frac{2(n-2)}{a(n-k-1)}$. En conséquence $T_{n,k}(a)$, le nombre moyen d'occurrences pour conserver tous les joueurs non malicieux, satisfait $T_{n,k}(a) \leq \mathbb{E}_{n,k}(a) + T_{n,k}(a-1) \leq \sum_{i=a}^{3} E_{n,k}(i) + T_{n,k}(2) = (\sum_{i=a}^{3} \frac{1}{i}) \frac{2(n-2)}{n-k-1} + T_{n,k}(2)$. Avec deux joueurs attaqués et c sauvés, considérons $T_{n,k=n-c-2}(2) = \frac{n-2}{c+1}$ tel que $T_{n,k}(a) \leq (H_a - \frac{3}{2}) \frac{2(n-2)}{n-k-1} + \frac{n-2}{n-k-1}$, avec $H_a = \sum_{i=a}^{3} \frac{1}{i}$. En utilisant la borne sur les nombres harmoniques [Bat11]), et sachant que $a \leq k-1$ et $a \leq n-k$, nous obtenons alors $2a \leq n-1$. En conclusion, on obtient donc $T_{n,k}(a) \leq 2\ln\left(\min\{k-1, n-k, \frac{n-1}{2}\}\right) \frac{n-2}{n-k-1}$. □

Par exemple, si k est le nombre d'adversaires est plus petit que le nombre de joueurs honnêtes, alors, il suffit en moyenne de répéter le protocole $O(\log k)$ fois pour éviter l'attaque par collusion. Il est aussi possible de revenir à une définition plus classique de sécurité, en fixant une probabilité d'attaque inférieure à un ε fixé. Nous nous plaçons maintenant dans le pire cas possible, à savoir $k = n-2$. Dans ce cas, le nombre de répétitions doit être augmenté jusqu'à $n\ln(1/\varepsilon)$. Ce résultat est résumé dans le lemme suivant :

Proposition 29. *Soient $n = 2t+1$ le nombre de participants au protocole et d le nombre de répétitions de l'algorithme de placement aléatoire (26). Dans le pire des cas, c.-à-d. que le nombre d'adversaires est égal à $k = 2t-1$, la probabilité qu'une attaque soit réussie est inférieure à ε en effectuant $d < n\ln(1/\varepsilon)$ répétitions du protocole.*

Démonstration. Supposons qu'au moins deux joueurs soient honnêtes (autrement le calcul du produit scalaire ne peut pas être effectué de manière sécurisée). Un joueur non compromis est sauvé s'il existe une répétition du protocole au cours de laquelle il est entouré d'un autre joueur intègre. Dans le cas où seuls deux joueurs sont honnêtes, cela se produit avec une probabilité $(1 - \frac{1}{n-1})^d$, où d est le nombre de répétitions. Si $d \geq n\left(\ln\left(\varepsilon^{-1}\right)\right)$, alors $d > (n-1)(-\ln\varepsilon) > \frac{\ln\varepsilon}{\ln\left(1-\frac{1}{n-1}\right)}$, ce qui montre que $(1 - \frac{1}{n-1})^d < \varepsilon$. □

Une méthode alternative au placement aléatoire, reposant sur l'utilisation de $d = (n-1)/2$ chemins hamiltoniens disjoints [SDKD10] peut aussi être utilisée dans le pire des cas.

4.3 | APPLICATION DE DSDP AU PRODUIT MATRICIEL

Le protocole $DSDP_n$ peut être appliqué au produit de deux matrices A et B de taille $n \times n$. Dans ce cas, la répartition des données est un peu différente : chaque joueur P_i connaît désormais deux vecteurs A_i et B_i, c.-à-d. une ligne dans

chacune des matrices. A la fin, chacun des joueurs apprend une ligne C_i de la matrice résultante $C = AB$. Afin d'accélérer le calcul matriciel, il est naturel de paralléliser le protocole $DSDP_n$. Ainsi, le produit scalaire va être découpé en blocs de produits scalaires de taille 2 ou 3 coefficients. En effet, le calcul d'un produit scalaire entre trois (resp. quatre) joueurs nécessite deux (resp. trois) nouveaux coefficients en plus de ceux déjà connus par le joueur maître. Pour P_1, l'idée est d'effectuer $DSDP_3$ sur ces coefficients u_1, v_1, u_2, u_3 avec les v_2, v_3 de P_2 et P_3. A la fin de l'exécution du protocole, P_1 connaît donc $s = u_1v_1 + u_2v_2 + u_3v_3$. Ensuite, P_1 peut appeler une nouvelle fois le protocole en utilisant $s, 1, u_4, u_5$, avec les joueurs P_4 et P_5 qui fournissent u_4 et u_5. Ce processus peut-être itéré avec les joueurs suivants jusqu'à l'obtention du produit scalaire de la bonne dimension. Afin d'accélérer le calcul, P_1 a la possibilité de lancer ce calcul en parallèle, en ajoutant sa part u_1v_1 une fois que tous les calculs ont été effectués. Pour cela, il suffit de modifier la dernière ligne (15) de l'algorithme (20) décrivant $DSDP_n$ en $P_1 : S = D_1(\gamma) - \sum_{i=1}^{n-1} r_i$. La modification est détaillée dans l'algorithme (30).

Algorithme 30 Protocole $ESDP_n$: Produit Scalaire Sûr Externe de taille n (External Secure Dot-Product)

Entrée(s): $n+1$ joueurs, P_1 connaît un vecteur $U \in \mathcal{F}^n$, chaque P_i connaît la coordonnée v_{i-1} de $V \in \mathcal{F}^n$, pour $i = 2 \dots n+1$.
Sortie(s): P_1 connaît $S = U^T V$.
Renvoyer $DSDP_{n+1}(P_1 \dots P_{n+1}, [0,U], [0,V])$.

Le découpage en blocs de 3 et 4 ($ESDP_2$ ou $ESDP_3$) est suffisant pour calculer tous les produits scalaires de dimension n (car le $pgcd(3,2)$ est égal à 1). Selon la parité de n, $ESDP_2$ est appelé $\frac{n-1}{2}$ ou $\frac{n}{2} - 2$ fois, et $ESDP_3$ est appelé 0 ou 1 fois. L'algorithme (31) détaille la procédure complète :

Théorème 32. *Le protocole $PDSMM_n$ décrit dans l'algorithme (31) est correct et nécessite moins de 5 échanges en parallèle.*

Démonstration. Commençons par montrer que le protocole est correct, c.-à-d. qu'à la fin de son exécution, chaque P_i a appris la ligne C_i, avec $C = AB$. Comme le protocole est appliqué sur toutes les lignes et les colonnes, montrons que le coefficient c_{ij} est égal à $\sum_{l=1}^{n} a_{il} * b_{lj}$. Remarquons tout d'abord que les coefficients k_i sont simplement les valeurs $1 \dots (i-1)$ et $(i+1) \dots n$ dans l'ordre. Ainsi, le résultat obtenu par une exécution quelconque d'un appel à $ESDP_2$ est $a_{i,k_1}b_{k_1,j} + a_{i,k_2}b_{k_2,j}$, et que le résultat potentiel de $ESDP_3$ est $a_{i,k_3}b_{k_3,j} + a_{i,k_2}b_{k_2,j} + a_{i,k_1}b_{k_1,j}$. En les additionnant, puis en ajoutant les termes $a_{i,i} * b_{i,j}$, P_i obtient $c_{ij} = \sum_{l=1}^{n} a_{il} * b_{lj}$.

Pour le coût en communication, pour tout i et j, tous les appels à $ESDP$ sont indépendants. Ainsi, chacun des joueurs peut recevoir et émettre simultanément

Algorithme 31 Protocole $PDSMM_n$: Calcul parallèle, distribué et sûr d'un produit matriciel (Parallel Distributed and Secure Matrix Multiplication)

Entrée(s): n joueurs, chaque joueur P_i connaît les lignes A_i et B_i de deux matrices A et B de taille $n \times n$.

Sortie(s): Chaque joueur P_i connaît la ligne résultante C_i de $C = AB$.

Pour Chaque ligne : i=1 ... n **Faire**
  Pour Chaque colonne : j=1 ... n **Faire**
    $s \leftarrow a_{i,i} b_{i,j}$
    Si n est pair **Alors**
      $k_1 \leftarrow (i-1) \mod n+1$
      $k_2 \leftarrow (i-2) \mod n+1$
      $k_3 \leftarrow (i-3) \mod n+1$
      $s \leftarrow s + ESDP_3(P_i, [P_{k_3}, P_{k_2}, P_{k_1}], [a_{i,k_3}, a_{i,k_2}, a_{i,k_1}], [b_{k_3,j}, b_{k_2,j}, b_{k_1,j}])$
      $t \leftarrow \frac{n-4}{2}$
    Sinon
      $t \leftarrow \frac{n-1}{2}$
    Pour $h = 1 \ldots t$ **Faire**
      $k_1 \leftarrow (i+2h-1) \mod n+1$; $k_2 \leftarrow (i+2h) \mod n+1$
      $s \leftarrow s + ESDP_2(P_i, [P_{k_1}, P_{k_2}], [a_{i,k_1}, a_{i,k_2}], [b_{k_1,j}, b_{k_2,j}])$
    $c_{i,j} \leftarrow s$

plusieurs données. En parallèle $ESDP_2$, comme $DSDP_3$ dans la figure (1), nécessite 4 échanges avec un nombre constant d'opérations : un pour c_i, un pour α_i, un pour β_3 et un pour γ. Comme le montre l'algorithme (30), $ESDP_3$, comme $DSDP_4$, ne requiert qu'un échange supplémentaire pour β_4. □

L'exécution du protocole *DSDP* peut donc être faite en parallèle avec des blocs de 3 ou 4 coefficients. Dans cette nouvelle version, le nombre global de communications reste le même, mais le nombre de tours est réduit de n à 5. Du point de vue de la sécurité, le protocole n'est plus sûr au sens du MPC : puisque les calculs sont découpés en bloc, les joueurs apprennent à chaque étape une partie du résultat. Le protocole étant défini comme le produit scalaire de taille n, la connaissance du seul résultat de ce produit scalaire n'est plus applicable. Pourtant, comme le protocole est une composition de sous protocoles sûrs (calculs de produits scalaires de dimension 3 ou 4), nous pouvons montrer que le protocole reste sûr dans un contexte plus restreint, c.-à-d. que la probabilité que des informations privées soient divulguées est négligeable (voir la section 4.2).

5 ■ *YTP-SS* : UN PROTOCOLE DUAL À *DSDP*

Dans l'article [YTP07], Yao, Tamassia et Proctor présentent un protocole permettant de calculer un produit scalaire distribué sûr, dénommé *YTP*. Dans cette partie, nous rappelons tout d'abord le fonctionnement du protocole, avant de montrer qu'il est possible d'améliorer son efficacité sans altérer sa sécurité.

5.1 | YTP

Dans la première partie du protocole, le joueur maître P_1 (c.-à-d. celui qui possède le vecteur complet U) commence par chiffrer avec sa propre clef publique chacune des valeurs u_i, $i \in [2,n]$. Il distribue ensuite chacune des valeurs $E_1(u_i)$ obtenues au joueur P_i. En utilisant un cryptosystème vérifiant les propriétés homomorphiques précédemment définies (II.1 et II.2), les joueurs P_i calculent $E_1(u_i)^{v_i}E_1(-r_i)$, où r_i est une valeur aléatoire et v_i leur entrée secrète. Chacun des joueurs envoie ensuite le résultat à P_1. Ce dernier calcule le produit de toutes les valeurs reçues, soit $D_1(\prod_{i=2}^{n} E_1(u_i v_i - r_i)) = \sum_{i=2}^{n} u_i v_i - r_i$.

Durant la seconde phase du protocole, les joueurs P_i vont calculer la somme des r_i, en utilisant un protocole de sommation sûr. Le protocole utilisé par les auteurs repose sur la technique de partage du secret en $O(n^2)$. Ils évoquent aussi l'utilisation du protocole défini dans la partie [AEDS03, § 5.1]. L'idée est de calculer la somme des aléas r_i en cachant leur valeur avec une seconde valeur aléatoire s_i. Les joueurs ayant un index pair vont envoyer la somme $r_i + s_i$ aux joueurs ayant un index impair. Ensuite, les joueurs pairs (resp. impairs) calculent $\sum s_{2i+1}$ (resp. $\sum r_{2i} + s_{2i}$). Enfin, à l'aide d'échanges biparties, les joueurs retrouvent la somme totale. Finalement, en utilisant les deux sommes, P_1 est capable de calculer $\sum_{i=2}^{n} r_i$, et donc de retrouver le produit scalaire.

En analysant cette approche, nous montrons par la suite que ce protocole de sommation est aussi sûr que le protocole classique appelé *salary sum* [Sch95]. Le protocole est le suivant : un joueur (par exemple P_2) choisit un nombre aléatoire s. Il l'ajoute simplement à son entrée secrète, r_i, et utilise la clef publique du joueur suivant (selon leur index) pour chiffrer la somme (c.-à-d. $E_3(r_2 + s)$) avant de lui envoyer. Après réception, le joueur déchiffre et complète par son entrée secrète à la somme précédente. En utilisant une topologie circulaire, les joueurs ajoutent au fur et à mesure des termes à la somme. Le dernier joueur envoie finalement au premier, qui enlève simplement l'aléa s pour obtenir la somme.

5.2 | SÉCURITÉ DES PROTOCOLES DE SOMMATION

Comme avec ***DSDP***, nous allons utiliser une vérification automatique des protocoles de sommation sécurisés [AEDS03] et [Sch95] afin de montrer que le premier est aussi sûr que le second. Les sources sont disponibles sur :

`https://casys.gricad-pages.univ-grenoble-alpes.fr/matmuldistrib/`.

Pour cela, nous nous servons de AVISPA [ABB+05], qui permet de modéliser le protocole via un seul langage pour une vérification par quatre outils différents. Notre idée est d'utiliser une abstraction pour sommes,via l'opérateur **OU-Exclusif** (*xor*). La somme classique et le *xor* ont globalement les mêmes propriétés (commutativité, associativité). Seule l'obtention d'un inverse pour l'addition nécessite un calcul supplémentaire, alors que dans le cas du *xor*, l'inverse d'un élément x est x. Concernant la vérification, cette abstraction n'influe pas sur la complétude, mais peut éventuellement rajouter de fausses attaques. L'utilisation d'AVISPA, et plus particulièrement de Cl-Atse [Tur06] et OFMC [BMV03] permet de gérer plus efficacement les modélisations avec des *xor* qu'avec ProVerif.

Dans le protocole du *salary sum* [Sch95], nous vérifions que les éléments r_i restent secrets. Les logiciels Cl-Atse et OFMC ne trouvent pas d'attaque lorsqu'un seul joueur est compromis. Dans le cas où deux joueurs sont corrompus, les deux outils trouvent une attaque similaire à ***DSDP***, lorsque le joueur honnête est entouré par les attaquants. Le joueur précédent (soit P_{i-1}) la cible P_i partage ses connaissances avec le joueur suivant (P_{i+1}), et le calcul de la différence entre les deux permet de retrouver la valeur secrète r_i, puisque P_{i+1} connaît $r_i i+1$: $(s+\sum_{j=2}^{i+1} r_j) - (s+\sum_{j=2}^{i-1} r_j) = r_i + r_{i+1}$.

Nous étudions ensuite l'impact sur le secret des termes s_i du protocole [AEDS03] en fonction de la corruption des participants. Comme précédemment, aucune attaque cohérente (qui ne soit pas due à un biais de la modélisation) n'est trouvée lorsqu'au plus un joueur est corrompu. En supposant que deux joueurs sont malicieux, les outils trouvent une attaque permettant de retrouver un s_i quelconque, sans que les adversaires ne soient contraints d'entourer le joueur honnête. La figure suivante (4) expose l'attaque sur la cible P_3, avec une coopération de P_2 et P_5. En suivant le protocole, P_2 envoie r_2+s_2 à P_3. Ce dernier envoie r_3 à P_2 et $r_2+s_2+r_3+s_3$ à P_5. Ainsi, P_2 et P_5 connaissent à eux deux : $r_2+s_2+r_3+s_3$, r_2, s_2, r_3, leur permettant de retrouver s_3.

Nous avons donc montré que les deux protocoles permettant de calculer des sommes de manière distribuée et sécurisée offrent des propriétés de sécurité similaires. Montrons maintenant qu'ils peuvent être échangés dans ***YTP***, soit que le protocole complet ***YTP-SS*** est sûr.

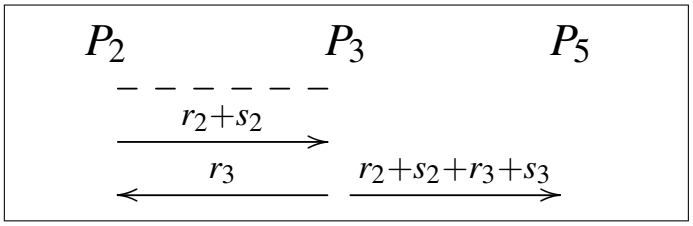

FIGURE 4 – Attaque en équipe de P_2 et P_5 sur P_3 dans le protocole de sommation [AEDS03]

5.3 | TRANSFORMATION DE *YTP* EN *YTP-SS*

Dans un premier temps, nous commençons par modifier le protocole ***YTP***, afin d'améliorer ses performances. Comme nous l'avons vu précédemment, l'utilisation du protocole décrit dans [AEDS03] n'apporte pas plus de garantie au niveau de la sécurité qu'un protocole plus simple, de type *salary sum*. Nous proposons donc de remplacer le protocole actuellement utilisé dans ***YTP*** par ce dernier, afin d'obtenir un nombre de communications linéaire par rapport au nombre de joueur. Ainsi, le nombre d'opérations cryptographiques effectué deviendra lui aussi linéaire pendant la phase de sommation. Nous dénommons cette variante du protocole ***YTP****with Salary Sum* (YTPSS)

Une exécution du protocole avec trois participants est illustrée dans la figure (5). L'algorithme suivant (33) décrit le protocole dans le cas général avec n joueurs.

5.4 | SÉCURITÉ DE *YTP-SS*

A l'instar de ***DSDP***, la sécurité du protocole est étudiée face à deux types d'adversaires : honnêtes-mais-curieux et malicieux. Dans le premier cas, nous utilisons le paradigme de preuve défini dans le MPC (comparaison des vues des exécutions réelle et idéale), tandis que dans le second nous nous plaçons dans un modèle affaibli assurant le secret des entrées privées. Dans un premier temps, nous nous intéressons à la sécurité des sous-protocoles permettant d'effectuer une somme sécurisée, en montrant que le protocole utilisé dans le protocole ***YTP*** est aussi sûr que le *salary sum* qui le remplace dans ***YTP-SS***.

5.4.1 • ADVERSAIRE HONNÊTE-MAIS-CURIEUX

Tout d'abord, nous exposons le protocole ***YTP-SS*** face à un adversaire honnête-mais-curieux. Nous obtenons le lemme suivant :

Lemme 34. *Soit $n \geq 3$. En supposant que le cryptosystème E est sémantiquement sûr, le protocole **YTP-SS** est sûr face à un adversaire semi-honnête.*

Algorithme 33 *YTP-SS* protocole : *YTP* avec *Salary Sum*

Entrée(s): $n \geq 3$ joueurs, deux vecteurs U et V tels que P_1 connaisse le vecteur U, et que chaque joueur P_i connaisse le coefficient v_i de V, pour $i = 1 \ldots n$;

Entrée(s): E_i (resp. D_i), fonction de chiffrement (resp. déchiffrement) de P_i, pour $i = 1 \ldots n$.

Sortie(s): P_1 apprend le produit scalaire $S = U^T V$.

1: **Pour** $i = 2 \ldots n$ **Faire**
2: $P_1 : c_i = E_1(u_i)$
3: $P_1 \xrightarrow{c_i} P_i$
4: $P_i : r_i \xleftarrow{\$} \mathbb{Z}/N_1\mathbb{Z}$
5: $P_i : \alpha_i = c_i^{v_i} * E_1(-r_i)$ soit $\alpha_i = E_1(u_i v_i - r_i)$
6: $P_i : \xrightarrow{\alpha_i} P_1$
7: $P_1 : S_a = \sum_{i=2}^{n} D_1(\alpha_i)$ avec $S_a = \sum_{i=2}^{n} u_i v_i - r_i$
8: $P_2 : s \xleftarrow{\$} \mathbb{Z}/N_1\mathbb{Z}$
9: $P_2 : \beta_3 = E_3(r_2 + s)$
10: $P_2 : \xrightarrow{\beta_3} P_3$
11: **Pour** $i = 3 \ldots n-1$ **Faire**
12: $P_i : \Delta_i = D_i(\beta_i)$
13: $P_i : \beta_{i+1} = E_{i+1}(\Delta_i + r_i)$
14: $P_i : \xrightarrow{\beta_{i+1}} P_{i+1}$
15: $P_n : \Delta_n = D_n(\beta_n)$
16: $P_n : \beta_2 = E_2(\Delta_n + r_n)$
17: $P_n : \xrightarrow{\beta_2} P_2$
18: $P_2 : S_b = D_2(\beta_2) - s$ avec $S_b = \sum_{i=2}^{n} r_i$
19: $P_2 : \xrightarrow{E_1(S_b)} P_1$
20: **Renvoyer** $P_1 : S = S_A + D_1(E_1(S_b)) + u_1 v_1 = \sum_{i=1}^{n} u_i v_i$.

Démonstration. Comme précédemment, nous allons décrire les algorithmes des simulateurs agissant dans le monde idéal, afin de montrer qu'il est possible de générer des vues indistinguables (*computationally indistinguishable*) de celles obtenues par un adversaire lors d'une exécution réelle.

Soit E le cryptosystème sous-jacent vérifiant une sécurité sémantique. L'exécution du protocole fait appel à trois rôles distincts :

- Le joueur maître, possédant le vecteur U complet ;
- Un joueur adjoint, en charge de la génération de l'aléa s utilisé pendant la phase de sommation ;
- Les autres joueurs, dont le rôle est générique (c.-à-d. qu'ils ne diffèrent que

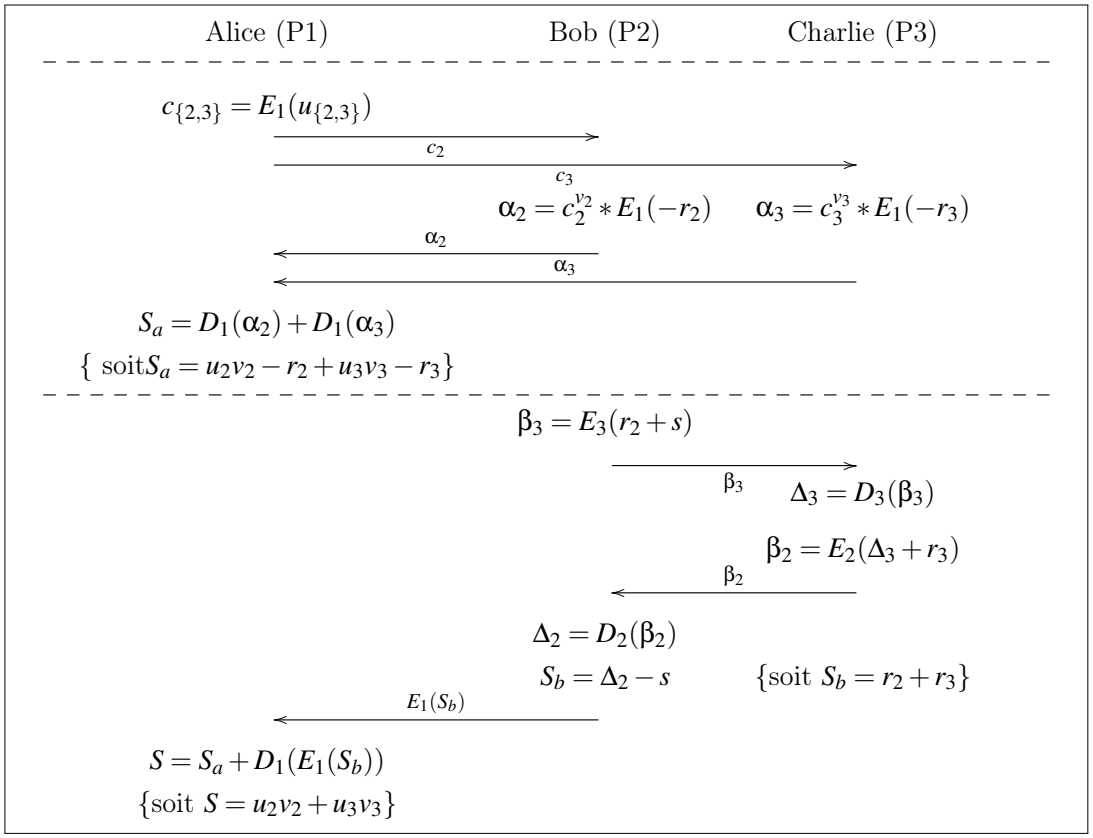

FIGURE 5 – Le protocole *YTP-SS* permettant de calculer un produit scalaire de manière sécurisée, en utilisant un cryptosystème homomorphe E

par leur indice)

Nous allons donc découper la preuve en trois parties, chacune dédiée à la description de l'algorithme des simulateurs Sim_A, où A dénote le joueur corrompu. Chaque simulateur a accès aux entrées secrètes et à l'aléa du joueur A, ainsi qu'à la sortie de l'algorithme, fournis par l'entité modélisant le tiers de confiance dans le monde idéal. On rappelle que les entrées secrètes sont le vecteur U pour P_1, le coefficient v_i pour P_i ($i \in \{2..n\}$), et la sortie est $S = \sum_{i=2}^{n} u_i v_i$ pour le joueur P_1. Les notations de la preuve reprennent celles qui sont définies dans le protocole (33). De plus, une variable notée X lors d'une exécution réelle du protocole sera écrite X' dans le cas d'une simulation dans le monde idéal. Enfin, nous définissons les variables suivantes : $\gamma = E_1(S_b)$, $U = \{u_i\}_{2 \leq i \leq n}$, $A = \{\alpha_i\}_{2 \leq i \leq n}$, $B = \{\beta_i\}_{2 \leq i \leq n}$, $C = \{c_i\}_{2 \leq i \leq n}$.

Premier cas : P_1 est corrompu. Lors d'une exécution réelle, le joueur P_1 obtient la vue suivante : $View_{P_1} = \{U, C, A, B, S_a, S_b, S\}$. Décrivons l'algorithme du simulateur Sim_{P_1} permettant d'obtenir la vue $View_{Sim_{P_1}} = \{U, C', A', B', S'_a, S'_b, S\}$. Tout d'abord,

le simulateur calcule $\forall i \in \{2..n\} : c'_i = E_1(u_i)$. Il tire ensuite $n-1$ nombres aléatoires w_i, et calcule $\alpha'_i = E_1(w_i)$. Il en déduit $S'_a = \sum_{i=2}^{n}(w_i)$. Les valeurs B' sont obtenues en chiffrant de nouveaux nombres aléatoires z_i en utilisant les clefs publiques des joueurs P_i, soit $\beta'_i = E_i(z_i)$. En utilisant la sortie S, Sim_{P_1} calcule $S'_b = S - S'_a$. Le simulateur a ainsi généré la vue $View_{Sim_{P_1}} = \{U, C', A', B', S'_a, S'_b, S\}$. La sécurité sémantique du protocole ainsi que les nombres aléatoires utilisés permettent de conclure sur l'indistinguabilité de $View_{P_1}$ et $View_{Sim_{P_1}}$.

Deuxième cas : $P_i, i \in \{3..n\}$ est corrompu. Une exécution réelle donne à P_1 la vue $View_{P_i} = \{v_i, r_i, C, A, B, \Delta_i, \gamma\}$. Sachant que le simulateur Sim_{P_i} connaît v_i et r_i, décrivons l'algorithme permettant de générer $View_{Sim_{P_i}} = \{v_i, r_i, C', A', B', \Delta'_i, \gamma'\}$. Pour simuler C', le simulateur tire $n-1$ valeurs aléatoires z_i, et chiffre chacune d'entre elles en utilisant la clef publique du P_i associé. L'ensemble des $n-1$ valeurs contenues dans B, exceptée β_{i+1}, sont générées en utilisant de nouvelles valeurs w_i prises au hasard, puis chiffrées avec la clef de P_i. Il utilise ensuite w_i comme valeur pour simuler Δ'_i et $\beta'_{i+1} = E_{i+1}(w_i + r_i)$ si $i \leq n-1$ ou $\beta'_2 = E_2(w_n + r_n)$ sinon. Pour les α'_j, avec $j \neq i$, Sim_{P_i} chiffre les aléas t_i. Pour α'_i, il lui suffit de calculer $\alpha'_i = c_i^{v_i} E_1(-r_i)$, puisque le simulateur connaît v_i et r_i du tiers de confiance. Pour la valeur γ', le simulateur chiffre un nouveau nombre aléatoire avec la clef publique de P_1. En utilisant les mêmes arguments que précédemment, on en déduit que la vue $View_{P_i}$ n'est pas distinguable de celle générée, soit $View_{Sim_{P_i}}$.

Troisième cas : P_2 est corrompu. Décrivons finalement l'algorithme de Sim_{P_2} permettant d'obtenir $View_{Sim_{P_2}} = \{v_2, r_2, s, C', A', B', \Delta'_2, S'_b, \gamma'\}$, avec les coefficients v_2, r_2 et s donnés au simulateur. La seule différence entre un joueur P_i $(i > 2)$ et P_2 se situe dans la phase de sommation, où le joueur P_2 génère un nombre aléatoire s et apprend S_b lors de l'exécution du protocole. En conséquence, l'algorithme précédemment défini pour $Sim_{P_i}, i \in \{3..n\}$ peut être utilisé à nouveau pour obtenir une vue partielle (c.-à-d. pour les valeurs communes aux deux vues). Détaillons l'algorithme permettant de simuler les coefficients manquants, soient S'_b et γ'. Afin de générer S'_b, Sim_{P_2} calcule $S'_b = \Delta'_2 - s$, en utilisant le Δ'_2 obtenu dans la vue partielle. Ensuite, il calcule $\gamma' = E_1(S'_b)$. De fait de la sécurité du cryptosystème E ainsi que de l'aléa, la vue simulée est indistinguable de celle obtenue lors d'une exécution réelle.

Précisons que dans le cas où $n = 3$, P_2 est capable de déduire la valeur aléatoire utilisée par le troisième joueur dans une exécution réelle, en calculant $r_3 = \Delta_2 - s$. Dans le cas idéal, l'adversaire obtient $r'_3 = \Delta'_2 - s$. Cependant, comme r_3 est une valeur aléatoire, P_2 n'est pas capable de faire la distinction avec une autre valeur aléatoire r'_3. Cela s'explique par le fait que seules les valeurs aléatoires du joueur corrompu sont utilisées comme entrées dans l'algorithme du simulateur : les autres

valeurs aléatoires utilisées au cours d'une exécution réelle ne sont pas contraintes à être les mêmes. □

Remarque 35. *En utilisant le modèle de preuve reposant sur la comparaison entre une exécution réelle et idéale, le protocole de sommation est sûr. Cependant, dans le cas où seuls deux joueurs sont impliqués dans l'exécution du protocole, le joueur P_2 est capable de retrouver la valeur secrète r_3 de l'autre joueur. Le protocole n'est donc pas safe. Toutefois,ce dernier est utilisé comme sous-protocole dans YTPSS, et sur des valeurs aléatoires, qui n'apportent aucune information sur les valeurs secrètes des autres joueurs. Ainsi, la connaissance de r_3 par P_2 ne compromet pas la sanité du protocole.*

5.4.2 • ADVERSAIRES MALICIEUX (ISOLÉS OU COOPÉRANTS)

Comme précédemment, nous avons utilisé le logiciel de vérification automatique ProVerif afin de trouver des attaques sur le protocole. Une attaque similaire à *DSDP* a été trouvée lorsque le joueur maître ainsi que deux autres joueurs P_{i-1} et P_{i+1} coopèrent. Ils sont ainsi capables de retrouver la valeur secrète v_i du joueur P_i.

A contrario de *DSDP*, l'attaque où seuls Alice et Charlie sont corrompus ne permet pas de retrouver la valeur v_2 de Bob. Cela s'explique simplement par le fait que Charlie n'obtiendra que la valeur aléatoire masquée $r_2 + s$ par le *salary sum*, et pas la valeur r_2 seule. Il lui est donc impossible d'en déduire des informations sur la valeur secrète de Bob.

Concernant l'efficacité, *YTP-SS* est légèrement plus léger que *DSDP*. Durant la seconde phase, il n'est plus nécessaire d'effectuer des opérations homomorphes, et l'utilisation d'un cryptosystème tel que RSA-OAEP [FOPS01, JMKR16] permet d'effectuer les chiffrements plus efficacement.

En utilisant cette nouvelle version *YTP*, il est possible d'appliquer la méthode du placement aléatoire précédemment définie dans la partie (4.2.3) à l'instar de *DSDP*. Ainsi, le théorème (28) donne le nombre d'occurrences nécessaire en fonction du niveau de sécurité désiré. Comme nous l'avons vu précédemment, un joueur honnête n'est pas vulnérable s'il est placé avec un joueur honnête pendant au moins un tour. Les résultats obtenus sont donc similaires à ceux décrits dans (4.2.3).

6 ■ COMPARAISON ET IMPLÉMENTATION DES PROTOCOLES

D'un point de vue général, les deux protocoles *DSDP* et *YTP* suivent le même modèle : une première phase permettant d'échanger les entrées secrètes des joueurs, et une seconde servant à supprimer l'aléa ajouté. La figure (6) montre la procédure

générale suivie des protocoles. On remarque cependant que les stratégies utilisées pendant la première phase sont en quelque sorte duales. Dans le cas de *DSDP*, la méthode employée pourrait être qualifiée de *tous-pour-un* tandis que la seconde est plutôt semblable à une tactique *un-pour-tous*. La seconde phase des deux protocoles diffère : *DSDP* s'appuie sur une topologie circulaire, alors que *YTP* s'apparente une approche bi-partie. Dans les deux cas, tels que décrits, les protocoles ne sont pas sûrs contre d'éventuelles coopérations d'adversaires malicieux. Il est néanmoins possible d'obtenir un compromis entre la sécurité et l'efficacité en utilisant la technique du placement aléatoire (4.2.3), elle-même adaptable afin de garantir une sécurité à un facteur ε prédéfini.

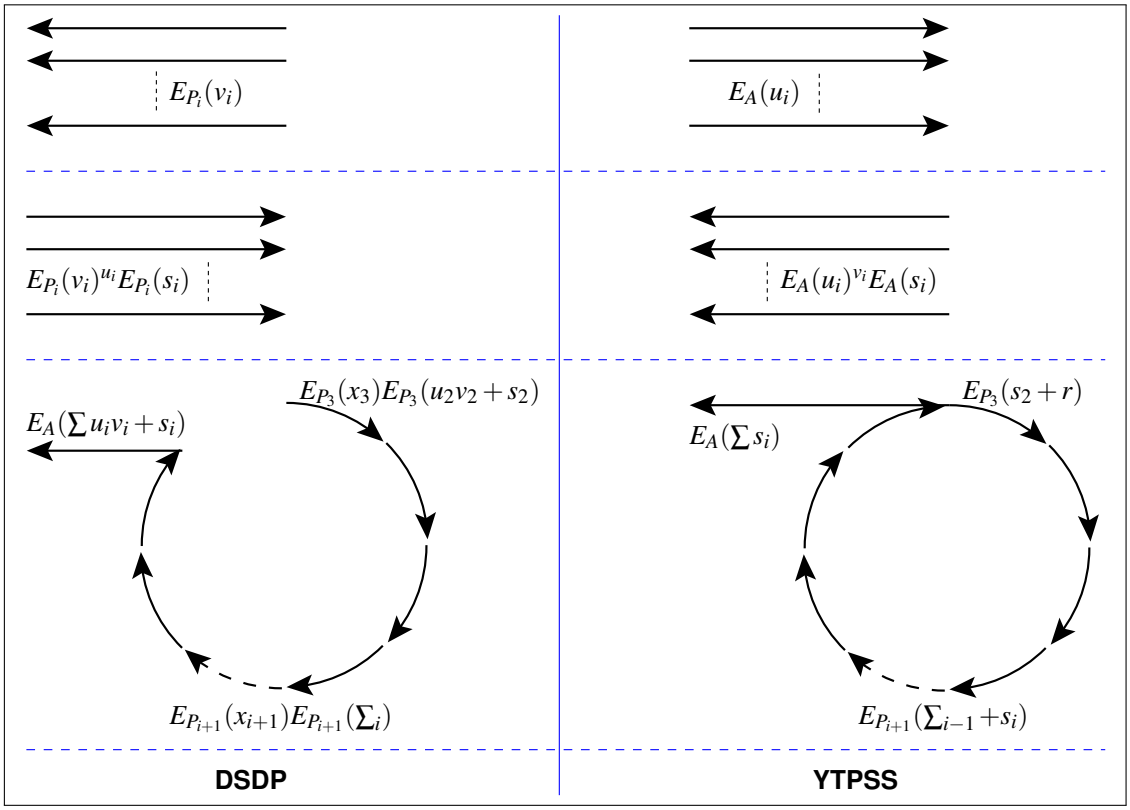

FIGURE 6 – Approches duales de protocoles sûrs et distribués (à gauche *DSDP*, voir la section (4) ; à droite *YTP-SS*, voir la section 5)

6.1 | COMPARAISON ENTRE *DSDP* ET *YTP-SS*

Les protocoles *YTP-SS* et *DSDP* ont tous les deux une complexité linéaire, avec un léger avantage pour *YTP-SS* qui n'utilise pas de cryptosystème homomorphe

lors de la seconde phase, dite de *salary sum*. La figure (7) montre que lors d'un exécution réelle, le protocole *YTP-SS* utilisant le cryptosystème de Paillier lors de la première phase puis RSA-OAEP pour la seconde est plus efficace que *DSDP* s'appuyant uniquement sur le cryptosystème de Paillier.

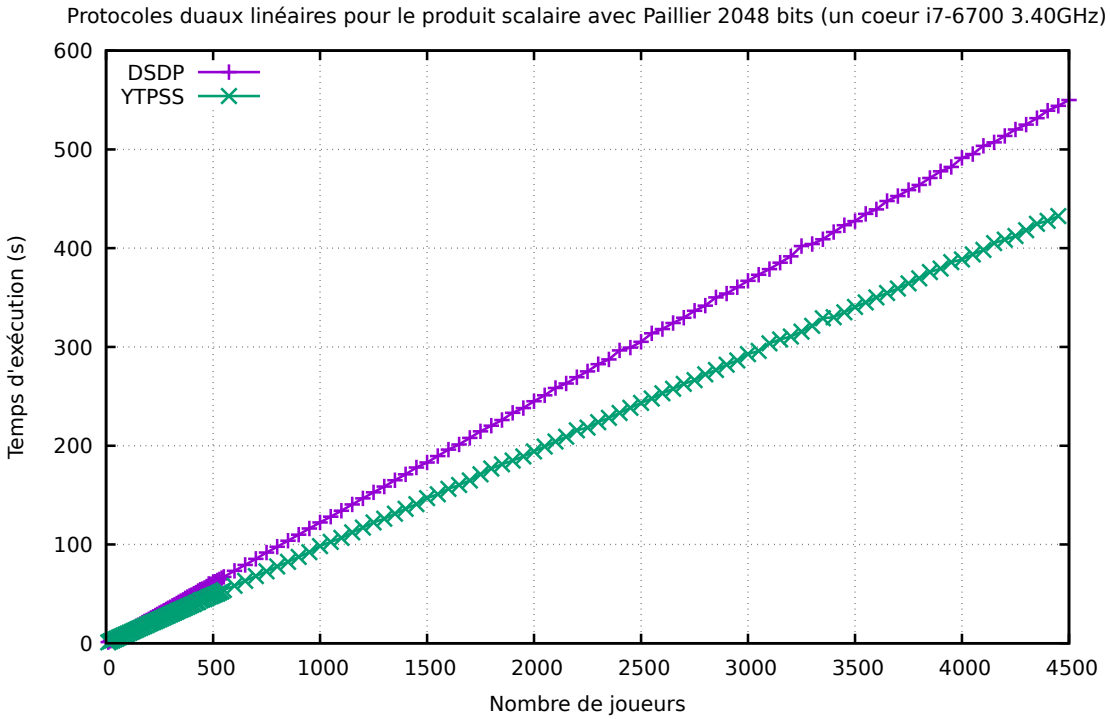

FIGURE 7 – Temps d'exécution de *DSDP* et de son dual *YTP-SS*

6.2 | INFLUENCE DU PLACEMENT ALÉATOIRE SUR L'EFFICACITÉ DE *DSDP*

Globalement, le placement aléatoire décrit dans l'algorithme (26) peut garantir un niveau de sécurité quelconque, au prix d'un certain nombre de répétitions du protocole. Le nombre d'occurrences est fixé en amont de la première exécution du protocole, et chacune des exécutions peut être effectuée en parallèle. Ainsi, si le volume global de communication est obtenu en multipliant le coût d'une exécution par le nombre de répétitions, le nombre de tour reste constant. La figure (8) montre l'influence du paramètre de sécurité sur le temps d'exécution du protocole. En particulier, le cas moyen ainsi que le pire cas avec $\varepsilon = 128$ sont représentés. Le graphe est complété par les temps d'exécution obtenus par P-MPWP, ainsi que *DSDP* sans placement aléatoire.

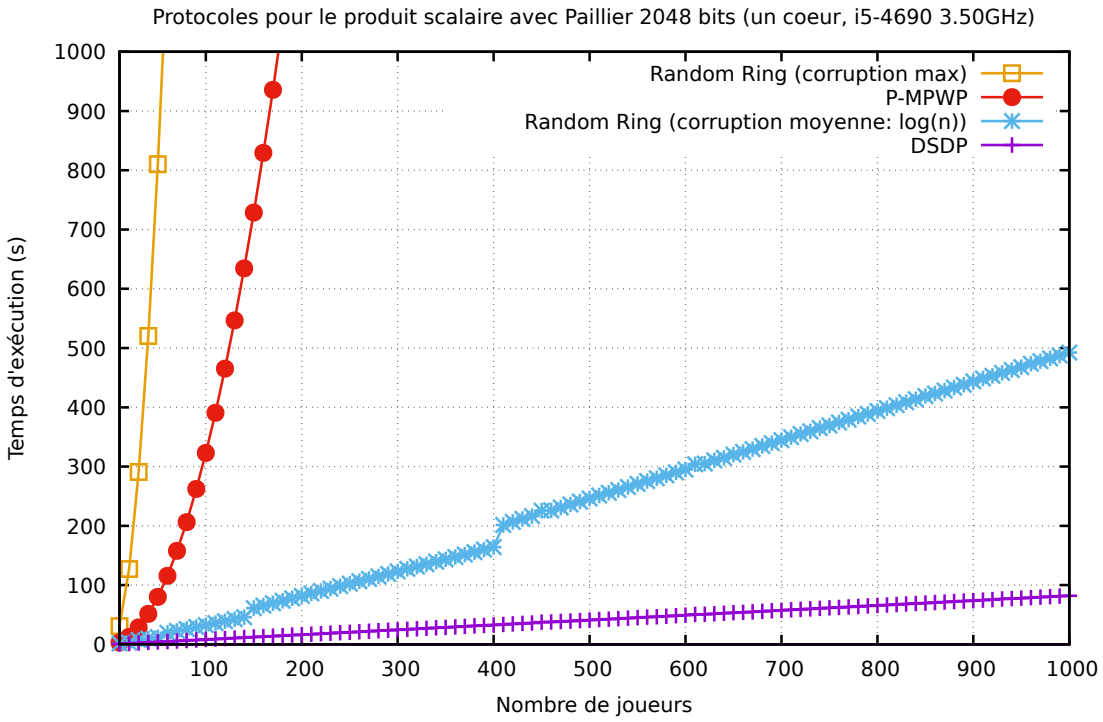

FIGURE 8 – Temps d'exécutions (linéaire et quadratique) des protocoles en fonction du paramètre de sécurité

En utilisant une sécurité moyenne, c.-à-d. suffisante pour éviter des attaques en moyenne, les protocoles montrent des performances quasi-lineaires. Pour obtenir une sécurité dans le cas où la majorité des participants est corrompue, nous voyons qu'il est encore difficile d'appliquer directement le protocole si le nombre de participants devient important. Précisons que les échelons vus sur la courbe correspondent à l'arrondi du terme $\log(n)$ à l'entier suivant, impliquant donc une répétition supplémentaire du protocole avec placement aléatoire des joueurs.

7 ■ APPLICATION À L'AGRÉGATION DE CONFIANCE

Nous nous intéressons désormais à l'application des protocoles précédents au calcul de l'agrégation de confiance. Comme nous l'avons vu dans la partie (2.2), il est possible de réduire cette opération à des opérations de puissance de la matrice d'adjacence associée au graphe de confiance. Nous montrons dans cette partie comment utiliser les protocoles calculant un produit scalaire sûr pour effectuer

cette opération.

7.1 | OPÉRATIONS HOMOMORPHES SUR LES COUPLES

Soit E une fonction de chiffrement, commençons par définir un morphisme naturel sur les paires, afin de pouvoir l'appliquer aux valeurs de confiance par la suite :

$$E(\langle a,\ b\rangle) = \langle E(a),\ E(b)\rangle \tag{V.4}$$

A partir de cette définition, il est possible d'étendre les propriétés homomorphes aux paires de sorte que les calculs d'agrégation (séquentiels et parallèles) puissent être effectués homomorphiquement, tant qu'une entrée est donnée en clair.

Lemme 36. *Soit $E : P \to C$ une fonction de chiffrement vérifiant les propriétés homomorphes (II.1) et (II.2). Nous pouvons définir deux fonctions **Mul** et **Add** sur $P^2 \times P^2 \to C^2$ telles que :*

$$\begin{aligned} Mul(E(\langle a,\ b\rangle); \langle c,\ d\rangle) &= E(\langle a,\ b\rangle \bigstar \langle c,\ d\rangle) \\ &= \langle E(a)^c E(b)^d,\ E(a)^d E(b)^c\rangle \\ Add(E(\langle a,\ b\rangle); \langle c,\ d\rangle) &= E(\langle a,\ b\rangle \maltese \langle c,\ d\rangle) \\ &= \langle E(a)E(c)E(a)^{-c},\ E(b)^d\rangle \end{aligned}$$

De plus, ces deux fonctions peuvent être appliquées si un des deux couples (par exemple, $\langle a,\ b\rangle$) est chiffré et l'autre couple ($\langle c,\ d\rangle$) est donné en clair i.e. elles sont définies sur : $C^2 \times P^2 \to C^2$.

Démonstration. En utilisant les propriétés homomorphes de la fonction de chiffrement E, nous avons les égalités suivantes :

1. $E(a)^c E(b)^d = E(ac + bd)$;
2. $E(a)^d E(b)^c = E(ad + bc)$;
3. $E(b)^d = E(bd)$;
4. $E(a)E(c)E(a)^{-c} = E(a + c + a(-c))$.

Les trois premières égalités nous montrent qu'en utilisant un couple $\langle a,\ b\rangle$ chiffré et un couple $\langle c,\ d\rangle$ clair, nous obtenons un résultat chiffré. Pour la dernière égalité, il suffit d'utiliser la clef publique adéquate pour chiffrer uniquement le coefficient c. □

Grâce au lemme (36), nous avons donc montré qu'il est possible de calculer l'agrégation de confiance de manière privée, c.-à-d. en appliquant une fonction de chiffrement sur les couples modélisant la confiance.

7.2 | EVALUATION SÛRE ET DISTRIBUÉE DE LA CONFIANCE

En utilisant les résultats précédents (7.1), nous montrons qu'il est possible d'utiliser les protocoles calculant un produit scalaire (présentés dans les sections 4 et 5) de manière distribuée et sûre pour calculer l'agrégation de confiance.

Corollaire 37. *Le protocole DSDP décrit par l'algorithme (20) est applicable aux valeurs de confiance, à condition que les valeurs aléatoires r_i soient inversibles pour* ✠.

Démonstration.
- u_i, v_i, r_i, c_i, α_i, β_i, Δ_i, γ sont maintenant écrites sous forme de couple. Par exemple, les entrées secrètes u_i représentent désormais des valeurs de confiance, d'où $u_i = \langle \mathcal{T}_i,\ \mathcal{D}_i \rangle$;
- Les fonctions de chiffrement et déchiffrement ($E(v_i)$, $D(\beta_i)$, $E(\Delta_i)$, $E(\gamma)$, etc.) sont appliquées sur des couples, en utilisant le morphisme précédemment défini ($E(\langle a,\ b\rangle) = \langle E(a),\ E(b)\rangle$) ;
- α_i est $E((u_i \bigstar v_i) ✠ r_i) = \mathrm{Add}(\mathrm{Mul}(E(v_i); u_i); r_i)$, et peut toujours être calculé par P_1, puisque $c_i = E(v_i)$, et que u_i et r_i sont connues par ce dernier ;
- De plus, $\beta_i = E(\alpha_i ✠ \Delta_i) = \mathrm{Add}(E(\alpha_i); \Delta_i)$.
- Finalement, comme l'opérateur ✠ est commutatif, S peut être calculé en ajoutant les inverses de r_i pour ✠.

□

En utilisant la définition [DH12, Definition 11], la d-agrégation de confiance est un produit scalaire légèrement modifié pour ne pas inclure la valeur $u_1 v_1$. En conséquence, il suffit de remplacer la ligne (3) de l'algorithme (31), $s \leftarrow a_{i,j} b_{i,j}$, par l'élément neutre de ✠, c.-à-d. $s \leftarrow \langle 0,\ 1\rangle$.

Encodage des valeurs de confiance. Nous devons maintenant encoder les valeurs de confiance, qui sont des proportions dans $[0,1]$ dans $\mathbb{D} = \mathbb{Z}/N\mathbb{Z}$. Avec n participants, nous allons utiliser une précision fixe 2^{-p} telle que $2^{n(2p+1)} < N \leq 2^{n(2(p+1)+1)}$. Nous arrondissons les coefficients de confiance à $\lfloor x 2^p \rfloor \mod N$ de $[0,1] \to \mathbb{D}$. Nous obtenons ainsi la borne suivante pour le produit scalaire :

Lemme 38. *Si tous les coefficients u_i et v_i sont compris entre 0 et $2^p - 1$, alors les coefficients de $S = ✠_{i=1}^{n} (u_i \bigstar v_i)$ sont bornés par $2^{n(2p+1)}$ en valeur absolue.*

Démonstration. Pour tout u, v, les coefficients de $(u \bigstar v)$ sont compris entre 0 et $(2^p-1)(2^p-1) + (2^p-1)(2^p-1) = 2^{2p+1} - 2^{p+2} + 2 < 2^{2p+1} - 1$, avec p un entier positif. Par récurrence, en agrégeant k de ces coefficients avec ✠, la valeur absolue appliquée sur le résultat est inférieure à $2^{k(2p+1)} - 1$. □

En conséquence, en utilisant un modulo N encodé sur 2048 bits, avec $n \leq 4$ dans le protocole ***ESDP*** (algorithme 31), le lemme précédent (38) permet d'obtenir une précision proche de $2^{-255} \approx 10^{-77}$.

Reprenons l'exemple du produit de matrices appliqué au calcul de la confiance entre les autorités de certification. Dans l'article [DKBH13], les auteurs ont identifié 1800 certificats pour ces entités, contrôlés par environ 700 organisations différentes. Nous avons donc exécuté le protocole ***YTP-SS*** en parallèle, à la manière de celle décrite dans $PDSMM_n$, avec $n = 700$ joueurs. En utilisant une métrique de confiance simple, il aura fallu moins de 430 secondes pour que le protocole termine le calcul du produit matriciel privé d'une matrice de taille 700×700 avec elle-même. D'un point de vue applicatif, chaque autorité de certification peut obtenir une vue globale de la confiance en utilisant la confiance de ses voisins en moins de 7 minutes. Afin d'obtenir une confiance globale plus juste, c.-à-d. en prenant en compte la confiance des noeuds plus distants de chaque autorité, il est nécessaire de réitérer le processus. Cependant, cela montre qu'il est possible, en pratique, de mettre à jour la confiance de manière privée, entre un nombre important d'autorités de certification. Ainsi, il est possible d'avoir une vision plus nuancée de la confiance accordée dans ces entités.

8 ■ CONCLUSION

Dans ce chapitre, nous avons développé deux protocoles dénommés DSDP et YTPSS, qui permettent de calculer de manière sûre face à des adversaires semi-honnêtes (au sens du MPC) des produits scalaires. Les deux protocoles ont des complexités en calculs et communications bornées par $O(n)$. Dans une approche parallèle, seules cinq phases de communications sont requises. Dans le tableau (2), nous détaillons le volume total des communications de chaque protocole, qui est cubique pour MPWP [DGK10], quadratique pour son adaptation avec le cryptosystème de Paillier P-MPWP, et finalement linéaire pour DSDP et YTPSS. Nous précisons aussi le nombre de tours qu'il est nécessaire d'effectuer pour échanger ce volume de données. Nous explicitons aussi les coûts en communication en fonction du niveau de sécurité souhaité, notamment grâce à l'utilisation du placement via l'anneau aléatoire. Dans ce cas, le protocole offre une résistance face à des adversaires malicieux (M), alors que les autres protocoles ne sont sûrs que face à des adversaires honnête-mais-curieux (H).

Les protocoles que nous avons présentés permettent de calculer efficacement l'agrégation de confiance dans un réseau, où cette dernière est définie comme un produit non usuel de matrices reposant sur un anneau intègre. Ce système constitue une alternative au modèle actuels des ancres de confiance. Son déploiement pourrait permettre la définition d'une politique de sécurité plus fine, profitant de ce système

Protocoles	Volume	Tours	Sécurité
P-MPWP (§ 3.2)	$n^{2+o(1)}$	$O(n)$	?
Alg. 20 ($DSDP_n$)	$n^{1+o(1)}$	$O(n)$	H
Alg. 31 ($PDSMM_n$)	$n^{1+o(1)}$	5	H
Alg. 33 ($YTP\text{-}SS_n$)	$n^{1+o(1)}$	$O(n)$	H
Alg. 33 (en parallèle)	$n^{1+o(1)}$	5	H
Alg. 26 combiné avec Alg. 20 ou Alg. 33 (sécurité en moyenne)	$n^{1+o(1)}$	5	M
Alg. 26 combiné avec Alg. 20 ou Alg. 33 (sécurité selon ε)	$n^{2+o(1)} \ln\left(\frac{1}{\varepsilon}\right)$	5	M

TABLE 2 – Complexités en communication de nos protocoles

de réputation. Typiquement, le navigateur pourrait faire parvenir ces évaluations à l'utilisateur. Selon son niveau d'expertise et son usage, il pourrait alors laisser le navigateur prendre la décision, ou au contraire décider par lui-même si il doit faire confiance ou non à cette ancre.

De nombreuses améliorations sont encore possibles pour ce système. Tout d'abord, nous utilisons actuellement un algorithme sous optimal pour le produit de matrices. L'adaptation de nos protocoles à un algorithme de produit matriciel efficace pourrait être intéressante, afin de réduire les complexités. Il faudrait alors être capable de découper nos matrices en blocs, et de combiner la récursivité de l'algorithme avec la sécurité des protocoles. Un autre axe d'amélioration concerne l'utilisation du cryptosystème de Schmidt-Samoa et Takagi [SST05] à la place de celui de Paillier. Ainsi, nous pourrions bénéficier de la souplesse du cryptosystème de Benaloh et de l'efficacité de celui de Paillier.

En termes de sécurité, il serait intéressant de développer une version sûre au sens du MPC de la version parallèle du protocole calculant un produit de matrices. En combinant le paradigme de preuves autorisant la composition de Canetti [Can01] ainsi qu'en utilisant des valeurs aléatoires ajoutées entre chaque tour puis supprimées à la fin de l'exécution, il semble possible d'obtenir un protocole vérifiant une sécurité plus générale. Au niveau des preuves de sécurité assistées, nous pourrons affiner notre modèle ProVerif afin que ce dernier réussisse à terminer lorsque les preuves à divulgation nulle de connaissance et la stratégie du placement aléatoire sont utilisés. Enfin, concernant le placement aléatoire, nous pourrions l'améliorer afin qu'il apporte une sécurité face à des attaquants capables de s'adapter (*adaptive* en anglais). Ces derniers ont la capacité de corrompre des ensembles de joueurs variant tout au long de l'exécution du protocole. Une première solution

consiste à utiliser un horodateur agissant comme tiers de confiance : les adversaires ne seraient alors plus en mesure de prévoir leur placement à l'avance, et leurs possibilités d'attaque seraient réduites.

CHAPITRE VI

CONCLUSION ET PERSPECTIVES

1 ■ CONCLUSION

Dans cette thèse, nous avons abordé différents thèmes en lien avec la sécurité et les infrastructures de gestion de clefs. Nous avons détaillé l'architecture d'un nouvel équipement visant à améliorer la sécurité des réseaux par rupture protocolaire. Nous avons ensuite proposé une infrastructure de gestion de clefs alternative, ayant pour objectif de démocratiser l'usage des certificats pour les utilisateurs finaux, mais qui est aussi appropriée au déploiement d'un module de rupture protocolaire. Enfin, nous avons exposé des protocoles multi-parties sûrs, permettant de calculer efficacement des produits scalaires. Ils sont, par extension, applicables au calcul de l'agrégation de confiance entre les autorités de certification. Nous obtenons finalement une solution permettant d'améliorer le fonctionnement des magasins d'ancres de confiance, présents dans PKIX comme dans LocalPKI.

2 ■ PERSPECTIVES DE RECHERCHE

Si ces approches constituent une première ébauche de réponses aux problématiques évoquées, elles sont aussi à voir comme des ouvertures à de nouveaux travaux de recherche. En effet, les axes d'amélioration sont nombreux :

- L'implémentation de la totalité des contre-mesures permettant qu'un outil de vérification automatique puisse terminer son analyse des protocoles DSDP et YTPSS. Actuellement, nous avons atteint les limites des outils utilisés, et complété les preuves manuellement. A l'aide d'une théorie équationnelle plus fine, nous pourrions réduire le nombre de traces à vérifier. De plus, nous pourrions aussi utiliser un outil permettant une preuve interactive, où la réduction du nombre de traces serait manuelle. Cette seconde approche nécessiterait ensuite de montrer que la preuve est toujours complète ;

- Un remaniement des protocoles proposés pour le calcul matriciel parallèle, afin d'étendre leur sécurité à des modèles plus génériques. La première étape consisterait à ajouter des valeurs aléatoires à chaque tour, afin de rendre le protocole sûr au sens habituel du calcul multi-parties. Une autre amélioration consisterait à généraliser le modèle de preuve par celui dit *universellement composable*. Cette approche donnerait un modèle d'exécution et de sécurité plus proche d'une exécution réelle des protocoles entre les autorités de certification ;

- Le développement de protocoles multi-parties dédiés au calcul du produit de matrices rapides. Nous utilisons actuellement l'algorithme classique du produit, qui est sous-optimal (*i.e.* dont la complexité est cubique en la taille

des entrées). L'utilisation d'un algorithme de produit de matrices rapide permettrait peut-être d'obtenir de meilleurs résultats. Son développement est un vrai défi, car il nécessite d'obtenir une preuve de sécurité reposant sur un algorithme récursif ;

- Etudier plus finement la sécurité calculatoire de LocalPKI. Nous avons démontré formellement la sécurité symbolique, l'idée serait d'obtenir des bornes théoriques sur l'avantage des adversaires, dans la veine de [BFPW07], et de les comparer ensuite à celles obtenues avec leur modélisation générale des PKI.

3 ■ PERSPECTIVES DE DÉVELOPPEMENT

Nos travaux pourraient aussi conduire à des développements ayant pour objectif une industrialisation, ou une application dans des domaines plus spécifiques. Nous pensons que les preuves de concept développées dans cette thèse pourraient être étendues, avec :

- L'implémentation de LocalPKI : il faudrait développer des programmes facilement distribuables pour chacune des entités. Le premier programme à implémenter aurait pour but de permettre aux notaires de gérer facilement leur base de données ainsi que les réponses aux demandes de validités. Le second serait dédié aux agences locales, pour leur permettre, ainsi qu'aux usagers, de générer les certificats. Comme pour les travaux associés à LocalPKI, une collaboration est en cours avec *NTX Research* ;

- L'analyse de sécurité et de performance de l'implémentation d'un module de rupture protocolaire, avec l'objectif d'obtenir une certification vérifiant de type *Evaluation Assurance Level* pour le module ARAMIS. Nous avons fourni une première analyse en nous inspirant des profils de protection, qui pourrait être étendue pour atteindre le niveau *EAL4* ;

- Le développement de logiciels associés au module, facilitant son déploiement et sa configuration. En particulier, la configuration des règles de filtrage est extrêmement difficile compte-tenu du nombre de variables à prendre en compte. La réalisation d'un programme d'apprentissage appliqué sur les flux réels lors d'une intégration pourrait faciliter ce processus en automatisant la configuration des filtres ;

- La généralisation du système d'évaluation de confiance aux systèmes d'authentification. Nous pourrions imaginer que lors de l'authentification des

usagers, ou lors de la remise de son certificat, l'utilisateur possède une mesure de confiance le concernant. Le serveur serait alors en mesure de décider s'il souhaite échanger avec l'usager, ou encore lui attribuer des droits différents ;

- Une adaptation de LocalPKI avec nos magasins de confiance pour la signature d'applications. En prenant le cas des smartphones et de leurs applications signées, l'utilisation de LocalPKI permettrait de générer simplement un certificat par application. L'utilisateur aurait alors la possibilité d'évaluer la valeur de la confiance dans les entités en charge de délivrer ou valider un certificat, et donc de choisir d'installer ou non cette application. Les objets connectés constituent un autre exemple où LocalPKI permettrait de faciliter leur intégration au sein d'un réseau grâce à ces certificats auto-signés.

4 ■ PERSPECTIVES À PLUS LONG TERME

Nous concluons finalement ces perspectives avec quelques idées de projets de recherche plus vastes :

- Les infrastructures de gestion de clefs permettent, entre autres, l'authentification d'un utilisateur possédant une clef. Leur fonctionnement repose cependant sur l'hypothèse qu'un tiers de confiance vérifie en amont l'identité de l'utilisateur, avant de la lier à sa clef publique. D'un point de vue théorique, les avancées de la cryptographie ouvrent de nouvelles possibilités. Par exemple, combiner l'utilisation de LocalPKI avec des schémas cryptographiques reposant sur des caractéristiques des utilisateurs (*Attribute-Based Encryption*) semblent intéressants à appliquer aux PKI, dans la veine de la combinaison PKI et *Identity-Based Encryption* [Gen03] ;

- Nous nous sommes intéressés à des protocoles cryptographiques homomorphes, dont la sécurité repose sur des problèmes supposés difficiles dans notre paradigme actuel. Les avancées dans le domaine de l'informatique quantique pourraient pourtant les rendre caduques. Nous pensons donc qu'il serait intéressant de développer un système d'évaluation de la confiance utilisant des cryptosystèmes sûrs dans un paradigme quantique. Actuellement, de tels cryptosystèmes existent, mais sont encore trop coûteux pour être exploités. Dans notre système d'agrégation de confiance, où nous n'avons cependant besoin que de cryptosystèmes semi-homomorphes. Il serait intéressant d'étudier des cryptosystèmes reposant sur des codes (tels que [Mce78]), et de les associer à des protocoles multi-parties permettant des calculs homomorphes de manière interactive (comme [CDN01]) afin d'obtenir un

compromis efficace entre la complexité en temps et en communication.

BIBLIOGRAPHIE

[Aas14] Josh Aas. Let's encrypt : Delivering ssl/tls everywhere. *Let's Encrypt*, 18, 2014.

[ABB+05] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International Conference on Computer Aided Verification*, pages 281–285. Springer, 2005.

[ADK14] Myrto Arapinis, Stéphanie Delaune, and Steve Kremer. Dynamic tags for security protocols. *Logical Methods in Computer Science*, 10(2 :11), June 2014.

[AEC07] Artak Amirbekyan and Vladimir Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *AusDM 2007*, volume 70 of *CRPIT*, pages 209–214, 2007.

[AEDS03] Mikhail J Atallah, Hicham G Elmongui, Vinayak Deshpande, and Leroy B Schwarz. Secure supply-chain protocols. In *E-Commerce, 2003. CEC 2003. IEEE International Conference on*, pages 293–302. IEEE, 2003.

[ALV03] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1) :695–740, 2003.

[ANS14] ANSSI. Référentiel général de sécurité, annexe b1, mécanismes cryptographiques, règles et recommendations concernant le choix et le dimensionnement des mécanismes cryptographiques. 2014.

[ANS15] ANSSI. Protection profile of an industrial firewall. Agence nationale de la sécurité des systèmes d'information, July 2015.

[AR00] Martın Abadi and Phillip Rogaway. Reconciling two views of cryptography. In *Proceedings of the IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer, 2000.

[Azi13] Ashar Aziz. The evolution of cyber attacks and next generation threat protection. In *RSA Conference*, 2013.

[Bat11] Necdet Batir. Sharp bounds for the psi function and harmonic numbers. *Mathematical inequalities and applications*, 14(4), 2011.

[BBB+07] Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid. Sp 800-57. recommendation for key management, part 1 : General (revised). Technical report, Gaithersburg, MD, United States, 2007.

[BC12] Tom Barker and Chris Cheese. The application of data diodes for securely connecting nuclear power plant safety systems to the corporate it network. In *7th IET International Conference on System Safety, incorporating the Cyber Security Conference 2012*, pages 1–6, Oct 2012.

[BCD+] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Sasa Radomirovic, Ralf Sasse, Lara Schmid, and Benedikt Schmidt. Tamarin manual. https://tamarin-prover.github.io/manual/. Accessed : 2018-05-04.

[BCK+14] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI : Attack resilient public-key infrastructure. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 382–393, November 2014.

[BDD+17] Benoît Badrignans, Vincent Danjean, Jean-Guillaume Dumas, Philippe Elbaz-Vincent, Sabine Machenaud, Jean-Baptiste Orfila, Florian Pebay-Peyroula, François Pebay-Peyroula, Marie-Laure Potet, Maxime Puys, Jean-Luc Richier, and Jean-Louis Roch. Security architecture for point-to-point splitting protocols. In *IEEE World Congress on Industrial Control Systems Security, Cambridge, UK*, page 8, December 2017.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In KennethG. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer Berlin Heidelberg, 2011.

[Ben94] Josh Benaloh. Dense probabilistic encryption. In *First Annual Workshop on Selected Areas in Cryptography*, pages 120–128, Kingston, ON, May 1994.

[BFM04] Eric J. Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *IEEE Conf. International Infrastructure Survivability Workshop (IISW '04). Institute for Electrical and Electronics Engineers*, 2004.

[BFPW07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI : Security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007 : 10th International Conference on Practice and Theory in Public-Key Cryptography Beijing, China, April 16-20, 2007. Proceedings*, pages 458–475, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BGT11] Samia Bouzefrane, Khaled Garri, and Pascal Thoniel. A user-centric PKI based-protocol to manage FC^2 digital identities. *International Journal of Computer Science*, 8(1) :1694–0814, 2011.

[Bin17] Ashok Bindra. Securing the power grid : Protecting smart grids and connected power systems from cyberattacks. *IEEE Power Electronics Magazine*, 4(3) :20–27, 2017.

[Bla04] Bruno Blanchet. *Cryptographic Protocol Verifier User Manual*, 2004.

[Bla14] Bruno Blanchet. Automatic verification of security protocols in the symbolic model : the verifier ProVerif. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 54–87. Springer, 2014.

[BM10] Jason Bau and John C. Mitchell. A security evaluation of DNSSEC with NSEC3. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.

[BMV03] David Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security – ESORICS 2003 : 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings*, pages 253–270, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Boy09] Stuart A. Boyer. *Scada : Supervisory Control And Data Acquisition*. International Society of Automation, USA, 4th edition, 2009.

[BSS05] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless public key encryption without pairing. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2005.

[Can01] Ran Canetti. Universally composable security : A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[CC04] Hubert Comon-Lundh and Véronique Cortier. Security properties : two agents are sufficient. *Sci. Comput. Program.*, 50(1-3) :51–71, 2004.

[CD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property : How to get rid of some algebraic properties. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[CD+15] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001 : International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings*, pages 280–300, Berlin, Heidelberg, 2001. Springer.

[cen] Censys. censys.io. Accessed : 2018-05-03.

[CKV+02] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2) :28–34, December 2002.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4) :225–259, 2011.

[Coo08] Dave Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[Cor16] Marie-Angela Cornelie. *Implementations and protections of software and hardware cryptographic mechanisms*. Thèse, Université Grenoble Alpes, April 2016.

[DA01] Wenliang Du and M. J. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, ACSAC '01, pages 102–110, December 2001.

[Del06] Stéphanie Delaune. An undecidability result for AGh. *Theor. Comput. Sci.*, 368(1-2) :161–167, December 2006.

[DGK10] Shlomi Dolev, Niv Gilboa, and Marina Kopeetsky. Computing multiparty trust privately : in $O(n)$ time units sending one (possibly large) message at a time. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1460–1465, New York, NY, USA, 2010. ACM.

[DH12] Jean-Guillaume Dumas and Hicham Hossayni. Matrix powers algorithm for trust evaluation in PKI architectures. In Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi, editors, *STM'2012, Proceedings of the eigth International Workshop on Security and Trust Management (co-ESORICS 2012), Pisa, Italy*, volume 7783 of *Lecture Notes in Computer Science*, pages 129–144, September 2012.

[dJC86] Wiebren de Jonge and David Chaum. Attacks on some rsa signatures. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 18–27, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[DKBH13] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 291–304, New York, NY, USA, 2013. ACM.

[DLM+17] Jean-Guillaume Dumas, Pascal Lafourcade, Francis Melemedjian, Jean-Baptiste Orfila, and Pascal Thoniel. LocalPKI : A user-centric formally proven alternative to PKIX. In Pierangela Samarati, editor, *14th International Conference on Security and Cryptography (SECRYPT 2017)*, page 12, July 2017.

[DLMS04] Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2) :247–311, 2004.

[DLOP16] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Private multi-party matrix multiplication and trust computations. In Pierangela Samarati, editor, *13th International Conference on Security and Cryptography (SECRYPT 2016)*, pages 61–72, July 2016.

[DLOP17] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Dual protocols for private multi-party matrix multipli-

cation and trust computations. *Computers & Security*, (71) :51–70, November 2017.

[DLR15] Jean-Guillaume Dumas, Pascal Lafourcade, and Patrick Redon. *Architectures PKI et communications sécurisées*. Dunod, 2015.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer Berlin Heidelberg, 2012.

[DT88] Denys D Davies and Kenneth CG Thompson. Firewall seal, July 19 1988. US Patent 4,758,028.

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2) :198–208, 1983.

[Eas11] Donald Eastlake. Transport layer security (tls) extensions : Extension definitions. RFC 6066, RFC Editor, January 2011. http://www.rfc-editor.org/rfc/rfc6066.txt.

[FAO10] Simon N. Foley, Wayne Mac Adams, and Barry O'Sullivan. Aggregating trust using triangular norms in the keynote trust management system. In Jorge Cuéllar, Javier Lopez, Gilles Barthe, and Alexander Pretschner, editors, *Security and Trust Management - 6th International Workshop, STM 2010, Athens, Greece, September 23-24, 2010, Revised Selected Papers*, volume 6710 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2010.

[FLA11] Laurent Fousse, Pascal Lafourcade, and Mohamed Alnuaimi. Benaloh's dense probabilistic encryption revisited. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2011.

[FMC11] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6), 2011.

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. In *Annual International Cryptology Conference*, pages 260–274. Springer, 2001.

[Gen03] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, pages 272–293, Berlin, Heidelberg, 2003. Springer-Verlag.

[GKRT04] Ramanathan V. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 403–412. ACM, 2004.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2) :270 – 299, 1984.

[Gol04] Oded Goldreich. *Foundations of Cryptography : Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[Gos16] Gossamer Laboratories. Stonesoft next generation firewall (ndpp11e3/stffe10) security target, 2016.

[HN10] Jingwei Huang and David M. Nicol. A formal-semantics-based calculus of trust. *IEEE Internet Computing*, 14(5) :38–46, 2010.

[JHZ+17] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. How to break secure boot on fpga socs through malicious hardware. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 425–442. Springer, 2017.

[JMKR16] Jakob Jonsson, Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS#1 : RSA cryptography specifications version 2.2. 2016.

[Jøs07] Audun Jøsang. Probabilistic logic under uncertainty. In Joachim Gudmundsson and C. Barry Jay, editors, *Theory of Computing 2007. Proceedings of the Thirteenth Computing : The Australasian Theory Symposium (CATS2007). January 30 - Febuary 2, 2007, Ballarat, Victoria, Australia, Proceedings*, volume 65 of *CRPIT*, pages 101–110. Australian Computer Society, 2007.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4 : Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[KHP+13] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. Accountable key infrastructure (aki) : A proposal for a public-key validation infrastructure. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 679–690, New York, NY, USA, 2013. ACM.

[Kir03] Olaf Kirch. Opensc-smart cards on linux. In *Proc. of the 10th International Linux System Technology Conference, Saarbruecken, Germany*, 2003.

[KKS15] S. Kent, D. Kong, and K. Seo. Template for a certification practice statement (cps) for the resource pki (rpki). BCP 173, RFC Editor, April 2015.

[KKSW12] S. Kent, D. Kong, K. Seo, and R. Watro. Certificate policy (cp) for the resource public key infrastructure (rpki). BCP 173, RFC Editor, February 2012.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[KMG12] Olaf M. Kolkman, Matthijs Mekking, and R. (Miek) Gieben. DNSSEC Operational Practices, Version 2. RFC 6781, December 2012.

[Koh78] Loren M Kohnfelder. *Towards a practical public-key cryptosystem.* PhD thesis, Massachusetts Institute of Technology, 1978.

[KP12] Tomi Kause and Martin Peylo. Internet X.509 Public Key Infrastructure – HTTP Transfer for the Certificate Management Protocol (CMP). RFC 6712, September 2012.

[KW16] Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 81–96. IEEE, 2016.

[LC16] Robert Lipovsky and Anton Cherepanov. Blackenergy trojan strikes again : Attacks ukrainian electric power industry. *Online at http//www. welivesecurity. com/2016/01/04/blackenergy-trojan-strikes-again-attacksukrainian-electric-power-industry*, 2016.

[Lin09] Yehuda Lindell. Secure computation for privacy preserving data mining. In John Wang, editor, *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*, pages 1747–1752. IGI Global, 2009.

[Lin17] Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.

[LLK11] Ben Laurie, Adam Langley, and E Kasper. Certificate authority transparency and auditability. *white paper*, 22, 2011.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, 2013.

[Mce78] Robert J Mceliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244 :114–116, 1978.

[MEFP08] Jose L. Muñoz, Oscar Esparza, Jordi Forné, and Esteve Pallares. H-ocsp : A protocol to reduce the processing burden in online certificate status validation. *Electronic Commerce Research*, 8(4) :255, 2008.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer.

[MG14] Stathis Mavrovouniotis and Mick Ganley. *Hardware Security Modules*, pages 383–405. Springer New York, New York, NY, 2014.

[Moh11] Payman Mohassel. Efficient and secure delegation of linear algebra. *IACR Cryptology ePrint Archive*, 2011 :605, 2011.

[Moz] Mozilla. Wosign-firefox. https://wiki.mozilla.org/CA:WoSign_Issues. Accessed : 2018-04-30.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[Mun11] Phil Muncaster. Stuxnet-like attacks beckon as 50 new scada threats discovered, 2011.

[MV14] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *black hat USA*, 2014, 2014.

[NS78] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12) :993–999, 1978.

[NSS16] NSS Labs. Next generation firewall. 2016.

[OSH13] Hamed Okhravi, Fredrick T Sheldon, and Joshua Haines. Data diodes in support of trustworthy cyber infrastructure and net-centric cyber decision support. In *Optimization and Security Challenges in Smart Power Grids*, pages 203–216. Springer, 2013.

[oST] National Institute of Standards and Technology. Security requirements for cryptographic modules, fips pub 140-2. Technical report.

[OW84] Lawrence H. Ozarow and Aaron D. Wyner. Wire-tap channel II. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *EUROCRYPT'84, Paris, France*, volume 209 of *LNCS*, pages 33–50. Springer, 1984.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology*

- *EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Pir13] John W. Pirc. Ssl performance problems. 2013.

[PY09] John Pescatore and Greg Young. Defining the next-generation firewall. *Gartner RAS Core Research Note*, 2009.

[RBBT17] Peter Rouget, Benoît Badrignans, Pascal Benoit, and Lionel Torres. Secboot—lightweight secure boot mechanism for linux-based embedded systems on fpgas. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017 12th International Symposium on*, pages 1–5. IEEE, 2017.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.

[RSA14] RSA Laboratories. PKCS11 cryptographic token interface standard, version 2.40, September 2014.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions, composed keys is np-complete. *Theor. Comput. Sci.*, 299(1-3) :451–475, 2003.

[RW10] R. Reddy and C. Wallace. Trust anchor management requirements. RFC 6024, RFC Editor, October 2010.

[Rya14] Mark Dermot Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[SAM+13] Stefan Santesson, Rich Ankney, Michael Myers, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.

[SCFY96] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2) :38–47, 1996.

[Sch95] Bruce Schneier. *Applied Cryptography (2nd Ed.) : Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[SDKD10] Samuel S. Shepard, Renren Dong, Ray Kresman, and Larry Dunning. Anonymous id assignment and opt-out. In Sio-Iong Ao and Len

Gelman, editors, *Electronic Engineering and Computing Technology*, pages 419–431, Dordrecht, 2010. Springer Netherlands.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. *Proc. of Symp. Math. Soc., 1971*, 20 :41–440, 1971.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11) :612–613, November 1979.

[Sky14] Thomas Skybakmoen. Next generation firewall comparative analysis. 2014.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE Computer Society, 2012.

[SP95] Malcolm Stevens and Michael Pope. Data diodes. Technical report, 1995.

[SST05] Katja Schmidt-Samoa and Tsuyoshi Takagi. Paillier's cryptosystem modulo p2q and its applications to trapdoor commitment schemes. In *International Conference on Cryptology in Malaysia*, pages 296–313. Springer, 2005.

[Tho12] Steven Thomason. Improving network security : next generation firewalls and advanced packet inspection devices. *Global Journal of Computer Science and Technology*, 2012.

[Tsa10] Rose Tsang. Cyberthreats, vulnerabilities and attacks on scada networks. 01 2010.

[Tur06] Mathieu Turuani. The CL-Atse Protocol Analyser. In Frank Pfenning, editor, *17th International Conference on Term Rewriting and Applications - RTA 2006 Lecture Notes in Computer Science*, volume 4098 of *LNCS*, pages 277–286, Seattle, USA, August 2006. Springer.

[UB13] DENX Das U-Boot. The universal boot loader. Technical report, 2013.

[VGP16] Jan Vcelak, Sharon Goldberg, and Dimitrios Papadopoulos. NSEC5, DNSSEC Authenticated Denial of Existence. Internet-Draft draft-vcelak-nsec5-03, Internet Engineering Task Force, September 2016. Work in Progress.

[WhSsH+08] I-Cheng Wang, Chih hao Shen, Tsan sheng Hsu, Churn-Chung Liao, Da-Wei Wang, and J. Zhan. Towards empirical aspects of secure scalar product. In *Information Security and Assurance, 2008. ISA 2008. International Conference on*, pages 573–578, April 2008.

[Wil14] Dick Wilkins. Uefi firmware security best practices. *UEFI Plugfest*, 2014.

[WL13] Wenye Wang and Zhuo Lu. Cyber security in the smart grid : Survey and challenges. *Computer Networks*, 57(5) :1344–1371, 2013.

[WSI02] Yodai Watanabe, Junji Shikata, and Hideki Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, pages 71–84, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Yao82] Andrew C. Yao. Protocols for secure computations. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0 :160–164, 1982.

[YCR16] Jiangshan Yu, Vincent Cheval, and Mark Ryan. DTKI : A new formalized PKI with verifiable trusted parties. *Comput. J.*, 59(11) :1695–1713, 2016.

[YTP07] Danfeng Yao, Roberto Tamassia, and Seth Proctor. Private distributed scalar product protocol with application to privacy-preserving computation of trust. In Sandro Etalle and Stephen Marsh, editors, *Trust Management : Proceedings of IFIPTM 2007 : Joint iTrust and PST Conferences on Privacy, Trust Management and Security, July 30– August 2, 2007, New Brunswick, Canada*, pages 1–16, Boston, MA, 2007. Springer US.

[Zim95] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

[ZM06] Kurt Zeilenga and Alexey Melnikov. Simple Authentication and Security Layer (SASL). RFC 4422, June 2006.

GLOSSAIRE

NGFW *Next Generation FireWall*
SCADA *Supervisory Control and Data Acquisition*
ICS *Industrial Control System*
IHM *Interface Homme-Machine*
NAT *Network Address Translation*
LAN *Local Area Network*
MAN *Metropolitan Area Network*
FTP *File Transfert Protocol*
OSI *Open Systems Interconnection*
TCP *Transmission Control Protocol*
SSL *Secure Socket Layer*
TLS *Transport Layer Security*
DMZ *DeMilitarized Zone*
HSM *Hardware Security Module*
IP *Internet Protocol*
ACC AC Client
ACS AC Serveur
ACI AC Intégrateur
ACE AC Employés
ACF AC Fabricant
PKI *Public Key Infrastructure*

PKIX *Public Key Infrastructure X.509*

PIN *Personal Identification Number*

CRL *Certificate Revocation Lists*

OCSP *Online Certificate Status Protocol*

PC *Politique de Certification*

DPC *Déclaration des Pratiques de Certification*

TPM *Trusted Platform Module*

FPGA *Field-Programmable Gate Array*

USB *Universal Serie Bus*

OS *Operating System*

TEE *Trusted Execution Environment*

MCU *MicroController Unit*

eMMC *embedded MultiMedia Card*

PKCS *Public Key Cryptography Standards*

SO *Security Officer*

FTPS *File Transfert Protocol Secure*

API *Application Programming Interface*

HTTP *HyperText Transfer Protocol*

HTTPS *HTTP Secure*

ARAMIS "Architecture Robuste pour les Automates et Matériels des Infrastructures Sensibles"

BPI Banque Publique d'Investissement

HMAC *keyed-Hash Message Authentication Code*

OPC-UA *OPC Unified Architecture*

SoC *System on Chip*

IDS *Intrusion Detection System*

UDP *User Datagram Protocol*

EN *Electronic Notary*

NE Notaire Electronique

LRA *Local Registration Authority*

CVL *Certificate Verification List*

CA *Certification Authority*

ILS *Integrity Log Server*
ARPKI *Attack Resilient Public-Key Infrastructure*
DTKI *Distributed Transparent Key Infrastructure*
IGC Infrastructure de Gestion de Clefs
MPC *Multi-Party Computation*
ASN.1 *Abstract Syntax Notation*
DER *Distinguished Encoding Rules*
TBS *To Be Signed*
MPWP *Multiple Private Keys Weighted Protocol*
P-MPWP *Paillier-MPWP*
AC Autorité de Certification
SN *Serial Number*
SASL *Simple Authentication and Security Layer*
nonce *Number used Once*
PGP *Pretty Good Privacy*
SIEM *Security Information and Event Management*
MITM *Man-In-The-Middle*
PLC *Programmable Logic Controller*
RFC *Request For Comments*
IETF *Internet Engineering Task Force*
CPA *Chosen Plaintext Attack*
DSDP *Distributed Secure Dot Protocol*
YTPSS *YTPwith Salary Sum*
IOT *Internet of Things*
AR *Authentication Request*

Résumé

Dans un monde de plus en plus connecté, la question de la confiance dans les systèmes d'information qui nous entourent devient primordiale, et amène naturellement à des interrogations quant à leur sécurité. Les enjeux de cette dernière concernent autant la confidentialité des données individuelles que la protection des architectures critiques, notamment déployées dans le domaine de l'énergie et du transport. Dans cette thèse, nous abordons trois problématiques liées aux architectures de sécurité des systèmes d'information. Tout d'abord, nous proposons une architecture pour un module de rupture protocolaire, fournissant une protection face aux attaques utilisant le réseau comme vecteur. Grâce à l'isolation et le filtrage des échanges qu'il réalise, nous montrons que ce nouvel équipement est particulièrement adapté à la sécurisation des systèmes de contrôle-commandes. Nous abordons ensuite le thème de la sécurité des utilisateurs finaux ou objets connectés, par la définition d'une Infrastructure de Gestion de Clefs (IGC) centrée sur ces derniers, dénommée LocalPKI. Elle repose sur l'utilisation de certificats auto-signés, et son objectif est d'allier la simplicité des IGC pair-à-pair avec la sécurité des IGC hiérarchiques. Enfin, nous nous intéressons à l'amélioration du mécanisme des ancres de confiance pour les autorités de certification, utilisé par exemple dans PKIX et LocalPKI. A cet égard, nous commençons par définir des protocoles multi-parties permettant de calculer des produits scalaires et matriciels, préservant la confidentialité des données. Nous montrons finalement comment les appliquer dans le cadre de l'agrégation de confiance, et par conséquent à la réputation des autorités de certification.

Abstract

In a increasingly connected world, trust in information systems is essential. Thus, many questions about their security arise. Topics of these questions include individual data confidentiality as well as protection of Industrial Critical Systems (ICS). For instance, ICS are deployed in sectors including energy or transportation where security is of high importance. In this thesis, we address three problems related to the security architecture of information systems. We first propose an architecture for a protocol splitting device. This provides protection against network attacks by isolating and filtering data exchanges. We show that this new security equipment is well suited for ICS. Then, we focus on end-user security. We define a user-centric Public Key Infrastructure (PKI) called LocalPKI. By using self-signed certificates, this infrastructure combines the user-friendliness of PGP-based PKI and the security of hierarchical PKI. Finally, we improve the trust anchor mechanism. It is employed by Certification Authorities (CA) and especially used in PKIX or LocalPKI. In that respect, we first define multi-party protocols to securely compute dot and matrix products. Then, we explain how to apply them on trust aggregations and thus on the reputation of certification authorities.