AIX-MARSEILLE UNIVERSITÉ
ED 184 - MATHÉMATIQUES ET INFORMATIQUE
I2M-ERISCS, UMR 7373

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : Informatique

Gabriel RISTERUCCI

Mécanismes et outils pour sécurisation de systèmes à accès distants
Application aux systèmes de gestion électronique de documents

Soutenue le 31/03/2016 devant le jury :

Jean-Louis LANET	Professeur des Universités, INRIA, Rennes	Rapporteur
Michel RIVEILL	Professeur des Universités, UNSA, Nice	Rapporteur
Rémi MORIN	Professeur des Universités, AMU	Examinateur
Léon MUGWANEZA	Maître de Conférence, AMU	Examinateur
Liam MURPHY	Professor, UCD, Dublin	Examinateur
Henri NOAT	SESIN SA	Invité
Traian MUNTEAN	Professeur des Universités, AMU	Directeur de thèse

RÉSUMÉ

Cette thèse a pour objet l'amélioration de la sécurité de systèmes à accès distant par l'utilisation d'outils cryptographiques. Elle s'applique en particulier aux applications de gestion de documents numériques pour leurs problématiques de communication, d'authentification et de gestion de droits. Contrairement aux approches classiques consistant à utiliser des moyens de protections ponctuels, nous proposons ici un ensemble d'outils conçu pour collaborer afin de renforcer la sécurité du système. La sécurisation des communications est réalisée grâce à la conception d'un protocole de communications sécurisée adapté aux applications distribuées. Les problématiques d'authentification ont donné lieu à l'élaboration de solutions permettant d'apporter un support cryptographique pour toutes modalités d'authentification. La gestion des droits fait l'objet d'un développement spécifique permettant d'associer des droits à des applications cryptographiques. Un point clé de ces réflexions est l'importance de l'accessibilité de ces outils de sécurité pour les utilisateurs du système. Cela a influé sur les propositions pour qu'elles perturbent le moins possible l'expérience utilisateur. Le résultat est l'intégration en un système global de différents outils et mécanismes apportant une sécurité complète à un système de gestion de documents numériques. Cette sécurité est basée sur des algorithmes cryptographiques afin de disposer de propriétés de sécurité prouvables et vérifiables. Comme support de ces mécanismes, une plateforme de sécurité logicielle a été conçu pour fournir les outils cryptographiques de façon portable.

Mots clés : sécurité, authentification, communication, contrôle d'accès, documents électroniques, cryptographie

ABSTRACT

This thesis' goal is the improvement of the security of remotely accessed systems with the use of cryptographic tools. Specifically it is applied to digital documents management software that raise issues in three fields : communication, authentication and rights management. Unlike common approaches that involve the use of individual protections for these three fields, we offer a set of tools made to work together to improve the system's security. Securing communication is done thanks to a new secure communication protocol designed for distributed applications. Authentication issues led to the development of two tailored solutions providing cryptographic support to the application for any authentication method. Rights management is handled through new associations between a given access right and specific cryptographic applications. A key element of those solutions is the emphasis put on the usability of these secure tools. It swayed the development of our proposals toward more transparent solutions that would not disturb the user experience. As a result, we obtained a secure system made of these tools and mechanisms that work together to provide full and transparent security for a digital documents management software. This security is fully based on cryptographic algorithms to provide provable and verifiable security properties. As a supporting layer for these mechanisms, a secure software library was designed to provide all the required tools for cryptographic uses in a portable way.

Keywords : security, authentication, communication, access control, digital documents, cryptography

REMERCIEMENTS

Je remercie tout d'abord mon directeur de thèse pour m'avoir offert l'opportunité de réaliser les travaux présentés ici. Je remercie également les rapporteurs pour le temps qu'ils m'ont accordé et leurs remarques sur le contenu de ce manuscrit. Enfin, pour le temps qu'il a accordé à la lecture et aux corrections de cette thèse je remercie M. Mugwaneza.

Pour les discussions sur des sujets variés qui m'ont permis au fil des années soit de construire mon raisonnement, soit plus simplement d'apporter une bonne ambiance de travail, je remercie (sans ordre particulier) Robert Rolland, Alexis Bonnecaze et Nicolas Baudru. Trop nombreux pour les citer ici, je dois aussi remercier les participants et les organisateurs des diverses conférences et séminaires auxquels j'ai pu participer, notamment Crypto'Puces et CAI. Les échanges réalisés à ces occasions m'ont permis d'avoir une vision globale sur de nombreuses problématiques de sécurité et sans cela la direction prises par mes recherches aurait été très différente. Je remercie aussi mes camarades, Laurent Vallet, Mila Tukumuli, Kévin Atighehchi, Rick Ghanem et Alexandre Antunes pour les nombreuses discussions que nous avons pu avoir au fil des années et pour leurs contributions à mes travaux, qui sont les fondations grâce auxquelles j'ai pu obtenir ces résultats.

Au-delà des collaborations directes ou indirectes à mes travaux, je tiens à remercier les membres des différentes structures de l'université pour leur assistance et leur compréhension vis-à-vis de certaines situations administratives complexes, en particulier la scolarité de l'université et l'école doctorale.

Il ne m'aurait pas été possible de terminer cette thèse sans le financement de la société SESIN. Je remercie donc M. Noat pour m'avoir permis de conclure mes travaux ; au-delà du financement, les problématiques soulevées lors de différentes collaborations avec cette société m'ont servies de base de réflexion et ont guidés les choix techniques abordés dans ce manuscrit.

Sur un plan plus personnel, je remercie chaleureusement toute ma famille pour leur soutient inconditionnel tout au long de mes études. Qu'il s'agisse d'un soutien moral ou matériel, rien n'aurait été possible sans eux. La même remarque vaut pour mes amis, proches ou non, qui ont parfois servis de soupape de sécurité mais qui (à ma connaissance) ne m'en ont pas tenu rigueur.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

AAD Additional Authentication Data, données d'authentification additionnelles non confidentielles

AES Advanced Encryption Standard, algorithme de chiffrement par bloc le plus utilisé de nos jours

ANSSI Agence Nationale de la Sécurité des Systèmes d'Information, chargée de proposer les règles à appliquer pour la protection des systèmes d'information de l'État et de vérifier l'application des mesures adoptées

API Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

ASN1 Abstract Syntax Notation One, syntaxe utilisée pour décrire la structure d'éléments pour les communications dans des réseaux informatiques

CBC Cipher Block Chaining, enchaînement de blocs chiffrés : mode d'utilisation consistant à faire intervenir pour chaque bloc chiffré l'état du bloc précédent, afin de construire une chaîne

CGI Common Gateway Interface, un standard permettant d'exécuter des programmes variés pour répondre à des requêtes web

CLMUL Carry-Less Multiplication, une extension au jeu d'instruction x86 permettant d'accélerer les multiplications sur des corps finis

CPS Carte de Professionel de Santé, carte à puce permettant d'authentifier les professionels de santé

CRL Certificate Revokation List, liste de certificats numériques révoqués par leur émetteur

CTR Compteur, cet acronyme est utilisé pour identifier le mode compteur, un mode de chiffrement par blocs

DES Digital Encryption standard, ancien algorithme de chiffrement par blocs

DSA Digital Signature Algorithm, standard de signature numériques

DoS Denial of Service, déni de service : empêcher le fonctionnement normal d'un service en en bloquant l'accès ou les processus

DTLS Datagram Transport Layer Security, une version de TLS fonctionnant sur des réseaux en mode datagramme tels qu'UDP

EAP Extensible Authentication Protocol, framework décrivant les éléments nécessaires à la réalisation d'une authentification

ECDH Elliptic-Curve Diffie-Hellman, Diffie-Hellman basé sur courbes elliptiques

ECDSA Elliptic-Curve Digital Signature Algorithm, signatures numériques basées sur courbes elliptiques

EVP Enveloppe, interface haut-niveau de la bibliothèque OpenSSL

FPGA Field-programmable gate array, réseau reprogrammable de portes logiques permettant d'implémenter des fonctions à la demande

GCM Galois/Counter-Mode, mode de chiffrement par blocs intégrant un mécanisme d'authentification des données

GMP GNU Multiple Precision Arithmetic Library, bibliothèque de calcul sur des grands nombres

HMAC Hash-based Message Authentication Code, code d'authentification de message basé sur des fonction de hachage

HTML Hypertext Markup Language, le langage de balises utilisé pour la construction de page web

HTTP Hypertext Transfer Protocol, protocole utilisé pour le transfert de pages Web

HTTPS Hypertext Transfer Protocol Secure, version de HTTP sécurisée par TLS

IDL Interface Description Language, syntaxe permettant de décrire des interfaces de haut-niveau souvent utilisé pour simplifier les communications inter-processus.

IPSec Internet Protocol Security, protocole de chiffrement des échanges au niveau IP

JCE Java Cryptographic Extension, implémentation d'outils cryptographiques du langage Java

JNA Java Native Access, bibliothèque Java permettant de simplifier les appels au code natif

JNI Java Native Interface, composant du langage Java permettant d'appeler du code natif

LTE Long Term Evolution, norme évolutive des réseaux de communications mobiles

MAC Message Authentication Code, code d'authentification de message permettant d'en vérifier l'intégrité

MD5 Message Digest 5, fonction de hachage simple

MitM Man in the Middle, l'homme au milieu : intercepter et/ou modifier les communications entre deux systèmes

MTU Maximum Transmission Unit, taille maximum d'un paquet transmis sur un réseau donné

NIST National Institute of Standards and Technology, institut Américain faisant référence notamment dans le domaine des algorithmes cryptographiques

NPAPI Netscape Plugin API, interface d'extensions pour navigateurs web

OCSP Online Certificate Status Protocol, mécanisme de contrôle de la révocation de certificats numériques remplaçant les CRL

OTP One-Time Password, mot de passe à usage unique généralement limité dans le temps

OTR Off-the-record, confidentiel

PHP PHP : Hypertext Preprocessor, un langage de script souvent utilisé pour la réalisation de sites web

PIN Personal Identification Number, code permettant entre autre de déverouiller un jeton cryptographique matériel

PKCS Public-key cryptography standard, ensemble de standard décrivant des éléments communs pour l'application d'outils cryptographiques

PKCS#3 La spécification #3 définit le standard déchange de clés Diffie-Hellman

PKCS#7 La spécification #7 définit un standard d'encapsulation de message chiffré

PKCS#8 La spécification #8 définit un format de stockage de paire de clés

PKCS#11 La spécification #11 définit une API générique pour l'utilisation de périphériques cryptographiques

PKCS#12 La spécification #12 définit un format de fichier utilisé pour stocker des clés privées

PKI Public Key Infrastructure, infrastructure à clé publique : ensemble de systèmes permettant la gestion de certificats numériques de clé publiques

PRNG Pseudo-Random Number Generator, générateur pseudo-aléatoire

RC2 Rivest Cipher 2, algorithme de chiffrement par bloc

RC4 Rivest Cipher 4, algorithme de chiffrement à flot

RSA Rivest Shamir Adleman, algorithme de chiffrement a clé publique parmis les plus utilisés au monde

S/MIME Secure/Multipurpose Internet Mail Extensions, enrobage sécurisé notamment utilisé pour les pièces jointes de messagerie éléctronique

SGED Système de Gestion Électronique de documents, permettant la création, manipulation, et distribution de documents de nature variée

SHA1 Secure Hash Algorithm 1, fonction de hachage

SHA2 Secure Hash Algorithm 2, fonction de hachage succédant à SHA1, plus sûr que ce dernier

SIM Subscriber Identity Module, les cartes SIM permettent d'identifier un abonné à un service

SSH Secure Shell, protocole de communication sécurisée permettant notamment l'ouverture d'une invite de commande sur un système distant

SSL Secure Socket Layer, protocole de communication sécurisée remplacé par TLS

SSO Single Sign-On, système permettant de s'identifier une seule fois pour l'utilisation de plusieurs systèmes

SVC Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

TCP Transmission Control Protocol, protocole de communications garantissant notamment l'ordre et l'unicité de la délivrance des messages

TCP/IP TCP over IP, suite de protocoles utilisant TCP sur un réseau IP

TLS Transport Layer Security, protocole de communication sécurisée remplaçant SSL

USB Universal Serial Bus, Bus de communication série répandu dont un usage courant est l'utilisation avec des périphériques de stockage amovibles

UDP User Datagram Protocol, protocole de communications rapide n'ayant ni garantie de délivrance des messages, ni de garantie d'ordre d'arrivé

VoIP Voice over IP, voix sur IP, communications audio circulant sur des réseaux informatiques basés sur le protocole IP

VPN Virtual Private Network, réseau privé virtuel, réseau logique pouvant regrouper des machines situées sur différents réseaux physiques distants

ZRTP Z-Real-time Transport Protocol, protocole de protections des communications VoIP

INTRODUCTION

Depuis quelques années l'informatique a pris une place majeure dans un grand nombre d'activités. La démocratisation des accès à Internet et l'omniprésence de périphériques "intelligents" a favorisé l'intégration des systèmes communicants dans la plupart des processus de traitement. Ces nouveaux usages sont venus avec un problème majeur qui est la protection des informations numériques. L'ensemble des données, que ce soit des fichiers échangés entre utilisateurs, des dossiers professionnels, du contenu multimédia ou des données personnelles transitent par de nombreux réseaux tous inter-connectés. Ces échanges nécessitent d'être protégé de différentes menaces. Le vol de données confidentielles, l'altération des informations et l'usurpation d'identité sont autant de problèmes qui se transcrivent du monde "physique" vers le monde numérique. Si des outils existent en effet pour permettre de sécuriser de façon adéquate un grand nombre d'usages des systèmes informatiques communicants, la pratique montre malheureusement une différence importante entre les possibilités existantes et la protection effective des données, tant au niveau individuel qu'au niveau professionnel. L'objectif est ici l'application pratique d'outils de sécurités cryptographiques fiables avec une importance particulière accordée aux contextes d'utilisations réelles, afin de permettre la démocratisation des usages sécurisés de la même façon que s'est démocratisée l'utilisation des systèmes communicants.

Les données numériques sont un enjeu majeur de la société moderne ; qu'il s'agisse d'information concernant la vie privée d'individus ou les données d'entreprises, il y a une forte convergence vers le "tout numérique". Des faits-divers récents ont montré que ces données n'étaient pas toujours protégées de façon adéquate. Le problème est ici multiple : plus il existe de données numériques, plus leur utilisation peut être efficace. L'agrégation, l'indexation et l'accès rapide à un grand volume de données permet par exemple d'alimenter des systèmes de prévisions et d'optimiser des aspects qui n'étaient pas envisagés avec les solutions "classiques". Un exemple trivial est la possibilité de déterminer, dans une entreprise commerciale, des tendances de consommations en se basant sur les précédents exercices. Les entreprises trouvent régulièrement de nouveaux moyens pour pouvoir exploiter les données et ce depuis qu'elles sont rendues accessibles. Ces approches sont ca-

taloguées sous l'appellation "Big Data" ; mais plus on injecte de données dans ces systèmes pour en améliorer l'efficacité, plus elles deviennent attractives. En effet, du point de vue d'un attaquant, l'agrégation d'un important volume de données représente autant un point positif pour leur exploitant qu'un moyen d'acquérir facilement de l'information.

Il est possible de classer les données en trois états en fonction de leur usage : les données au repos, les données en mouvement et les données en utilisation. Les données au repos (en anglais, "Data at Rest") représentent le moment où elles sont conservées sur des supports de stockage fixes et ce quel que soit l'emplacement. Cela inclue la conservation des données sur les appareils des utilisateurs et leur stockage sur des serveurs. La sécurité de ces données dépend principalement de deux points : leur accès physique et leur accès via une intrusion numérique. L'accès physique est évident ; il s'agit de pouvoir manipuler ou subtiliser le support de stockage. Les intrusions numériques consistent à obtenir illégitimement l'accès à un système informatique afin de pouvoir récupérer les données qu'il contient.

Les données en mouvement (en anglais, "Data in Motion") sont les données lorsqu'elles sont échangées entre deux systèmes. Ces échanges se produisent principalement sur des réseaux informatiques et peuvent parfois transiter par des réseaux publics tels qu'Internet pour rejoindre leur destination. Le transfert de ces données les rends plus vulnérables, car l'utilisation de réseaux peu ou pas sécurisés entraîne un risque important pour la confidentialité et l'intégrité des échanges.

Enfin, l'état des données en utilisation (en anglais, "Data in Use") concerne les informations activement manipulées, que ce soit par un utilisateur ou par un système automatisé de traitement de l'information. La nature du risque pesant sur ces données est différente du risque existant pour les données au repos et les données en mouvement. Lors de la manipulation d'informations, ces dernières doivent être rendues accessibles aux applications quelles que soient les protections mises en place par ailleurs. Il faut alors que les applications elles-mêmes soient sécurisées afin d'éviter les fuites d'informations involontaires et les vulnérabilités techniques.

Les différentes approches permettant de sécuriser les données dans ces trois états peuvent se classer en deux grandes catégories : les systèmes de contrôles d'accès non cryptographiques et les systèmes cryptographiques. Dans le premier cas, l'approche de sécurité consiste à assurer que l'accès à une donnée ne peut se faire que par un chemin contrôlé et à placer des points de vérifications comme une gestion des droits d'accès pour les utilisateurs du système. Cette solution est très courante et donne l'impression d'un contrôle totale de la diffusion de l'information. En pratique cependant, les attaquants rivalisent d'imaginations et peuvent exploiter des failles à plusieurs niveaux, parfois même à la périphérie du système visé, afin de contourner le "chemin contrôlé" et obtenir un accès direct aux

données.

En opposition à ces approches de sécurité "évitables" il existe une solution "incontournable" : l'utilisation de la cryptographie. Cette dernière permet de protéger les données directement dans leur forme plutôt que de les placer derrière une barrière de sécurité. Les approches cryptographiques permettent de manipuler l'information pour la rendre indéchiffrable via l'utilisation d'éléments secrets. Ces éléments secrets sont idéalement connus par les seuls utilisateurs autorisés à accéder aux informations ainsi protégées. Pour ces systèmes, l'utilisation d'une approche de "gestion des droits" reste possible, mais en cas de contournement du système par un attaquant, les informations extraites sont inexploitables. Afin de réaliser cette sécurité incontournable, des techniques de chiffrement et de gestion de clés cryptographiques sont nécessaires. Leur résultat est une protection liée directement aux données et non plus lié à l'environnement permettant de les manipuler.

Le support choisi ici pour illustrer l'usage de systèmes communicants et les solutions pour y apporter la sécurité est un SGED [1] générique reprenant les principes de fonctionnement des solutions courantes. Un tel système présenté de façon générique permettra par la suite d'appliquer les résultats à des systèmes plus spécifiques. La finalité d'un SGED est la manipulation de documents numériques ; il permet leur création, leur stockage, leur partage et leur indexation. Chacune de ces fonctionnalités peut être limitée par un système de permissions afin de restreindre l'accès à certains contenus en fonction des droits des utilisateurs.

Un SGED moderne est un cas complet d'application distante et de besoins de sécurités. Qu'il s'agisse de systèmes classiques client-serveur, multi-serveur ou distribué, ces systèmes font intervenir un grand nombre de communications réseau. Les besoins de sécurités d'un SGED sont multiples ; en plus des communications, il est nécessaire de gérer l'authentification des utilisateurs ainsi que leurs droits d'accès et le stockage sécurisé des contenus. De plus, le type d'approche de sécurité est important. La problématique d'authentification des utilisateurs est elle-même complexe, car elle inclut une contrainte de fiabilité avec des modalités d'authentifications fortes ainsi qu'une contrainte de mobilité. L'association de ces deux contraintes pose un défi intéressant, car les solutions usuelles présentent généralement une incompatibilité sur ce point.

L'approche la plus courante concernant la gestion des droits des utilisateurs pour des SGED distants ou distribués est l'utilisation d'un système basé sur les contrôles d'accès. C'est sur cette base que l'exemple de SGED générique utilisé ici est conçu afin de permettre une transition vers un système utilisant une sécurité "incontournable" par la suite. Cette transition permet d'apporter une sécurité cor-

1. Système de Gestion Électronique de documents, permettant la création, manipulation, et distribution de documents de nature variée

recte en impactant au minimum les contraintes d'utilisations. En effet, l'un des problèmes majeurs dans les applications sécurisées est la rigidité des politiques de sécurités, poussant souvent les utilisateurs à contourner le système, voir à l'ignorer complètement en faveur d'autres approches moins contraignantes, mais moins sécurisées.

Le point final sur la sécurité d'un SGED concerne le stockage des documents. Ce dernier est couramment basé sur la sécurité distante ; considérant que les données stockées sur un ou plusieurs serveurs est sécurisée. Malheureusement, à partir du moment où les documents sont visibles par le serveur, toute solution de protection locale présente la même faiblesse : les outils permettant d'accéder aux données sont présents au sein de l'infrastructure distante, qu'il s'agisse d'une même machine ou de plusieurs se répartissant les tâches de stockage et de sécurisation. Cette infrastructure distante peut être vue comme une seule entité dont la compromission partielle est équivalente à la compromission globale. Une solution ici est de réduire au maximum la confiance accordée aux intermédiaires situés entre deux utilisateurs devant manipuler le même document, en particulier la confiance accordée à l'infrastructure du SGED. Pour cela, il est nécessaire de mettre en place des moyens de protections en amont du service, directement sur le poste client.

L'ensemble des points de changements évoqués ici peuvent être regroupés sous un même principe : la transformation progressive d'un système initial dont la sécurité dépend de contrôles d'accès vers un système disposant de la même accessibilité, mais intégrant la sécurité au sein même des données manipulées. Ces évolutions sont basées sur des approches cryptographiques, seules solutions à même de fournir des outils de protections incontournables. Les outils cryptographiques permettent de réaliser des opérations variées sur les données afin de garantir certaines propriétés, dont la confidentialité et l'authenticité. Ces opérations dépendent d'éléments secrets, des clés cryptographiques, qui garantissent la protection fournie par les mécanismes cryptographiques. Elles doivent être conservées de façon sécurisée et leur communication doit faire l'objet de précautions importantes.

Parmi les outils cryptographiques se trouvent des méthodes permettant d'appuyer des mécanismes d'authentification forte. Certains mécanismes d'authentification reposent sur l'utilisation de procédés cryptographiques pour fonctionner, en particulier des mécanismes de type défi-réponse qui sont très fiables lorsqu'ils sont supportés par des outils cryptographiques. En effet, l'utilisation d'algorithmes de chiffrement symétriques ou l'utilisation d'algorithmes de signatures numériques permettent de prouver que l'utilisateur connaît et/ou dispose bien du secret qui lui est associé de façon irréfutable.

Les outils cryptographiques permettent aussi d'apporter des garanties de confidentialités. Les algorithmes de chiffrement permettent de transformer un élément de façon à le rendre incompréhensible en l'absence de connaissance de la clé se-

crète utilisée. Cette clé secrète devient alors la donnée sensible à protéger. La plupart des opérations de chiffrement utilisées à cette fin de confidentialité sont des opérations symétriques pour lesquelles la possibilité de chiffrement n'est pas distincte de la possibilité de déchiffrement. Cela pose le problème de la distribution des clés secrètes. Par nature ces dernières doivent être communiquées pour permettre le partage de l'information confidentielle ; cependant ces communications représentent un maillon faible dans le système. Cette gestion des clés pose des problèmes multiples : qui est responsable de leur création, comment sont-elles transmises et où sont-elles stockées.

Pour pouvoir répondre à certaines de ces questions, il faut faire la distinction entre cryptographie réalisée par le serveur et cryptographie réalisée par le client. Dans le premier cas, le serveur doit connaître les clés utilisées pour réaliser les opérations de chiffrement. La confidentialité ainsi obtenue n'exclue pas le serveur de la connaissance du contenu des fichiers. En cas d'acte de malveillance par ce dernier, ou en cas de faille de sécurité menant à la divulgation des informations connues par le serveur, les données qu'il contient sont compromises. Dans le cas de la réalisation des opérations cryptographiques par le client, la confidentialité obtenue exclue le serveur. Pour cela, la gestion des clés utilisée doit prendre en compte des mécanismes de partage compatibles avec la cryptographie côté client ; en particulier elle doit permettre la transmission des clés entre clients et conserver ces clés hors de la connaissance du serveur.

Le dernier point pour lequel la cryptographie présente une importance majeure est la protection des communications. Les algorithmes de chiffrement, d'authentification et de contrôle d'intégrités cryptographiques permettent d'établir des échanges de données sécurisés entre plusieurs entités. Une connexion sécurisée protège également des altérations qu'elles soient malveillantes ou non. Il existe de nombreux protocoles de communications sécurisés, qui font l'objet d'une section détaillée dans la suite de ce document.

Le travail présenté dans ce manuscrit a pour objectif l'utilisation de solutions accessibles et sécurisées pour la manipulation des données numériques. L'accessibilité regroupe les contraintes d'utilisation des solutions proposées. Réduire au maximum les contraintes et tirer profit des limites initialement présentes dans les systèmes existants est donc un thème qui sera récurrent dans ce document. La sécurisation des données se fait à partir de deux éléments : l'application méthodique d'outils cryptographiques permettant de répondre aux différents usages et la coopération entre ces outils. La finalité est d'obtenir un système sécurisé par des méthodes incontournables en imposant le moins de restrictions possibles sur son utilisation.

Afin de parvenir à ce résultat, il est nécessaire de concevoir des outils adaptés à différentes problématiques. Ces outils sont tous basés sur des procédés cryp-

tographiques afin de rendre impossible les contournements. Les différents points considérés font écho aux problématiques décrites : la protection des communications, la protection des données, les moyens d'authentification des utilisateurs et la coopération entre ces composants.

La protection des communications n'est pas un problème nouveau ; il existe de nombreuses solutions permettant de le résoudre. Ces solutions partagent un point commun : l'interopérabilité entre différents systèmes. Il existe une part non négligeable de communications numériques n'ayant pas cette restriction, notamment les applications distribuées pour lesquelles des communications internes ont lieu entre différentes instances identiques. Ces communications nécessitent un moyen de protection adapté ; afin de répondre à ce problème un protocole de communication sécurisée spécifique pour ce type d'usage a été conçu.

La protection des données est, comme pour la protection des communications, un domaine déjà exploré. Le véritable problème dans la protection des données varie en fonction du type de données que l'on considère. Pour les données en utilisation, la protection dépend principalement du système d'exploitation, alors que pour les données en mouvement c'est l'utilisation de protocoles de communication sécurisée qui est utilisée. Il reste le cas des données au repos. Les solutions sont généralement basées sur l'utilisation d'algorithmes de chiffrement appliqué aux données. Le problème peut se réduire à la gestion correcte des clés cryptographiques utilisées pour chiffrer les données. Une réponse à ce problème est apportée sous la forme d'un mécanisme de transfert d'un trousseau de clés sécurisé.

L'authentification des utilisateurs mélange des solutions cryptographiques et non cryptographiques. Cependant, afin de fournir une authentification forte un support cryptographique correct est nécessaire. C'est pour cela qu'un protocole d'authentification générique est proposé ; il permet d'intégrer un procédé de signatures numériques à tous les moyens d'authentification, y compris ceux n'étant pas basé sur des solutions cryptographiques. Cette approche permet d'exécuter un protocole d'authentification distant, transférant les informations d'authentification à un serveur dédié en garantissant la fiabilité de cet échange et du résultat obtenu.

Afin de créer un système cohérent, il est nécessaire d'avoir une coopération importante entre ses différents composants. Dans le cas d'un système sécurisé par des moyens cryptographiques, cette coopération prend principalement la forme d'un mécanisme de gestion de clés. Ce mécanisme est partagé par chaque composant et est basé sur le processus d'authentification. Ce processus d'authentification a deux finalités : l'authentification de l'utilisateur vis-à-vis du serveur et la récupération des clés cryptographiques sur le poste client. Ces clés, présentées sous la forme d'un trousseau de clés sécurisé, ne sont rendues accessibles qu'en présence d'informations spécifiques que l'utilisateur a préalablement fournis.

En plus de la coopération entre composants cryptographiques, il y a égale-

ment l'aspect de la coopération entre le système cryptographique et le support, ici un SGED. Ce dernier présente des fonctionnalités particulières permettant la gestion de documents, en particulier le partage de ces documents et leur stockage à distance sur le serveur. L'intégration de fonctions de sécurités se fait de façon à tirer profit du support applicatif existant. Le mécanisme d'authentification est utilisé en remplacement du système initialement présent dans le SGED ; il permet toujours d'authentifier l'utilisateur vis-à-vis du service, mais permet également la récupération d'un trousseau de clés sécurisé. Cette récupération a lieu en utilisant le système de transfert de documents du SGED, considérant ce trousseau de clés comme un document quelconque, ne nécessitant donc pas de modifier le comportement du serveur.

Au-delà des sujets traités tout au long de cette thèse, des travaux en lien avec la problématique centrale ont été réalisés et ont donné lieu à la rédaction des articles suivants : *An efficient parallel algorithm for skein hash functions* (implémentation et analyse des performances d'un nouvel algorithme de fonction de hachage dans le cadre du développement d'outils de sécurité)[17], *A new secure virtual connector approach for communication within large distributed systems* (conception d'un protocole de communication sécurisée pour des cas d'usages spécifiques, décrit au chapitre 1)[58] et *New models for efficient authenticated dictionaries* (nouvelle structure de données permettant l'authentification de grandes quantités d'information de façon efficace)[5].

Ce manuscrit est découpé en quatre chapitres, chacun portant sur un aspect particulier lié à la sécurisation des systèmes à accès distants. Le premier chapitre porte sur l'établissement de communication sécurisée. Il commence par une introduction aux propriétés de sécurité attendue lors de l'établissement de connexions sécurisées. Cette introduction est suivie d'une présentation des solutions existantes, ainsi que des contraintes qui ont été considérées pour la conception d'un protocole spécifique nommé SVC [2]. Ce protocole est ensuite présenté en détail. Ce chapitre se conclut sur une évaluation de performances et de sécurité du protocole, ainsi que sur des perspectives d'évolutions et des recommandations générales sur l'établissement de communication sécurisée.

Le chapitre suivant est dédié aux moyens d'authentifications. Il débute par une présentation des systèmes d'authentification ainsi que des différentes modalités d'authentification existantes. Ensuite est présenté le problème spécifique de l'utilisation de l'authentification comme fournisseur d'éléments cryptographiques pour une utilisation par ailleurs ; une solution à ce problème est proposée sous la forme d'un système de gestion de clés sécurisés. Enfin, un protocole d'authentification complet est présenté, associé à la description d'un prototype d'implémentation.

2. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

L'objet du chapitre trois est l'intégration de fonctionnalités de sécurité dans un SGED. Tout d'abord, un modèle complet de SGED générique non sécurisé est décrit. Les parties suivantes présentent l'ajout d'outils sécurisés sur différents points de fonctionnement du système : authentification, communications et stockage sécurisé. Une description technique conclue ce chapitre, présentant les points clés des solutions proposées.

Pour finir, le quatrième et dernier chapitre présente une plate-forme logicielle de sécurité. Après une introduction sur les solutions existantes et leurs limites, la plate-forme Arcana est présentée. Cette présentation est découpée en plusieurs sous-parties : origines et objectifs du projet, composants principaux, architecture et croissance. Pour finir sont évoqués la compatibilité avec les standards du domaine et les perspectives de développement.

1 PROTECTION DES COMMUNICATIONS

Il y a plusieurs façons de définir une communication sécurisée. Ces définitions s'appuient sur un ensemble de propriétés de sécurité que l'on peut attendre d'une communication. Les principales propriétés de sécurité sont les suivantes : confidentialité, intégrité, authenticité, le déni plausible, la confidentialité persistante et la protection de l'identité lors des communications. La confidentialité permet d'éviter qu'un tiers non autorisé n'ait accès au contenu de la communication. L'intégrité assure que les messages échangés ne sont pas modifiés durant leur transfert. L'authenticité recouvre en partie l'intégrité et apporte en plus des garanties sur l'identité de l'émetteur d'un message. Le déni plausible donne essentiellement la possibilité à l'émetteur d'un message de nier en être l'auteur. La confidentialité persistante permet de garantir que même en cas de divulgation ultérieure des informations d'un utilisateur (clé cryptographique, certificats,...) il reste impossible de déchiffrer une communication passée. Il s'agit là d'une propriété de sécurité différente de la confidentialité, car elle concerne la protection a posteriori des messages échangées. La protection de l'identité est une forme d'anonymat. Elle peut recouvrir plusieurs notions. Nous parlons ici principalement de masquage d'identité, afin de ne pas divulguer d'informations sur l'origine de la communication sécurisée. Nous pouvons donc avoir plusieurs définitions de communication sécurisée, par exemple en considérant uniquement la confidentialité, ou en considérant à la fois confidentialité et authenticité.

Chacune des propriétés de sécurité induit un coût parfois important sur la mise en œuvre de communication sécurisée. En particulier, l'authenticité des communications nécessite un système de gestion de clés des utilisateurs. La gestion de clés peut être particulièrement complexe si l'on considère un grand nombre d'utilisateurs, ou un réseau de grande taille sur lequel il est difficile d'utiliser des canaux secondaires pour partager les clés. Ainsi, selon le type de réseau et d'utilisation, certaines propriétés de sécurité peuvent se voire exclues pour des raisons pratiques ou de performances. Nous considérons ici qu'une communication sécurisée nécessite l'application de toutes les propriétés de sécurité sans exception.

Il existe plusieurs façons de mettre en place un système utilisant des communi-

cations sécurisées. Les deux approches courantes sont l'utilisation d'un protocole de communication sécurisée et la mise en place d'un VPN [1]. L'utilisation directe d'un protocole de communication sécurisée permet d'établir, au niveau de l'application, une connexion sécurisée avec une autre entité (une autre instance de l'application, un serveur,...). Cette approche permet un contrôle fin au niveau de l'application du niveau de sécurité que l'on souhaite obtenir et facilite la maintenance des éléments de sécurité de l'application. En effet, en intégrant ces éléments de sécurité au niveau applicatif, il est possible de les maintenir à jour même lorsque les éléments extérieurs (bibliothèques de sécurité, fonctions du système d'exploitation,...) ne sont plus mis à jour.

La mise en place d'un VPN présente des caractéristiques différentes de l'utilisation directe d'un protocole de communication sécurisée mais fournit une sécurité similaire. En effet, un VPN peut se présenter de différentes façons, mais les communications effectives sont réalisées, dans le cas d'un VPN sécurisé, via un protocole de communication sécurisée "classique" par l'application VPN. La principale différence entre les deux approches est qu'un VPN peut permettre d'apporter des communications sécurisées à des applications n'ayant pas nativement cette possibilité. Comme un VPN est vu par les applications comme une interface réseau, n'importe quelle application communiquant sur un protocole supporté par la solution de VPN (typiquement du TCP [2] ou de l'UDP [3]) bénéficie de la sécurité fournie. Un point négatif de l'utilisation d'un VPN est qu'il n'est pas sous le contrôle des applications qui l'utilisent et peut donc présenter un niveau de sécurité inadéquat en fonction de certains usages. En particulier, une application n'a pas le moyen de savoir si un VPN est utilisé, car cet aspect est masqué par le système d'exploitation, ni le type de sécurité qu'il fournit le cas échéant ; certains VPN sont utilisés uniquement pour leur propriété de jonction de réseaux distants sans aucun élément de sécurité. Il est également plus complexe de mettre en œuvre une solution de type VPN, car elle nécessite l'installation et la configuration d'éléments tiers à une application. Cet aspect n'est pas en soit un problème de sécurité, mais apporte une contrainte de déploiement qui rend cette solution incompatible avec certaines classes d'applications. Un cas de situation incompatible est l'utilisation d'une application nomade, lancée sur des systèmes pour lesquelles l'utilisateur n'a pas accès à la configuration du système d'exploitation, rendant impossible la mise en place d'un VPN.

À partir de ces éléments, nous pouvons voir qu'il existe des contraintes à la

1. Virtual Private Network, réseau privé virtuel, réseau logique pouvant regrouper des machines situées sur différents réseaux physiques distants

2. Transmission Control Protocol, protocole de communications garantissant notamment l'ordre et l'unicité de la délivrance des messages

3. User Datagram Protocol, protocole de communications rapide n'ayant ni garantie de délivrance des messages, ni de garantie d'ordre d'arrivé

mise en œuvre de protocoles de communication sécurisée qui vont au-delà de l'application des propriétés de sécurités. Les contraintes d'exploitations doivent être prises en compte au plus tôt dans la conception d'un protocole afin que ce dernier soit adapté au besoin. Comme indiqué au paragraphe précédent, une contrainte basique est la simple possibilité de mise en place du protocole. Cette contrainte impose des restrictions sur les outils utilisables et leur déploiement. D'autres contraintes d'exploitations qui seront détaillées dans la suite du chapitre incluent les performances qui ne doivent pas nuire à l'expérience utilisateur, l'indépendance des outils utilisés vis-à-vis de composants non maîtrisés de l'application et la disponibilité sur différents systèmes notamment des systèmes mobiles.

Nous allons étudier dans ce chapitre des solutions permettant de mettre en place des systèmes de communication sécurisée répondant à la fois aux propriétés de sécurité proposées et aux contraintes d'exploitations. Ces dernières vont au-delà de la simple mise en place d'outils à chaque extrémité d'une communication. Cela implique que les outils de sécurité considérés ne peuvent pas dépendre d'éléments comme l'accès à la configuration avancée d'un système d'exploitation pour fonctionner. Pour cette raison, les solutions à base de VPN ne seront pas davantage détaillées.

Nous avons pour cela développé un protocole de communication sécurisée permettant d'établir une connexion sécurisée point à point entre deux systèmes. Les termes *client* et *serveur* sont utilisés ici pour désigner les deux extrémités d'une communication. Il ne s'agit pas de limiter les descriptions au modèle "un serveur, plusieurs clients". Ces termes sont utilisés ici pour différencier le système en attente de connexion (le *serveur*) du système initiant de façon active la connexion (le *client*). Ce chapitre couvre donc d'autres modèles, incluant les connexions point à point pour lesquels les rôles "serveur" et "client" restent présents.

L'annexe A présente un rappel rapide des outils cryptographiques auxquels il est fait référence tout au long du chapitre.

1.1 Protocoles de communications standards

Il existe actuellement plusieurs protocoles permettant de fournir certaines des propriétés de sécurité décrites dans ce chapitre. Nous nous intéressons ici aux protocoles permettant de sécuriser des communications réseau. Les trois outils les plus courants sont TLS [4], SSH [5] et IPSec [6]. Il existe d'autres protocoles spécialisés pour

4. Transport Layer Security, protocole de communication sécurisée remplaçant SSL

5. Secure Shell, protocole de communication sécurisée permettant notamment l'ouverture d'une invite de commande sur un système distant

6. Internet Protocol Security, protocole de chiffrement des échanges au niveau IP

des usages spécifiques. Nous pouvons citer en exemple le protocole ZRTP [7] dédié au chiffrement des communications VoIP [8], ainsi qu'OTR [9] dédié aux échanges de messages instantanés. Ces deux solutions sont très spécifiques à des problèmes particuliers et ne seront pas étudiées plus en détail dans ce document.

Du point de vue de l'utilisation, les protocoles et outils que sont TLS, SSH et IPSec ont tous un fonctionnement similaire : ils permettent l'établissement d'une connexion sécurisée, utilisée par la suite pour l'échange de nombreux messages.

Les solutions construites sur IPSec sont, comme les VPN, incompatibles avec les contraintes d'exploitations, notamment la possibilité de mise en œuvre en mobilité. La figure 1.1 présente deux façons générales d'établir des communications sécurisées. Seule des approches ne nécessitant pas la participation active du système d'exploitation sont considérés, afin de rester compatible avec les contraintes d'exploitations. C'est pour cela qu'IPSec comme les VPN ne seront pas davantage détaillés ici.

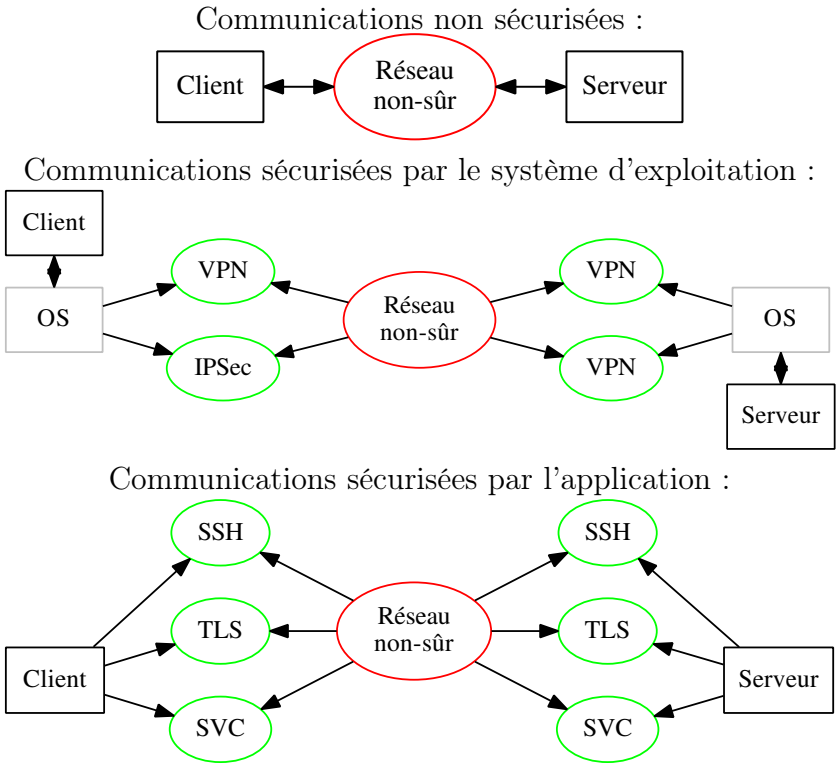

FIGURE 1.1 – Méthodes d'établissement de communication sécurisée

7. Z-Real-time Transport Protocol, protocole de protections des communications VoIP
8. Voice over IP, voix sur IP, communications audio circulant sur des réseaux informatiques basés sur le protocole IP
9. Off-the-record, confidentiel

Nous ne considérons que les protocoles TLS et SSH qui sont les protocoles les plus utilisés pour sécuriser les communications au niveau applicatif. Dans leur fonctionnement minimum, TLS et SSH permettent d'assurer la confidentialité des échanges et l'authenticité du serveur. Au-delà de ces fonctionnalités minimales, ils permettent sous certaines conditions d'assurer des propriétés d'authenticité du client. Dans la suite du chapitre, nous présentons en détail ces deux protocoles. En particulier sont décrits les éléments qui réduisent leur efficacité pour certaines classes d'applications (communications internes à une application, voir section 1.2.3), qu'il s'agisse d'affaiblir la sécurité ou d'impacter négativement les performances du système.

1.1.1 TLS

TLS est actuellement le protocole le plus utilisé pour sécuriser les communications au niveau applicatif sur réseau TCP/IP. Il fait suite au protocole SSL [10], publié pour la première fois par Netscape[24]. Deux versions de SSL ont été exploitées : SSL2.0 et SSL3.0. Bien qu'initialement basé sur SSL, TLS n'est pas strictement compatible avec ces dernières. Les deux objectifs principaux de TLS sont la sécurité des communications et l'interopérabilité entre implémentations, comme cela est décrit dans la RFC 5246[14]. La sécurité des communications fournie par TLS est basée sur des opérations cryptographiques incluant chiffrement, échange de clés et signatures numériques pour l'authentification. L'interopérabilité est fournie par une série d'étapes de négociation permettant d'une part de déterminer les capacités cryptographiques des deux systèmes cherchant à établir une connexion sécurisée et d'autre part à échanger dans un format standardisé les informations d'identification.

Pour fonctionner lors d'interaction avec des tiers inconnus, l'interopérabilité nécessite le support d'anciens protocoles (notamment SSL et les premières versions de TLS). En effet, l'utilisation de ces différents protocoles que ce soit du côté des applications (clients comme serveur) est toujours très répandue, comme indiqué dans la table 1.1. Les premiers chiffres présentés sont issues de l'observation d'un million de sites "populaires" début 2014[67]. Sur ce million, 45,1% disposaient d'un accès TLS (soit 451 470 serveurs). Les valeurs plus récentes proviennent du site SSLLabs[46] et sont produites à partir des résultats de tests de près de 200 000 serveurs visant à évaluer leur niveau de support du protocole TLS. Si un net recul des anciennes versions du protocole (versions non sûres) est visible, elles restent toutefois très présentes et restent un vecteur d'attaque important. De même, les versions plus récentes restent sous-représentées, entre autre à cause de l'inertie liée au standard.

10. Secure Socket Layer, protocole de communication sécurisée remplacé par TLS

Version	Date de création	Disponibilité début 2014	Disponibilité mi-2015	
SSL2.0	1995	18,9%	10,8%	(-8,1)
SSL3.0	1996	99,6%	33,8%	(-65,8)
TLS1.0	1999	98,9%	99,2%	(+0,3)
TLS1.1	2006	32,2%	64,3%	(+32,1)
TLS1.2	2008	33,2%	66,5%	(+33,3)

TABLE 1.1 – Disponibilité des différentes versions de SSL/TLS

Disponibilité de chaque versions pour 451 470 serveurs permettant l'utilisation des protocoles SSL et TLS pour les chiffres de 2014, et sur près de 200 000 serveurs pour les chiffres de 2015.

1.1.1.1 Sécurité

TLS fournit la confidentialité des échanges entre des clients et un serveur, ainsi que l'authentification de ce dernier. Lorsque la négociation TLS aboutit le client a la garantie qu'il est bien connecté à un serveur légitime et que leur échange est confidentiel. Nous pouvons alors parler de connexion sécurisée. Il est également possible d'utiliser le protocole TLS pour authentifier le client. Dans les deux cas, l'authentification est basée sur un système de clé publique utilisant des certificats X.509[9]. L'authentification du serveur peut être réalisée par le client sans échange préalable, en s'appuyant sur un système de tiers de confiance connus du poste client. Il faut noter que dans tous les cas, les informations d'identité, c'est-à-dire les certificats, sont échangée sans chiffrement. Cela inclut le certificat du client si l'authentification via TLS est utilisée. TLS ne fournit donc pas de confidentialité sur l'identité des utilisateurs, mais uniquement sur le contenu des échanges.

Les protocoles SSL et TLS fonctionnent sur un schéma similaire : une phase de négociation permet d'établir les paramètres de sécurité de l'échange et une phase d'exécution met en œuvre ces paramètres.

Phase de négociation : La phase de négociation implique deux éléments : le choix des algorithmes à utiliser et l'exécution de l'échange de clés. Le choix des algorithmes se fait sur la base des informations échangées entre le client et le serveur : le client annonce la liste d'algorithmes qu'il supporte et le serveur choisit dans cette liste ceux qu'il supporte et qui présentent le meilleur niveau de sécurité. En théorie, cela permet de garantir un niveau de sécurité minimale depuis les deux extrémités de la connexion. En pratique, il existe encore de nombreux clients et serveurs dépendant d'algorithmes vulnérables, qui imposent par inertie de conserver leur support. La table 1.2 présente les principaux algorithmes supportés par les différentes versions de SSL et TLS. Nous constatons que les plus anciennes versions ne supportent que peu (ou pas) d'algorithmes permettant d'apporter la

confidentialité persistante. De manière générale il est déconseillé d'utiliser SSL. Pour assurer un niveau de sécurité acceptable il faut déployer au minimum des solutions basées sur TLS version 1.1. Finalement, bien que TLS permette une négociation sûre basée sur un échange de clés de type ECDH [11] fournissant à la fois un haut niveau de sécurité et une confidentialité persistante, leur usage reste optionnel et il est toujours possible d'utiliser des échanges de clé plus faibles en termes de propriétés de sécurité.

Version du protocole	Échange de clés (sans conf. persist.)	Échange de clés (avec conf. persist.)
SSL2.0	RSA	-
SSL3.0	DH-(RSA/DSS)	DHE(RSA/DSS)
TLS(v1, 1.1, 1.2)	ECDH-(RSA/DSS),	ECDHE(RSA/DSS)
	PSK, PSK-RSA, SRP-(RSA/DSS)	(DHE/ECDHE)-PSK

TABLE 1.2 – Algorithmes d'échange de clés de TLS

DH :Diffie-Hellman, DHE :Ephemeral Diffie-Hellman, ECDH :Elliptic-Curve DH, PSK :Pre-Shared Key, SRP :Shared Password, conf. persis. : Confidentialité persistante

Chiffrement : Une fois la phase de négociation complétée, le protocole TLS est utilisé pour chiffrer les échanges afin d'apporter la propriété de confidentialité. Le chiffrement est réalisé à partir des paramètres négociés, dont le choix de l'algorithme à utiliser. TLS supporte différent algorithmes de chiffrement, qui peuvent être utilisés de différentes façons, selon les versions du protocole. Certains algorithmes sont maintenant considérés comme vulnérables. En particulier l'utilisation du mode CBC [12] par SSL et les premières versions de TLS n'étaient pas faites de façon sécurisée. Les algorithmes de chiffrement supportés par SSL et les premières versions de TLS incluent des algorithmes considérés comme non sûres comme DES [13] ou RC2 [14]. Comme montré dans la table 1.3, seule la version 1.2 de TLS permet d'utiliser autre chose que le mode CBC pour les opérations de chiffrement. L'utilisation en place de ce mode de chiffrement est à la base de plusieurs attaques sur différentes versions de TLS.

11. Elliptic-Curve Diffie-Hellman, Diffie-Hellman basé sur courbes elliptiques
12. Cipher Block Chaining, enchaînement de blocs chiffrés : mode d'utilisation consistant à faire intervenir pour chaque bloc chiffré l'état du bloc précédent, afin de construire une chaîne
13. Digital Encryption standard, ancien algorithme de chiffrement par blocs
14. Rivest Cipher 2, algorithme de chiffrement par bloc

Version du protocole	Algorithmes supportés	Mode de chiffrement
SSL(2.0,3.0)	3DES,IDEA,DES,RC2,RC4	CBC
TLS1.0	+(AES,Camellia,ARIA)	identique
TLS1.1	-RC2	identique
TLS1.2	-(IDEA,DES)	+(GCM,CCM)

TABLE 1.3 – Algorithmes de chiffrement de TLS

Le mode de chiffrement s'applique uniquement aux chiffrements par blocs. Les lignes concernant TLS montrent les ajouts/suppressions par rapport à la ligne précédente.

1.1.1.2 Fonctionnement

TLS agit comme une couche intermédiaire entre l'application et la connexion TCP/IP utilisée pour la communication. L'exemple de fonctionnement le plus connu de TLS est le protocole HTTPS [15][56] utilisé pour transporter des requêtes HTTP [16] de façon sécurisée. Dans cet exemple, des requêtes HTTP non sécurisées sont encapsulées par le protocole TLS afin de sécuriser les échanges. Le traitement au niveau applicatif, que ce soit côté client ou côté serveur, a lieu indépendamment du protocole TLS sur les requêtes après leur déchiffrement. Le protocole TLS dégage l'application de tous les aspects de sécurité de la communication en assurant de façon quasi transparente l'authentification et le chiffrement. La seule contrainte sur l'utilisation du protocole TLS est la disponibilité de composants logiciels permettant de l'implémenter, qu'il s'agisse du protocole lui-même ou des algorithmes cryptographiques utilisés.

TLS est également utilisé comme élément de sécurité pour les VPN. La fonction de base d'un VPN est de créer un réseau logique virtuel pouvant tourner sur plusieurs réseaux différents, afin d'offrir une continuité réseau à des machines distantes. Comme pour HTTPS, les communications de base VPN ne sont pas sécurisées. Ces communications peuvent en revanche être encapsulées dans différents protocoles de sécurité, dont TLS, afin de fournir une connexion sécurisée.

1.1.1.3 Faiblesses

Bien que la dernière version du protocole TLS, la version 1.2, apporte des solutions à la plupart des problèmes des versions précédentes, il existe des faiblesses liées à la construction du protocole lui-même qui ne peuvent pas être corrigés sans en modifier le fonctionnement. D'autres vulnérabilités sont liées à un manque de contrôle de l'environnement d'utilisation du protocole TLS qui dépend de nom-

15. Hypertext Transfer Protocol Secure, version de HTTP sécurisée par TLS
16. Hypertext Transfer Protocol, protocole utilisé pour le transfert de pages Web

breux éléments extérieures comme la configuration du système d'exploitation ou l'utilisation d'une bibliothèque extérieure pas nécessairement maintenue à jour.

Algorithmes vulnérables : Le point faible le plus exploitable des protocoles SSL et TLS est l'utilisation d'algorithmes cryptographiques vulnérables, ou de combinaisons d'algorithmes introduisant de nouvelles vulnérabilités. Comme indiqué précédemment, ces algorithmes sont toujours largement utilisés. Certains algorithmes en particulier ont posé ou posent d'importants problèmes. Le premier, RC4 [17], est considéré comme vulnérable. Son utilisation est interdite dans toutes les versions de TLS[52]. Malgré cela, il y a toujours des clients et des serveurs en exploitation qui utilisent cet algorithme : 56,1% de serveurs supportent RC4 d'après les résultats de 2014 d'une enquête permanente[46]. Le deuxième problème majeur dans les algorithmes mis en œuvre par TLS est l'utilisation incorrecte du mode CBC. Ce mode d'utilisation, comme beaucoup, nécessite un vecteur d'initialisation pour fonctionner. Bien que non secret, ce vecteur d'initialisation doit être unique et imprévisible pour chaque utilisation, sans quoi de nombreuses attaques peuvent être appliquées. Dans le cas des versions vulnérables (SSL2.0, SSL3.0, TLS1.0), le vecteur d'initialisation est prévisible et peut permettre de récupérer le contenu confidentiel de la communication. Pour finir sur ce point, les algorithmes utilisés pour assurer l'intégrité des données, de type HMAC [18], sont basés sur des fonctions de hachage faibles : soit MD5 [19], soit SHA1 [20] pour toutes les versions de TLS avant la version 1.2. Seule cette dernière inclut de nouveaux algorithmes comme SHA2 [21]. La faiblesse du contrôle d'intégrité peut permettre l'altération des données sans que cela ne soit détecté. Combiné à d'autres faiblesses, il devient alors possible de falsifier des messages apparemment légitimes. L'utilisation de ces algorithmes peut à elle seule rendre inutile la protection fournie par TLS en rendant le chiffrement inefficace ou en permettant de falsifier des messages.

Rétro-compatibilité : Le protocole TLS permet de revenir à d'anciennes versions, y compris à des versions de SSL. Cette rétro-compatibilité permet de conserver l'interopérabilité dans le cas où un client se connecterait à un serveur ne disposant pas des dernières versions du protocole ou vice versa. Cependant, cette interopérabilité permet des attaques sur TLS qui forcent la négociation à utiliser une version vulnérable du protocole afin d'exploiter des failles connues et corrigées[49]. Nous pouvons citer la faille POODLE, qui exploite une faiblesse de

17. Rivest Cipher 4, algorithme de chiffrement à flot
18. Hash-based Message Authentication Code, code d'authentification de message basé sur des fonction de hachage
19. Message Digest 5, fonction de hachage simple
20. Secure Hash Algorithm 1, fonction de hachage
21. Secure Hash Algorithm 2, fonction de hachage succédant à SHA1, plus sûr que ce dernier

SSL version 3, et qui pouvait impacter des systèmes disposant de versions plus récentes, grâce à la rétro-compatibilité. De manière générale, les recommandations sur l'interdiction d'anciens algorithmes et les changements dans leurs modes d'utilisation, ne s'appliquent pas aux anciennes versions du protocole. Pour réellement renforcer la sécurité des communications basées sur TLS, il a fallu limiter cette rétro-compatibilité en interdisant de descendre à un niveau inférieur à TLS1.0[6].

Environnement d'utilisation : Les problèmes liés à l'environnement dans lequel est utilisé le protocole TLS peuvent compromettre la sécurité à plusieurs niveaux. L'utilisation d'une PKI [22] gérée par un tiers (en général le système d'exploitation) peut faciliter des attaques de type MitM [23] ou d'usurpation d'identité[43]. En effet, l'authentification du serveur est basée sur la validation de son certificat. Cette validation dépend d'un magasin de certificats dit "de confiance" stocké sur la machine cliente et géré le plus souvent par le système d'exploitation. L'injection d'un certificat par un attaquant dans ce magasin compromet l'élément de confiance qu'il fournit, rendant invalide son utilisation pour l'authentification du serveur et donc, cassant la sécurité fournie par TLS.

Un autre problème lié à l'environnement d'utilisation de TLS est en rapport avec l'origine de l'implémentation utilisée. Certains systèmes d'exploitation fournissent directement leur implémentation de TLS. Cela permet un déploiement plus simple, mais associe la sécurité de l'application à la sécurité fournie par le système d'exploitation. Au-delà du risque posé par l'utilisation d'un élément critique provenant d'une source non contrôlée, il existe un problème bien plus important : celui des mises à jour du système d'exploitation. En effet, obtenir des mises à jour de sécurité pour l'ensemble du système devient obligatoire pour contrer de nouvelles vulnérabilités. Cependant, certains systèmes d'exploitation ont une durée de maintenance très courte et ne peuvent rapidement plus être mis à jour. Cela n'empêche pas leur utilisation pendant cette période de "fin de vie". Le déploiement d'applications sur des environnements mobiles tels que les smartphones est particulièrement concerné par cette restriction, car le plus souvent il est très difficile (parfois même impossible) d'obtenir des mises à jour pour des appareils encore couramment utilisés. Dans ces conditions, les applications déployées pour de telles systèmes et dépendantes de composants logiciels obsolètes peuvent devenir vulnérable au fil de la découverte de nouvelles vulnérabilités, sans possibilité de corrections.

22. Public Key Infrastructure, infrastructure à clé publique : ensemble de systèmes permettant la gestion de certificats numériques de clé publiques

23. Man in the Middle, l'homme au milieu : intercepter et/ou modifier les communications entre deux systèmes

Fonctionnalités superflues : Certaines fonctionnalités de TLS sont parfois superflues et ont pu servir de vecteur d'attaque pour affaiblir la sécurité des communications. Parmi ces fonctionnalités nous comptons des optimisations de renégociation et une fonctionnalité de "battement de cœur" (ping). La renégociation devait permettre d'économiser des ressources dans le cas ou un même client devait établir plusieurs connexions successives avec un même serveur. Le fonctionnement est le suivant : les deux extrémités de la communication retiennent les paramètres négociés, associé à un numéro de session. Lors d'une nouvelle connexion, un client peut envoyer son numéro de session. Si le serveur retrouve cet identifiant, la session précédente est réutilisée. Cette fonctionnalité permet dans certaines conditions à un attaquant d'injecter un message dans l'échange en se faisant passer pour le client initial ou de complètement intercepter une communication sécurisée. L'autre fonctionnalité ayant servi à attaquer le protocole est le "heartbeat", ou "battement de cœur". Cette fonction, similaire à un "ping", sert essentiellement à s'assurer que le serveur est toujours actif. En soi, elle ne présente pas de problème particulier, mais une vulnérabilité logicielle a été trouvée dans la bibliothèque OpenSSL, l'une des implémentations les plus répandues. Cette vulnérabilité permet à un attaquant de récupérer des blocs arbitraires de la mémoire du serveur et ce sans laisser de traces. Parmi les données ainsi exposées il est possible de trouver des informations sur les utilisateurs, mais surtout la clé privée utilisée par le serveur. La compromission de cette dernière rend possible la mise en place par un attaquant d'un serveur usurpant l'identité du serveur initial. Puisque la fonctionnalité de base "heartbeat" est peu utilisée en pratique, cette vulnérabilité se trouve être une illustration intéressante du fait que plus un protocole est "large" en application, plus sa surface d'attaque est importante. Cette faille est connue sous le nom de "heartbleed"[15, 27]. Dans les deux cas, la solution à court terme a été de désactiver la fonctionnalité mise en cause, ce sans perturbation majeure à l'utilisation. En effet, la renégociation de TLS, bien que coûteuse, n'a pas un impact très important sur des systèmes correctement dimensionnés et la faille "heartbleed" exploitant une fonction peu utilisée, sa désactivation n'a pas posé de problèmes. Ces deux exemples ont fait l'objet de corrections par la suite, sous la forme d'une renégociation plus sûre pour TLS[57] et sous la forme d'un correctif logiciel pour OpenSSL concernant "heartbleed".

En plus des problèmes évoqués, il existe aussi des problèmes liés à la façon d'utiliser certains algorithmes. Des attaques existent, dont certaines sont encore applicables à TLS version 1.2[62, 60].

En dehors des problèmes de vulnérabilités et d'attaques sur l'utilisation du protocole, il y a un élément du protocole qui s'avère être un point faible dans le contexte des communications sécurisées envisagé dans ce chapitre. Les informations d'identité, qu'il s'agisse du serveur ou du client, sont échangées sans aucune protection. Il n'y a lors de cette étape aucun chiffrement mis en œuvre. De manière

générale, si une connexion avec un serveur “fixe” est établie, dévoiler l'identité de ce système n'est pas un problème majeur.

En revanche, si l'authentification du client via TLS est utilisée, l'information d'identité du client est transmise et peut permettre d'identifier les utilisateurs d'un service de façon fiable. Cela pose un problème pour la protection de la vie privée et ne peut pas être corrigé dans la version actuelle de TLS. Afin d'éviter ce problème, toute application utilisant TLS et souhaitant authentifier ses clients doit mettre en place un mécanisme secondaire d'authentification opérant via la connexion sécurisée par TLS. Cette approche est contraignante ; en effet si l'authentification de l'utilisateur doit se faire sur une connexion sécurisée, cela signifie terminer l'ensemble de la négociation dans un premier temps et authentifier le client ensuite. Dans le cas ou l'authentification échouerait à ce niveau, les ressources utilisées pour l'établissement de la connexion sécurisée ont déjà été consommées. Cette utilisation superflue de ressources tant sur le serveur que sur le client peut mener à une autre forme d'attaque : le DoS [24].

1.1.2 SSH

Le protocole SSH est largement et principalement utilisé pour interagir avec des systèmes distants via l'ouverture d'une invite de commande au travers d'une connexion réseau, mais ce n'est pas là son seul usage. En fait, SSH est la combinaison de trois éléments qui lui permettent d'agir comme une couche de transport sécurisé pour différents types d'applications[72].

Le premier élément est appelé la couche transport[74]. Cet élément fournit des communications bidirectionnelles sécurisées entre le client et le serveur, ainsi que l'authentification du serveur. Cette authentification est basée sur la machine hôte et identifie donc le système plutôt qu'un utilisateur ou un service. Elle peut se faire de plusieurs façons, mais le plus souvent une procédure d'authentification à clé publique est utilisée. Lors de la première connexion, cette clé est récupérée par le client puis validée et retenu pour les connections ultérieures afin de ne pas avoir à répéter le processus de validation. La couche transport de SSH permet aux autres composants de bénéficier d'une connexion sécurisée, car elle fournit le chiffrement pour la confidentialité ainsi que l'intégrité des échanges.

Le deuxième élément du protocole est le protocole d'authentification[71]. Il s'exécute avec le support de la couche transport sécurisé, ce qui permet de garder confidentielles les informations d'identification de l'utilisateur. Le protocole d'authentification de SSH fournit plusieurs méthodes d'authentification, dont des mécanismes basé sur un système challenge/réponse et des signatures à clé publique.

24. Denial of Service, déni de service : empêcher le fonctionnement normal d'un service en en bloquant l'accès ou les processus

Il inclut également des méthodes basiques comme la transmission des identifiants en texte clair, mais ces dernières sont plus rarement utilisées.

Le troisième et dernier élément du protocole SSH est le protocole de connexion[73]. Il permet de réaliser des communications au niveau applicatif grâce à la gestion de différents canaux pouvant être utilisés pour différents types d'informations. Ces canaux sont ensuite multiplexés de façon transparente sur la couche transport de SSH. Il peut y avoir par exemple un canal pour l'utilisation d'une invite de commande et un second canal pour réaliser du routage IP de façon sécurisée entre le client et le serveur. Comme ces différents canaux utilisent la couche transport pour leur communication effective, ils sont automatiquement rendus confidentiels et leur intégrité est vérifiée.

Il existe aujourd'hui peu de faiblesses connues sur le protocole SSH. La plupart des vulnérabilités datent de la première version du protocole[11] et ont été rendues obsolète par la seconde version (SSH-2). L'adoption quasi universelle de la deuxième version du protocole fait qu'il est acceptable et recommandé de complètement désactiver le support pour la version initiale, ce qui évite les problèmes liés à la rétrocompatibilité. Comme pour TLS, il peut persister des faiblesses liées à l'utilisation d'algorithmes de chiffrement faibles[3]. De la même manière, la technique permettant de réduire l'impact de ces faiblesses consiste à retirer les algorithmes faibles de l'étape de négociation. Un autre problème du protocole SSH est la difficulté à l'intégrer dans certaines applications. L'implémentation la plus populaire (OpenSSH) est conçue pour fonctionner en tant que processus indépendant, ce qui n'est pas toujours possible. D'autres implémentations existent, mais elles posent pour la plupart des problèmes de portabilité ou de licences. Il s'agit principalement de produits commerciaux dont la disponibilité est limitée à certaines architectures.

1.1.3 Autres approches

Comme indiqué précédemment, il existe d'autres types de solutions pour établir des communications sécurisées que l'utilisation d'un protocole de communication sécurisée. Parmi les exemples donnés, nous considérons l'utilisation de VPN et IPSec, ainsi que les protocoles comme OTR. Il existe également des approches moins standardisées, mais s'approchant toujours d'une solution existante. Certains systèmes permettent, de façon similaire au protocole OTR, d'envoyer des messages hors-ligne à un destinataire connu à l'avance, via un système à clé publique. Ces approches ne sont pas nécessairement moins sécurisées et certaines peuvent fournir l'ensemble des propriétés de sécurité décrites ci-dessus. Cependant, elles présentent des contraintes non pas de sécurité, mais d'utilisation, qui les rendent incompatibles avec le modèle considéré. C'est pour cela que cet état de l'art s'est concentré sur les protocoles répondant aux problématiques de connexion sécurisée dans un contexte compatible avec le déploiement d'applications sur systèmes variés.

1.2 Types de communications

Des clients lourds aux applications basées sur le cloud computing, les logiciels modernes dépendent des communications réseaux pour fonctionner. Ces communications peuvent contenir des informations confidentielles ou sensibles et nécessitent donc une sécurité adaptée.

Il y a plusieurs approches pour catégoriser les communications ; par exemple basé sur le contenu (privée/publique), le volume de données ou la latence. Ici nous considérons une contrainte de nature différente. Nous séparons les communications en deux types distincts : les communications dites "externes" ou interopérable et les communications "internes". Les communications externes regroupent les communications établies entre applications différentes. Un exemple pour ce type de communication est l'ouverture d'une page web par un navigateur communiquant avec un serveur HTTP. Les deux logiciels concernés ici, le client et le serveur, peuvent être totalement indépendant l'un de l'autre. Pour que ce type d'échanges fonctionne, il est nécessaire d'avoir un protocole de communication permettant l'interopérabilité, en l'occurrence le protocole HTTP. Les communications internes regroupent les échanges réalisés entre les instances d'une même application. Cela recouvre entre autre le schéma classique d'une application client/serveur, pour laquelle le logiciel client et le logiciel serveur font partie d'une unique application. Dans ce type de communications, l'interopérabilité n'est pas requise, car la même entité a le contrôle des deux extrémités lors des échanges.

Les protocoles de communication sécurisée TLS et SSH correspondent au modèle externe. Même s'il est possible de les utiliser pour les communications internes ils disposent de nombreux éléments qui deviennent alors redondants. Ces éléments, comme une phase de négociation complexe, sont des vecteurs d'attaques potentiels. Dans ce chapitre, nous nous intéressons au cas des communications internes et ce pour deux raisons. Tout d'abord, nous avons pu voir qu'il existait de nombreux protocoles existants pour le cas des communications externes, alors qu'il n'en existe pas pour le cas des communications internes. Cela laisse un vide qu'il est utile de combler. Ensuite, le modèle des communications internes permet d'avoir plus de latitude vis-à-vis des contraintes qui peuvent être imposées à l'utilisation. En particulier, nous pouvons dans ce cas présumer du support de certains algorithmes ou de l'existence d'une solution de mise à jour unifiée applicable aux différents composants intervenant dans un échange. Ces contraintes supplémentaires permettent de réduire considérablement le nombre de situations attendues lors de la négociation d'une communication sécurisée. Une conséquence de cela est qu'il devient possible de développer un protocole de communication sécurisée pour lequel l'étape de négociation est simplifiée sans faire de compromis sur les propriétés de sécurité fournies.

1.2.1 Contraintes communes

Qu'il s'agisse du cas "externe" ou "interne" l'ensemble des applications communicantes partagent des contraintes communes : performances, fiabilité du support de communication et sécurité du système d'exploitation.

Performances : Il y a plusieurs types de métriques lorsque l'on parle de performances des communications comme la bande passante, la latence, la gigue, le nombre d'échanges, le nombre de sauts, etc. Dans le contexte des protocoles de communications au niveau applicatif, nous nous intéressons aux métriques sur lesquelles l'application peut avoir une influence : la bande passante, la latence et le nombre d'échanges. Nous pouvons aussi ajouter aux métriques de performances réseau le coût en ressources processeurs et mémoire imposé sur les systèmes communicants par le protocole de communication. Un protocole de communication sécurisée impacte négativement toutes ces métriques. Afin d'établir une connexion sécurisée, un protocole de communication sécurisée doit ajouter des données permettant de mettre en œuvre au moins le chiffrement. Ces données additionnelles peuvent faire l'objet d'une négociation initiale, ou de données ajoutées à chaque message. L'ajout d'autres propriétés de sécurité (comme l'intégrité) impose aussi d'ajouter davantage de données à un échange. Ces éléments additionnels augmentent le besoin en bande passante de l'application et dans le cas d'une négociation initiale, augmentent le nombre d'échanges nécessaires sur le réseau. L'utilisation d'algorithmes complexes, notamment d'algorithmes cryptographiques, impose un coût en temps de calcul qui s'ajoute au temps de traitement de l'application. Ce temps de calcul impacte lui aussi négativement les performances réseau, car il peut augmenter la latence de l'application.

Fiabilité du support de communication : La sécurité d'une communication ne se limite pas à la confidentialité. Parmi les propriétés de sécurité décrites au début de chapitre, l'intégrité et l'authenticité des communications sont essentielles. Nous considérons deux causes possibles de falsification de message lors d'un échange sécurisé : cause accidentelles et malveillance. Le cas de la malveillance est clair : elle est du fait d'un attaquant cherchant à modifier le contenu d'une communication. Les causes accidentelles peuvent être plus nombreuses. Les différents supports utilisés pour les communications numériques pouvant introduire des erreurs accidentelles, il est essentiel de pouvoir détecter ces cas, même si certains types de support (par exemple les communications TCP/IP) ont une garantie d'intégrité des messages. Notons que cette garantie est là pour détecter les altérations accidentelles et ne protège pas contre la malveillance. Dans les deux cas, du point de vue d'un protocole de communication sécurisée, il est primordial de pouvoir garantir de façon fiable et prouvable l'intégrité d'un message. Cela prémunit

à la fois contre la malveillance et les accidents de communication.

Sécurité du système d'exploitation : La nature modulaire des systèmes modernes a conduit à l'intégration de nombreuses fonctionnalités au niveau du système d'exploitation, que ce soit sous la forme d'une API [25] spécifique ou sous la forme de bibliothèques logicielles embarquées. Comme exemple, nous pouvons citer le déploiement de bibliothèques logicielles gérées par différentes distributions Linux, l'utilisation de fonctions cryptographiques de systèmes d'exploitation tels que Windows ou MacOS ou encore l'intégration de bibliothèques dans le système d'exploitation Android. La fourniture par le système d'exploitation de composants de sécurité introduit une dépendance entre la mise à jour générale du système d'exploitation et la mise à jour de ces bibliothèques. Cette dépendance devient un problème lorsque l'on se place dans un contexte de sécurité, en particulier pour les bibliothèques de cryptographie. En effet, dans le monde de la sécurité, les mises à jour logicielles sont parfois nombreuses et doivent être rapidement déployées pour contrer des attaques dites "zero day". Ces dernières mettant en œuvre des vulnérabilités dès leur découverte (parfois même avant leur publication), la réactivité dans le déploiement de correctifs est essentielle. C'est dans ces conditions que la dépendance au système d'exploitation pour des services de sécurité est un problème qu'il s'agisse de délai avant le déploiement de correctifs, ou plus simplement de l'absence de mises à jour, car le cycle de vie du système d'exploitation est considéré comme terminé par son fournisseur. Cette dépendance peut rendre vulnérable une application. Ce problème n'est pas directement dû à la construction d'un protocole de communication sécurisée, mais il peut être limité dans sa spécification. Des recommandations sur la façon d'intégrer et d'utiliser un protocole sont essentielles pour éviter cette situation.

1.2.2 Communications interopérables

Le cas des communications externes, ou "interopérable", nécessite que des logiciels provenant de différentes sources puissent communiquer. Ces logiciels peuvent utiliser des versions différentes d'un même protocole, ou des versions identiques, mais supportant des ensembles différents d'algorithmes ou d'options. Pour permettre cela, il est nécessaire de supporter au niveau du protocole des options de rétro-compatibilité et de négociations d'options. Dans ces conditions, le développeur d'une application ne peut faire que des suppositions faibles sur les algorithmes qui seront supportés par l'autre extrémité d'une communication. Ces éléments de rétro-compatibilité et de sélection d'options élargissent la surface d'attaque pos-

25. Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

sible. Les protocoles qui répondent à ce modèle font un compromis entre la disponibilité et la sécurité, au détriment de cette dernière.

La plupart des protocoles de communication sécurisée, en particulier TLS et SSH se placent dans ce cadre. Comme les communications inter applications peuvent être une nécessité, ces protocoles restent une réponse acceptable dans cette situation. Cependant, ils nécessitent une attention particulière à l'utilisation. En particulier, TLS est la réponse "classique" mais son utilisation doit prendre en compte tous les points faibles évoqués plus haut.

Il est possible de mettre en place un ensemble de recommandations qui permettent d'utiliser TLS avec un bon niveau de confiance. Le protocole SSH ne présente pas les mêmes risques dans une utilisation générique et ne nécessite donc pas de précautions particulières, au-delà des considérations usuelles lors de la mise en place de connexion sécurisées.

Utilisation sûre de TLS : Dans le cas où la situation imposerait l'utilisation d'un protocole interopérable, TLS reste un bon choix, à condition d'éviter les problèmes associés. La première recommandation est de s'assurer de la possibilité d'utiliser la version la plus récente du protocole, que ça soit côté client ou côté serveur. Cela élimine bon nombre de vulnérabilités au niveau protocole mais implique également un suivi des publications sur le sujet afin de s'assurer que seul les versions encore "sûres" sont maintenues actives. Malheureusement, dans le cas de communications avec des entités extérieures ne maintenant pas nécessairement leur système à jour, une version antérieure et vulnérable peut être nécessaire. Dans ce cas, soit les communications utiliseront une ancienne version, donc vulnérable, soit les communications avec cette entité seront impossibles. En plus de cette recommandation initiale, la configuration du système doit être prise en compte. En particulier, s'assurer que les certificats utilisés pour garantir l'authenticité de TLS ne sont pas altérés. Pour cela, une approche simple est de ne pas dépendre du magasin de certificats fourni par le système d'exploitation, mais d'embarquer un magasin de certificats propre à l'application. Enfin, l'implémentation utilisée doit aussi être maintenue à jour, afin d'éviter les vulnérabilités purement logicielles qui pourraient, alors que la version du protocole utilisée est correcte, entraîner une baisse de la sécurité des communications ou des systèmes communicants impliqués. En prenant en compte ces recommandations, on peut éviter les risques majeurs liés à l'utilisation de TLS. D'autres inconvénients persistent, notamment la possible divulgation d'information d'identité du client, ou le besoin important en bande passante.

1.2.3 Communications internes

Le modèle des communications internes permet d'avoir un meilleur contrôle et une meilleure connaissance des deux extrémités d'une communication sécurisée. La levée de la contrainte d'interopérabilité permet de s'affranchir d'un bon nombre de vérifications et donc de réduire la phase de négociation au minimum nécessaire pour l'établissement d'une connexion sécurisée. Cette réduction a deux avantages : elle nécessite moins de bande passante et présente moins d'éléments variables réduisant ainsi la surface d'attaque à cette étape. En effet, contrairement au cas des communications externes, les deux extrémités de la communication sont maîtrisées par la même entité, ce qui permet de faire des suppositions fortes sur les protocoles et algorithmes supportés de chaque côté. De plus, en considérant que tous les éléments constituent un même ensemble applicatif, il est raisonnable de penser qu'un système de mise à jour cohérent existe et permet d'appliquer les correctifs à toutes les instances de l'application. Cet aspect élimine les risques de conflits de versions, où une instance pourrait tenter d'utiliser une version plus ancienne du protocole. Les deux seuls cas à détecter lors d'une négociation sont donc extrêmement simplifiés : soit l'entité à l'autre extrémité de l'échange utilise le même protocole, soit ce n'est pas le cas.

L'idée de développer un protocole dont la principale caractéristique "visible" serait l'absence d'interopérabilité va à l'encontre des solutions courantes consistant à favoriser l'interopérabilité. Cependant, malgré l'absence de solutions existantes pour ce problème, le besoin est réel : les applications ayant un besoin de communications internes, sans interopérabilités, sont une réalité. Le cas typique de ces communications internes est une application constituée d'un serveur et d'applications clientes fonctionnant sur un protocole spécifique ; il n'y a pas de besoin de compatibilité avec des clients génériques, car les deux extrémités forment un tout. Pour ces applications, l'utilisation de solutions génériques comme TLS impose un surcoût en ressources, tout en apportant des risques de sécurité liés à cette généricité. C'est pour cela que le protocole SVC [26] (section suivante, 1.3) a été développé. Sa conception a été guidée autant par des contraintes d'exploitation que par des contraintes théoriques.

1.3 Le protocole SVC

Le protocole SVC[58] a été conçu explicitement pour répondre au cas des communications internes. En effet, l'absence de solutions existantes permettant d'appliquer l'ensemble des propriétés de sécurité souhaitées nécessite l'apport d'une

26. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

solution adaptée. Il faut un protocole de communication ayant un coût minimum en termes de bande passante et de temps de calcul, fournissant toutes les propriétés de sécurité décrites en introduction et pouvant aussi bien s'intégrer à des applications existantes que servir de base à la construction de nouvelles applications.

Pour cela, nous avons conçu le protocole SVC à partir de zéro avec les objectifs suivants : restreindre autant que possible les fonctionnalités disponibles, réduire le surcoût en bande passante imposé par les algorithmes de sécurité, limiter le nombre d'échanges nécessaires à l'établissement d'une connexion sécurisée et fournir l'ensemble de propriétés de sécurité décrites précédemment.

1.3.1 Motivations pour un nouveau protocole

En plus des propriétés de sécurité classiques que sont la confidentialité et l'authenticité, nous avons considéré deux propriétés supplémentaires qui améliorent la sécurité des communications : la confidentialité persistante et la protection de l'identité du client. En rendant la confidentialité persistante obligatoire, les échanges sont non seulement protégés lors de l'échange, mais aussi en cas de fuite d'informations ultérieures. En effet, si la clé cryptographique associée à un utilisateur est divulguée après qu'un échange sécurisé ait eu lieu, cela ne met pas en péril la confidentialité de cet échange. La protection de l'identité du client permet de fournir un niveau minimum d'anonymat en chiffrant les éléments qui permettraient d'identifier un utilisateur lors de l'établissement d'une connexion sécurisée. Ce chiffrement présente le même niveau de sécurité que le reste de la communication, y compris la confidentialité persistante, ce qui fait qu'il n'est pas possible a posteriori, même en s'appropriant les informations de l'utilisateur, de le relier à une communication ayant eu lieu dans le passé.

Le tableau 1.4 résume les propriétés fournies par les trois protocoles TLS, SSH et SVC. Il apparaît que TLS ne fournit pas l'ensemble des propriétés de sécurité requises et que même si SSH fournit l'ensemble de ces propriétés de sécurité, il impose une consommation importante de bande passante pour sa phase de négociation et ne peut pas être intégré aisément dans certains types d'applications.

Puisque notre objectif était de concevoir un protocole en accord avec les contraintes des communications d'applications modernes, nous avons dû prendre en compte des contraintes d'exploitation en plus de contraintes théoriques et de sécurité. Une des contraintes les plus importantes est la mobilité des utilisateurs et de leurs appareils.

En considérant les restrictions et les comportements courants observables sur des réseaux de natures différente, nous avons pu ajuster le protocole SVC pour qu'il puisse s'adapter à différents réseaux et appareils sans compromettre les propriétés de sécurité fournies. La prise en compte des contraintes des différents réseaux observés, notamment les réseaux mobiles, ont motivé la réduction à la fois du coût

Propriétés	TLS	SSH	SVC
Confidentialité	+	+	+
Conf. persistante	O	+	+
Authentification serveur	+	+	+
Authentification client	O	+	+
Protection identité client	–	+	+
Économie de bande passante	–	–	+
Facilité d'intégration	–	–	+

TABLE 1.4 – Propriétés des différents protocoles

'–' : absente, 'O' : optionelle, '+' : obligatoire

en bande passante et du nombre d'échanges nécessaires pour la négociation. Ces limitations permettent un fonctionnement sur tous type de réseau, qu'il s'agisse de réseaux classiques, à faible débit, ou avec des temps de réponses élevés.

Afin d'illustrer l'importance de ces restrictions, qu'il s'agisse des contraintes de capacité réseau ou des restrictions imposées par les propriétés de sécurité supplémentaires, deux cas d'usages sont présentés au paragraphe 1.4.1.2 ainsi qu'une expérimentation au paragraphe 1.4.2. Ces deux cas ont été volontairement choisis pour représenter des cas génériques de communications point à point classiques. En illustrant l'utilisation des différents protocoles de communications sécurisés dans une communication point à point, nous intégrons un grand nombre de cas pratiques, qu'il s'agisse de communication client/serveur, de communications machine à machine, de synchronisation d'instances dans un cloud numérique, etc. Ici, les paramètres essentiels motivant la création d'un protocole dédié sont la vitesse de négociation, l'occupation des réseaux utilisés et la sécurité effective des algorithmes mis en place. Ces paramètres dépendent davantage de l'environnement dans lequel ces communications ont lieu que leur finalité. L'analyse qualitative des différents réseaux étant compliquées d'une part par le nombre de variables et d'autre part par les difficultés d'accès à ces réseaux, le paragraphe 1.4.2 présente une expérimentation sur l'efficacité du protocole SVC dans un réseau perturbé générique.

Une autre contrainte opérationnelle concerne la mobilité des utilisateurs qui a comme conséquence que le protocole doit s'adapter à des politiques de sécurité variées. Ce besoin d'adaptation a été pris en compte dans la conception du protocole afin de pouvoir être flexible notamment au niveau de l'authentification du client sans compromettre la protection de cette étape d'authentification. Cette mobilité augmente la variété des appareils ayant à établir des communications sécurisées et impose des choix formats et d'algorithmes utilisables.

Cas des réseaux restreints : Nous pouvons considérer le cas "classique" des communications comme s'appliquant à des réseaux sur lesquelles les limites en bande-passante et en latence sont raisonnablement larges. Cependant, le nombre d'appareils mobiles (comme les smartphones ou les réseaux de capteurs) bénéficiant d'un accès réseau a explosé ces dernières années. Nous nous trouvons donc dans une situation ou une part importante des échanges opère non pas sur des réseaux classiques, mais sur des réseaux restreints, comme les réseaux mobiles. Ces réseaux peuvent avoir des restrictions bien plus importantes sur leur bande passante, leur latence, ou même sur la nature des contenus autorisés à y circuler. L'utilisation de ces réseaux n'est par ailleurs pas limitée aux appareils mobiles. Ils sont parfois aussi utilisés pour des systèmes fixes en utilisant un smartphone comme passerelle réseau. Puisque la notion de communication sécurisée implique, au-delà de la confidentialité, d'apporter une certaine garantie sur la délivrance des messages, un protocole sécurisé doit s'accommoder des restrictions des supports de communication considérés.

Politiques de sécurité : Les politiques de sécurité concernant les applications (mobiles ou non) peuvent recouvrir plusieurs aspects, allant de l'authentification des utilisateurs au stockage sécurisé des données sur un appareil client. La mobilité des utilisateurs impose de nouvelles approches notamment sur la fiabilité de l'authentification d'un utilisateur. Cette authentification pouvant avoir lieu sur plusieurs appareils de natures différentes, avec des modalités d'authentification différentes. Un exemple d'un tel système est l'utilisation d'une authentification à clé publique. Dans un tel système, nous conservons les clés cryptographiques de l'utilisateur sur son appareil. Dans le cas d'un système mobile, il faut mettre en place des solutions permettant de conserver un haut niveau de confiance dans le processus d'authentification tout en prenant en compte le fait qu'un utilisateur puisse utiliser de nombreux appareils différents pour s'authentifier. Le protocole SVC en lui-même n'a pas pour objectif d'apporter une solution à cette question. En revanche, il fournit un mécanisme de sécurité permettant à l'application qui l'utilise de mettre en œuvre de façon sécurisée un système d'authentification au niveau de l'application. Un espace est réservé lors de la négociation d'une communication sécurisée permettant à l'application d'utiliser un mécanisme d'authentification de son choix. Cet espace est transmis de façon sécurisée et confidentielle. De cette façon, l'intégration du protocole SVC dans une application existante permet de continuer à utiliser un éventuel mécanisme d'authentification existant. Cette approche permet de respecter les politiques de sécurité existante en fournissant un moyen sécurisé d'exécuter un protocole d'authentification quelconque, spécifié par l'application. Le protocole SVC est ici en charge uniquement de la protection de cet échange d'authentification, déléguant ainsi l'authentification à un protocole

dédié. Nous disposons ainsi d'une flexibilité importante sur ce point sensible, sans risque de compromettre la confidentialité des informations utilisées pour l'authentification.

Sécurité du système d'exploitation : Un autre élément qui a mené à la conception du protocole SVC est la sécurité des appareils. Différents systèmes d'exploitation peuvent fournir des outils de sécurité dont le comportement peut être modifié de façon significative. L'exemple le plus courant est la présence d'un trousseau de clés cryptographiques géré par le système. Si un attaquant parvient à altérer ce trousseau de clés, il peut totalement compromettre la sécurité des applications qui en dépendent. Il est important que la sécurité d'un protocole soit aussi indépendante que possible de la sécurité du système d'exploitation, en se séparant de tout élément de configuration ou tout composant logiciel qui y est lié et qui peut être altéré par une tierce partie. Une implémentation du protocole SVC doit suivre strictement cette règle et donc être auto-contenue, en incluant tous les algorithmes et éléments de configuration au niveau de l'application plutôt qu'au niveau du système d'exploitation.

1.3.2 Conception du protocole SVC

Le protocole SVC est conçu pour répondre à la problématique des communications internes (section 1.2.3). La plupart de ses éléments peuvent se présenter comme une version allégée de TLS et SSH. Ce protocole fournit cependant davantage de propriétés de sécurité, implémentant l'ensemble des propriétés de sécurité décrites dans la section 1 tout en ayant un impact minimal sur les coûts en ressources comparé aux autres protocoles.

Le protocole SVC fonctionne comme une couche intermédiaire entre la charge utile d'une application et le réseau effectif. Il n'impose pas de restrictions à l'utilisation sur la quantité de données. Les limitations liées aux algorithmes sous-jacents, comme les limites de volume qu'un algorithme de chiffrement peut traiter, sont gérées de façon transparente par le protocole. D'un autre côté, les limitations du réseau utilisé sont gérées dans la mesure du possible au niveau du protocole. En particulier les limitations sur les tailles de paquets, si applicable, sont prises en compte dans les opérations de chiffrement afin de profiter au mieux des ressources disponibles. Pour ces raisons, le protocole SVC fonctionne comme une encapsulation transparente de paquets, masquant les restrictions du réseau sous-jacent. Par conception, le protocole se situe au niveau application pour rester indépendant autant que possible des fonctions du système d'exploitation.

La seule contrainte du protocole SVC vis-à-vis du protocole d'authentification mis en œuvre est que ce dernier permette de réaliser une forme de signature

numérique. Cette signature est utilisée pour authentifier, en plus des éléments d'identification, les paramètres de sécurités du protocole. De par sa nature générique, la méthode d'authentification ne dépend pas de l'utilisation d'une PKI telle que c'est le cas pour le protocole TLS. Il est toutefois possible d'intégrer ce type de mécanisme d'authentification avec l'utilisation du protocole SVC. Cela permet d'intégrer l'utilisation du protocole SVC dans des applications existantes qui seraient basées sur un tel système.

L'étape de négociation, point central du protocole, apporte les garanties suivantes : l'existence d'un lien confidentiel entre les deux hôtes, l'authenticité du serveur, l'authenticité du client et la protection des informations d'identité du client. Les échanges suivants la négociation se placent donc dans un contexte sécurisé où les données sont correctement chiffrées et protégées contre toutes altérations. Les éléments de sécurité devant avoir une durée de vie restreinte (en particulier les clés de chiffrement) sont changées régulièrement à partir des paramètres de sécurité échangées pendant la négociation.

1.3.3 Établissement d'une connexion sécurisée

1.3.3.1 Négociation

Le protocole opère en deux phases : négociation et échange de messages sécurisés. La phase de négociation est illustrée dans la figure 1.2. Cette étape confirme en premier lieu la version du protocole utilisé par les deux hôtes. En cas de conflit de version, le protocole s'arrête. Ce cas ne se présente qu'en cas de défaut de mise à jour d'une des extrémités de l'échange, situation gérée au niveau applicatif. Le reste du premier envoi permet de définir les paramètres de sécurité à utiliser (les clés de chiffrements, les vecteurs d'initialisation,...). Ces paramètres de sécurité sont obtenus via un protocole d'échange de clés classique. Enfin, ce premier envoi permet aussi d'authentifier le message comme provenant d'un serveur autorisé. Le client ne répond que si tous les éléments sont validés. Il termine le protocole d'échange de clés, après quoi toutes les communications seront chiffrées et donc rendues confidentielles. Cela inclut les informations d'identité et d'authentification du client afin d'éviter la divulgation de ces informations. Le choix d'émettre le premier message par le serveur a été fait pour réduire la durée de la négociation sur certains réseaux à forte latence.

L'information d'identité du serveur et du client peuvent prendre une forme arbitraire, tel que requis par l'application en charge de l'authentification. Des exemples typiques d'informations d'identité sont des clés publiques (en forme brut ou sous forme de certificats), ou des identifiants de clé pré-enregistrées. Cette partie est spécifique à l'application ; le protocole SVC se contente de fournir un moyen de transfert sécurisé entre le client et le serveur.

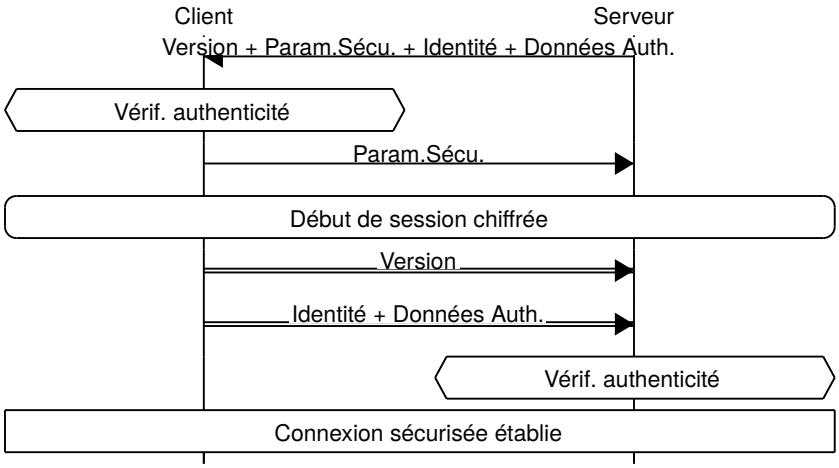

FIGURE 1.2 – Phase de négociation du protocole SVC

double flèche : contenu chiffré ; flèche simple : contenu en clair

La version du protocole sert à identifier à la fois le protocole et les algorithmes utilisés pour le chiffrement et l'intégrité. La première version du protocole SVC s'appuie sur un échange de clés basé sur le problème de Diffie-Hellman. Cet échange de clés fait partie des données authentifiées lors de la négociation. Lorsque l'échange de clés se conclut, les deux extrémités de la connexion sécurisée disposent d'un secret commun K_M. Ce secret commun est utilisé pour dériver la première clé de la session sécurisée K_1. Cette clé est utilisée pour mettre en œuvre le chiffrement. Le secret K_M est également utilisé pour générer les autres paramètres de sécurité comme des vecteurs d'initialisation ou des données d'authentification additionnelles.

1.3.3.2 Échange de messages sécurisé

Après l'étape de négociation, chaque extrémité de la communication peut envoyer des messages enrobant la charge utile, comme illustré par la figure 1.3. Du point de vue de l'application, les deux instances n'ont à se préoccuper que de la charge utile. Le protocole SVC se charge du chiffrement et de l'intégrité de l'échange. Le client et le serveur gardent tous les deux un compte du volume de données échangées. Dès que les limites imposées par la sélection d'algorithmes sont atteintes (en particulier sur le volume de donnés que l'on peut chiffrer sans changer les paramètres de sécurité), l'identifiant de la clé de session courante est incrémenté et une nouvelle clé de session K_i est calculée à partir du secret commun K_M. Dès que l'une des extrémités reçoit un message indiquant un nouvel identifiant de clé, cette nouvelle clé est utilisée pour toutes les communications ultérieures. La clé précédemment utilisée est définitivement mise au rebut.

Inst.A → SVC S	:	Envoi de la charge utile par l'application
SVC S → SVC R	:	Envoi de l'identifiant de clé
SVC S ⇒ SVC R	:	Envoi de la charge utile chiffrée et des informations d'intégrité
SVC R	:	Mise à jour des clés (si nécessaire)
SVC R → Inst.B	:	Transmission de la charge utile à l'application

FIGURE 1.3 – Enrobage de la charge utile d'une communication par SVC

SVC S et *SVC R* représentent les extrémités du protocole SVC. *Inst.A* et *Inst.B* représentent deux instances de l'application. Les doubles flèches indiquent un contenu chiffré.

La génération des clés de session est basé sur une fonction de hachage cryptographiquement sûre et est basé sur le secret commun K_M. La génération d'une clé de session d'identifiant i se fait en calculant $H\,(K_M||i)$ où H est la fonction de hachage et $||$ est l'opération de concaténation. Une méthode similaire est utilisée pour calculer les autres paramètres de sécurité à partir de K_M. L'utilisation d'une fonction de hachage cryptographiquement sûre, ainsi que d'une valeur initiale dépendant des deux hôtes permet de produire des paramètres difficiles à retrouver par un attaquant. De plus, cette approche permet de conserver un haut niveau de sécurité tout en réduisant considérablement l'impact d'un changement de clé sur la bande passante utilisée par les fonctions de sécurité du protocole. En effet, un identifiant de clé peut se limiter à un octet et toutefois servir à générer tout un ensemble de nouveaux paramètres.

Le renouvellement de la clé de session peut se produire après deux conditions : soit après l'écoulement d'une durée prédéterminée, soit après le chiffrement d'une certaine quantité de données, fixée par les algorithmes sous-jacents utilisés. Conserver les mêmes paramètres de sécurité après avoir chiffré ou déchiffré un volume de données trop important peut affaiblir la sécurité de certains algorithmes autrement sûrs[44].

1.3.4 Choix des algorithmes

Les algorithmes cryptographiques utilisés à différents niveaux doivent être choisis en fonction de leurs propriétés de sécurité et de leur résistance théorique sur le long terme. Le choix d'algorithmes est ici fixé avec la version du protocole. Nous parlons des algorithmes de chiffrement, d'échange de clés et d'intégrité, car le choix de l'algorithme d'authentification est délégué à l'application appelante. Cela contraste avec les approches classiques qui proposent une liste d'algorithmes et permettent au client (ou au serveur) de choisir dans cette liste ce qu'il supporte. Ce type de choix peut mener à des combinaisons non sûres basées sur des algorithmes sûrs par ailleurs, mais qui présentent des interactions problématiques. La

restriction à un jeu validé d'algorithmes permet d'une part d'éviter ces interactions malheureuses et d'autre part de simplifier la phase de négociation.

Contrairement à TLS et SSH, en se plaçant dans le cadre des communications internes nous pouvons ici nous permettre de fixer un choix et interrompre toute négociation qui ne le respecterait pas. En effet, en ayant la maîtrise sur les deux extrémités de la connexion, nous avons la certitude que le choix "imposé" sera accepté lors d'une négociation. De plus, l'utilisation d'un ensemble restreint d'algorithmes permet de faciliter leur intégration au niveau applicatif. Cette dépendance entre les mises à jours de l'application (contenant l'ensemble des implémentations de sécurité) et les protocoles de sécurité est une caractéristique acceptable uniquement dans le modèle des communications internes, contrairement au modèle des communications externes où client et serveur peuvent disposer de versions conflictuelles d'un protocole.

La première version du protocole SVC utilise les algorithmes suivants :

— ECDH [27][41] pour l'échange de clés
— AES-128/GCM[42] pour le chiffrement
— La fonction GHASH (composante du mode d'utilisation GCM [28]) pour l'intégrité et l'authenticité
— SHA2-256[1] pour la génération des paramètres de sécurité

Ces algorithmes ont été choisis pour plusieurs raisons. L'utilisation d'opérations sur courbes elliptiques permet d'avoir un niveau de sécurité comparable avec l'arithmétique classique tout en utilisant des éléments de taille inférieure : une courbe elliptique ayant une taille de corps de 512 bits apporte une sécurité comparable à une opération basée sur le logarithme discret utilisant un groupe de 15 360 bits tout en étant plus rapide. L'algorithme de chiffrement AES [29] est l'algorithme de chiffrement par bloc recommandé actuellement pour une taille de clé d'au moins 128 bits[4]. Le mode d'utilisation GCM est une extension du mode compteur réputé sûr permettant d'apporter des capacités d'authentification des échanges. La famille de fonctions de hachage SHA2 est elle aussi le standard actuel en matière de fonction de hachage sécurisée. Les tailles de clés et de blocs présentés ont été choisis selon les recommandations actuelles en matière de sécurité. L'ANSSI [30] conseille en effet pour un système cryptographique cohérent l'utilisation d'un chiffrement symétrique utilisant une clé de 128 bits, une taille

27. Elliptic-Curve Diffie-Hellman, Diffie-Hellman basé sur courbes elliptiques
28. Galois/Counter-Mode, mode de chiffrement par blocs intégrant un mécanisme d'authentification des données
29. Advanced Encryption Standard, algorithme de chiffrement par bloc le plus utilisé de nos jours
30. Agence Nationale de la Sécurité des Systèmes d'Information, chargée de proposer les règles à appliquer pour la protection des systèmes d'information de l'État et de vérifier l'application des mesures adoptées

de groupe de 256 bits pour les calculs sur courbes elliptiques et une fonction de hachage de 256 bits afin de fournir une sécurité fiable pour plusieurs années. Ces recommandations rejoignent celles émises par le NIST [31][16]. Ces choix d'algorithmes et de tailles de clés apportent un niveau de sécurité élevé sans impact significatif sur les performances du système.

Quand ces algorithmes (ou leur combinaison) feront l'objet de vulnérabilités effectives, il sera possible de spécifier une nouvelle version du protocole utilisant un autre jeu d'algorithmes. Pour le déploiement d'une nouvelle version du protocole SVC, il suffit alors de déployer une nouvelle version de l'application l'utilisant. Cela est possible parce que l'ensemble du protocole est contenu au niveau applicatif sans dépendance avec des composants externes ou le système d'exploitation.

Nous ne mentionnons pas dans cette spécification les algorithmes permettant de réaliser la gestion éventuelle des clés publiques et des signatures numériques. Ces éléments sont fournis par l'application et sont transférés de façon transparente par le protocole SVC entre le client et le serveur. Cela permet d'être compatible avec toute politique de sécurité déployée au niveau de l'application. La seule contrainte posée sur le système d'authentification et qu'il doit permettre de produire une signature numérique, c'est-à-dire fournir une fonction prenant en entrée les données d'identité (dont le format est lui aussi libre) et les données à authentifier et produire en sortie un élément pouvant permettre de vérifier a posteriori l'authenticité. Après que l'application ait fourni au protocole SVC une telle fonction et les informations d'identité adéquates, le protocole se charge d'encapsuler ces éléments dans sa phase de négociation. Il apporte automatiquement la confidentialité sur l'ensemble des éléments identifiant du client et vérifie de façon transparente l'authenticité de la négociation. Un effet de bord positif de cette approche est qu'il est possible d'intégrer au protocole SVC une architecture d'authentification basée sur une PKI, y compris celle utilisée par TLS. Même si cette approche n'est pas recommandée, cela peut s'avérer nécessaire en termes d'utilisabilité pour intégrer le protocole SVC dans des applications existantes.

1.4 Comportement du protocole

1.4.1 Performances

1.4.1.1 Coût théorique en ressources

Le surcoût en ressources du protocole SVC par rapport à des communications non sécurisées est minimal à la fois en termes de bande passante et de nombre

31. National Institute of Standards and Technology, institut Américain faisant référence notamment dans le domaine des algorithmes cryptographiques

d'échanges. Pour une exécution normale, un seul aller/retour est nécessaire pour compléter la négociation. Le volume nécessaire pour effectuer cet aller/retour varie selon la modalité d'authentification utilisée. Les échanges suivants ont un surcoût en bande-passante fixe. Chaque message échangé se voit augmenté de trois éléments : un identifiant de clé (un octet), la position du paquet (trois octets), et un élément de contrôle d'intégrité (HMAC). L'algorithme GCM permet d'obtenir un élément de contrôle d'intégrité de taille variable. L'utilisation de paramètres de sécurité appropriés permet d'utiliser une taille de 12 octets pour cet élément ; il s'agit de la plus petite taille proposée pour l'utilisation du contrôle d'intégrité du GCM ne mettant pas en péril la confiance de cet élément[44]. Dans ces conditions, le surcoût en bande-passante se limite à 16 octets, en suivant le modèle de la figure 1.4.

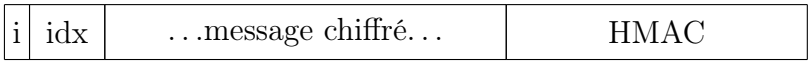

FIGURE 1.4 – Constitution d'un message du protocole SVC

L'étape de négociation est aussi compacte que possible. L'échange de clés est basé sur des courbes elliptiques, ce qui permet de réduire la bande passante nécessaire en conservant le même niveau de sécurité que les approches classiques. La terminaison de la négociation en un seul échange la rend efficace sur des réseaux à forte latence. Une autre conséquence de cette négociation rapide est que la renégociation sur des réseaux avec de fortes pertes de paquets ou des déconnexions régulières, est acceptable. En effet, en considérant un cas extrêmement défavorable ou seule deux échanges peuvent avoir lieu avant une perte de connexion, le deuxième échange permettra malgré tout de communiquer de façon sécurisée.

1.4.1.2 Cas d'usages

Certains éléments du protocole sont de tailles variables : tout ce qui est lié à la modalité d'authentification impacte la taille des éléments d'identité et d'authentification. Afin de pouvoir comparer les performances du protocole SVC avec les protocoles classiques que sont TLS et SSH, il faut fixer la taille de ces données. Nous développons ici deux exemples utilisant deux modalités d'authentification différentes. La première présente un cas simple de communication entre deux entités inconnues, le deuxième sert de point de comparaison avec les protocoles existants. Ces deux cas d'usages se concentrent sur l'impact des protocoles de sécurités présentés dans ce chapitre (TLS, SSH et SVC). Ils ne présentent pas immédiatement de cas d'utilisation concrets, mais deux situations permettant de couvrir de nombreux usages (communications client/serveur, communications machine à machine, etc).

Authentification a clé publique : Le premier exemple pourra servir à la mise en place d'un échange ponctuel entre deux entités proches (tels que deux smartphones). Pour cet exemple, nous considérons un système d'authentification à clé publique utilisant ECDSA [32] basé sur la courbe $P-521$ du NIST. Ici, ni le serveur ni le client ne se connaissent. La validation des clés peut avoir lieu de manière visuelle, avec l'affichage d'une empreinte comparable sur chacun des appareils. La taille d'une clé publique dans ce système est d'environ 150 octets et les signatures produites sont d'environ 130 octets. Afin de transmettre ces éléments de taille variable de façon contrôlée, nous utilisons un encodage simple ou leur longueur est elle-même codée sur un entier non signé de deux octets. Pour des raisons de clarté, cet exemple utilise la même courbe elliptique pour l'échange de clés. Au total, la négociation nécessite pour chaque entité : un octet pour le numéro de version, 150 octets pour l'échange de clés (ECDH), 150 octets d'information d'identité (la clé publique), et 130 octets d'authentification, plus trois fois deux octets pour la taille des champs de taille variable, soit 437 octets. La négociation complète opère donc en un aller-retour et 874 octets. Ce volume de données est adapté pour de nombreux types de réseaux, y compris des réseaux à faible capacité tel qu'un réseau de type Bluetooth, sur lequel la valeur de MTU [33] courante est de 672 octets. Sur ce genre de réseaux, chaque message de la négociation pourra être transmis en une seule fois et sera moins sujet aux problèmes de retransmission ou de congestion. Évidemment, la même chose reste vraie pour des réseaux moins limités.

Comparaison avec les protocoles existants : Afin de pouvoir comparer la négociation du protocole SVC avec les protocoles SSH et TLS, il faut définir une modalité d'authentification similaire. En ce qui concerne la négociation de SSH, effectuer la comparaison avec un système à clé publique comme décrit dans le paragraphe précédent est acceptable. En effet, SSH utilise une authentification à clé publique pour authentifier le serveur et l'utilisation d'une clé publique côté client est aussi possible. Pour permettre une comparaison avec TLS, il est plus raisonnable de considérer une authentification du serveur avec un certificat X.509. À titre d'exemple, un certificat X.509 contenant une clé RSA [34] 2048 bits fait environ 1100 octets. Il s'agit du type de certificat couramment utilisé de nos jours. Il est intéressant de noter que même si les certificats X.509 utilisés par TLS peuvent supporter une authentification basée sur courbes elliptiques, leur utilisation n'est pas encore répandue. L'authentification du client par le protocole TLS pouvant utiliser des moyens très variés et nécessitant le support d'outils spécifiques complexes, la com-

32. Elliptic-Curve Digital Signature Algorithm, signatures numériques basées sur courbes elliptiques

33. Maximum Transmission Unit, taille maximum d'un paquet transmis sur un réseau donné

34. Rivest Shamir Adleman, algorithme de chiffrement a clé publique parmis les plus utilisés au monde

paraison est ici faite sans cette étape. Dans la mesure où le protocole TLS présente un coût en ressources plus important malgré l'absence d'authentification client, le résultat de cette comparaison reste pertinent. Pour permettre de comparer avec le protocole SVC, il faut toutefois noter que même si l'authentification est basée sur RSA 2048, le protocole d'échange de clés utilisé par TLS dans le tableau 1.5 est lui aussi basé sur ECDH. Pour résumer, nous avons deux scénarios de comparaisons :

— Comparaison avec SSH : authentification client et serveur via leurs clés publiques respectives basées sur courbe elliptique
— Comparaison avec TLS : authentification serveur uniquement via un certificat X.509

Le tableau 1.5 résume les différences au niveau de la négociation entre les trois protocoles. Il y a un net avantage pour le protocole SVC, que ce soit en nombre de messages échangés ou en bande passante nécessaire. La conséquence est que dans le cas de communications utilisant un support ayant une qualité de service faible (telle qu'un réseau mobile), les processus de négociations nécessitant plus d'échanges et plus de bandes passantes sont susceptibles d'échouer avant d'avoir pu se terminer.

Protocole	Messages échangés	Taille totale (o)
Auth. par clé publique		
SSH	16	6383
SVC	2	880
Auth. par certificats X.509		
TLS	6	5244
SVC	2	1670

TABLE 1.5 – Comparaisons entre SVC, SSH et TLS

Valeurs expérimentales obtenues après plusieurs mesures des étapes de négociations.

1.4.2 Analyse qualitative

1.4.2.1 Réseaux fortements perturbés

Une des contraintes prise en compte lors de la conception du protocole SVC a été d'améliorer le comportement des connexions sécurisées aussi bien sur des réseaux performants que sur des réseaux limités. Par réseaux performants, on considère des réseaux sur lesquels les paramètres de latence, perte de paquets et bande passantes sont optimums. C'est par exemple le cas d'une connexion Ethernet directe, sur laquelle la latence et la perte de paquets sont minimums, alors que la

bande passante est importante. Sur ces réseaux, il y a peu de difficultés à établir des communications fiables ; dans ces conditions, le protocole SVC, comme d'autres protocoles, se comportent correctement et n'apportent pas de pénalités aux communications.

Cependant, nous considérons également la présence de réseaux moins performants comme des réseaux mobiles ou des interconnexions moins efficaces. Sur ces types de réseaux, les trois paramètres considérés peuvent se retrouver dans une situation inverse de la situation parfaite : latence importante, forte perte de paquets et bande passante limitée. Parfois ces paramètres fluctuent sur une même connexion. C'est dans ces conditions que les protocoles de communication sécurisées ont un impact important sur la qualité de la connexion. En effet, les protocoles de communication sécurisées ont tous un surcoût d'utilisation du réseau que l'on peut mesurer par deux paramètres : le nombre d'échanges supplémentaires induits par le protocole, et la bande passante supplémentaire.

Parmi les réseaux que l'on peut trouver, un exemple typique de réseau sur lequel la qualité de service est fluctuante est les réseaux mobiles (de type 3G/4G LTE [35]). Ces réseaux peuvent avoir de très bonnes performances (par exemple, une bande passante maximum théorique de 100 Mbit/s pour un réseau LTE), mais sont soumis a des variations importantes car perturbés par de nombreux facteurs extérieurs : nombre d'utilisateurs, bruits radio, qualité de signal, etc. Dans les plus mauvais cas, la perte de paquets constatés sur la connexion mobile augmente considérablement, causant une réduction importante de la bande passante et une augmentation de la latence.

Dans le cas du protocole SVC, le surcoût réseau induit par le protocole est réduit au minimum permettant de maintenir une connexion sécurisée. La phase de négociation impose un aller-retour supplémentaire initial, et chaque message transmis a un surcoût minimal comme indiqué au paragraphe précédent (section 1.4.1). Sur un réseau perturbé, cela signifie que la phase de négociation s'achève rapidement, permettant à l'application de maximiser l'utilisation des ressources limitées du réseau.

La considération des réseaux perturbés est essentielle, car l'interconnexion de réseaux de natures différentes est de plus en plus importante : un cas courant est l'utilisation d'une connexion mobile pour communiquer avec Internet, afin de joindre un réseau privé performants. Ce type de scénario nécessite des échanges pouvant traverser sans difficultés les trois types de réseaux.

35. Long Term Evolution, norme évolutive des réseaux de communications mobiles

1.4.2.2 Protocoles réseaux et perturbations

Le comportement des différents protocoles, qu'il s'agisse de protocoles de communications ou autre, sur des réseaux limités n'est pas un problème nouveau. En particulier deux aspects important doivent être préalablement décrit afin de pouvoir interpréter les résultats expérimentaux de ce paragraphe.

Le protocole le plus couramment utilisé aux dessus des différents réseaux physiques est le protocole TCP. Ce dernier apporte de façon transparente (vis-à-vis des protocoles supérieures) des propriétés de fiabilités de la connexion avec entre autre un mécanisme de retransmission de paquets perdus et de contrôle d'ordre de séquence. Cet aspect est important, car sur un réseau faiblement perturbé, le protocole TCP permet de masquer les petites fluctuations de qualité du réseau aux applications. En particulier, une perte de paquets ponctuelles entraînera simplement une retransmission, réduisant temporairement la bande passante du réseau et augmentant la latence, mais sans impacter davantage la communication. Il existe cependant un point de rupture à partir duquel le protocole aggrave la situation : lorsque la perte de paquets est trop importante, le protocole TCP entraîne des retransmissions de plus en plus lentes des paquets afin de parvenir à rétablir la communication. Cette retransmission peut augmenter la latence de plusieurs secondes ; c'est une fois ce seuil franchis que le protocole TCP dégrade la communication. En effet, si une perte de paquet ponctuelle s'achève après une longue période de perturbation, la connexion TCP a beaucoup de mal à se rétablir, en particulier si une quantité importante de paquets est en attente.

L'autre aspect important des protocoles mis en jeu est le délai avant abandon d'une connexion, et est partiellement lié au comportement du protocole TCP. En effet, ce délai d'expiration est un paramètre important pour les serveurs : plus il est élevé, plus un serveur doit conserver longtemps les ressources allouées à une connexion. Au contraire, un délai d'expiration bas permet de libérer rapidement les ressources. Ce réglage est typiquement d'une ou deux secondes pour un serveur accessible depuis Internet. Combiné avec un délai de retransmission TCP important, cela signifie que la communication depuis un réseau perturbé vers un serveur de ce type est régulièrement abandonnée, nécessitant de fréquentes renégociation d'un éventuel protocole sécurisé utilisé.

Ces deux éléments, conçus pour augmenter l'efficacité des réseaux et des systèmes, s'associe mal avec le surcoût des protocoles de communications. Ces derniers monopolisent le début de toute connexion pour réaliser leur négociation. Si une application utilisant TCP via un réseau fortement perturbé doit attribuer cette fenêtre "utile" pour la négociation de sécurité, elle ne peut plus communiquer.

Afin d'évaluer l'impact des perturbations réseaux sur le comportement des communications sécurisées, un scénario de test a été mis en place. Une application tente de communiquer un fichier au travers d'un réseau présentant des perturba-

tions contrôlées. Le protocole de l'application est très simple, et opère au-dessus d'un protocole de sécurité ou d'une communication non sécurisée servant de témoin. Lors de l'envoi du fichier le délai d'expiration réglé sur le serveur est d'une seconde. En cas de dépassement, la connexion est coupée et nécessite l'établissement d'une nouvelle connexion sécurisée.

Cette évaluation a été réalisée sur un réseau maîtrisé sur lequel les paramètres de bande-passante, et latence et de perte de paquets peuvent être ajustés. On ne présente ici que les résultats obtenus en faisant varier le taux de perte de paquets. En effet, la fluctuation des autres paramètres n'exhibe pas de comportement intéressants. En particulier, l'augmentation de la latence n'entraîne effectivement qu'une réduction de la bande passante causée par le mécanisme d'acquittement, jusqu'au point où la latence dépasse le délai d'expiration rendant impossible l'établissement d'une connexion. De même, la fluctuation de la bande passante ne fait qu'augmenter le temps de transmission de façon linéaire. La variation du taux de perte de paquets se fait via un mécanisme statistique simulant les séquences de pertes. Concrètement, un taux de perte de 20 % ne signifie pas que chaque paquet présente un risque de disparition de 20 % ; le risque de perte de chaque paquet individuel est pondéré par le statut du paquet précédent. Ce système apporte sur le long terme un taux de perte moyen effectif de 20 %, mais présente donc des séquences consécutives de pertes de paquets associés à des séquences de non pertes, s'approchant plus du comportement attendu sur réseaux mobiles.

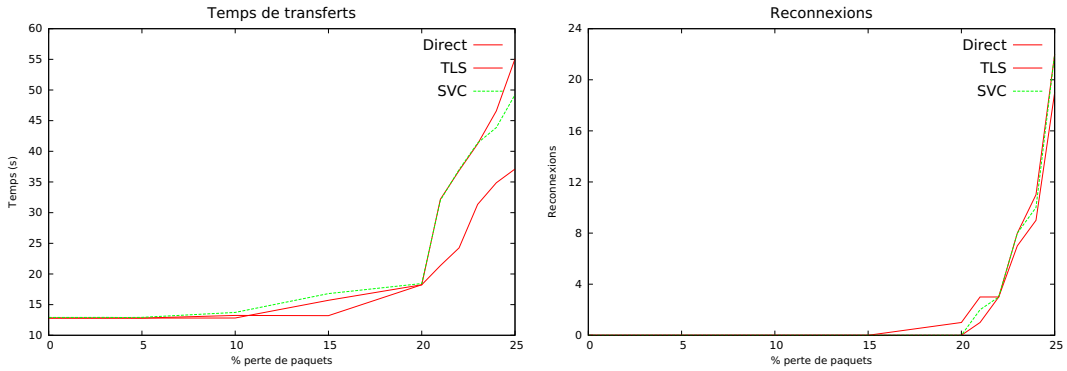

FIGURE 1.5 – Protocoles de communication sécurisés sur réseaux perturbés

L'observation de la perte de paquets est plus pertinente, car elle cause l'établissement de nouvelles connexions tout en étant un cas réaliste observé. La figure 1.5 résume les résultats obtenus lors de cette évaluation. Elle compare le temps de transfert total et le nombre de reconnexions pour trois types de communications : directe (non sécurisée), TLS et SVC en fonction du taux de pertes de paquet du réseau. On y observe trois phénomènes : la présence d'un seuil séparant deux comportements, le faible impact des protocoles de communication sécurisés

et l'importance d'une négociation rapide.

Seuil de pertes acceptable Tout d'abord, en dessous de 20 % de pertes de paquets, l'utilisation du protocole TCP masque totalement les pertes au protocole applicatif. Cela signifie que jusqu'à ce seuil, les délais de retransmission restent assez faible pour ne pas dépasser le délai d'expiration d'une connexion. En dessous de ce seuil on observe donc une légère augmentation du temps total de transmission lié à la diminution temporaire de la bande passante, Au-delà de ce seuil, on commence à observer des pertes de connexions. Ces pertes provoquent des reconnexions et nécessitent de nouvelles négociations du protocole de communication sécurisés. Ces négociations répétées agissent en cascade. Plus elles sont nombreuses, plus elles monopolisent la disponibilité du réseau, augmentant en conséquence le temps de transfert. Au-delà des 20 % de pertes de paquets, l'efficacité de la connexion diminue donc rapidement.

Impact des communications sécurisées Le temps de transfert dans la situation idéale sans perte et jusqu'au seuil observé de 20 % est similaire que l'on utilise un protocole de communication sécurisés ou non. Cela indique clairement que ces protocoles (SVC et TLS) ont une vitesse de traitement suffisante pour ne pas pénaliser sensiblement les temps de transferts en situation normale. Cependant, la situation au-delà du seuil de 20 % indique une dégradation importante de l'efficacité du transfert. Cette dégradation est amplifiée par l'utilisation de protocoles de communication sécurisés. En effet, le surcoût principal de ces protocoles se produit durant leur phase de négociation et cette dernière est répétée régulièrement sur réseau perturbé.

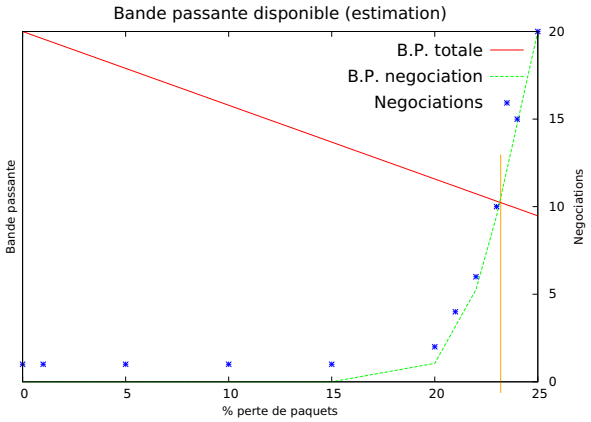

FIGURE 1.6 – Visualisation de l'impact des négociations répétées

B.P. totale représente la bande passante théorique du réseau considéré alors que B.P. negociation représente la bande passante monopolisée par les négociations répétées

Temps de négociation La conclusion de ce test est la suivante : le délai de négociation, multiplié par le nombre de reconnexions imposés par le réseau, impacte considérablement une communication sécurisée lorsque le réseau utilisé est perturbé. Cet impact est illustré sur la figure 1.6 où l'on voit qu'à partir d'un certain seuil, le surcoût des négociations répétées dépasse la bande passante disponible. La figure présente un cas général ; ce seuil varie selon trop de paramètres pour être représenté en un seul graphique. Par exemple, dans le cas considéré dans cette section, ce seuil critique est situé à 25 %. En dessous de 20 %, la communication n'est que faiblement perturbée par les négociations. Entre 20 % et 25 %, le temps de transfert augmente considérablement, et au-delà la communication "utile" ne passe quasiment plus. Cette limite dépend de la bande passante du réseau, du coût de la négociation, de la durée des périodes perturbées du réseau, de la latence habituelle qui s'ajoute à la latence des paquets retransmis, etc. Au niveau des protocoles de communication sécurisés, la réduction du coût de la négociation permet de repousser ce seuil.

La confrontation de ces résultats expérimentaux avec des caractéristiques de réseau réelles est compliquée. En effet, il existe peu d'information publics fiables permettant d'évaluer les performances des réseaux mobiles dans différentes situations, soit parce que ces données sont difficilement quantifiables, soit parce qu'il existe une volonté de ne pas diffuser cette information. Une expérimentation ponctuelle sur le terrain n'apporterait pas de données pertinentes : en plus de ne disposer que d'un faible échantillon, les mesure en parallèle des performances de communications et de la qualité de service d'un réseau à un moment donné se perturbent mutuellement. De même, les qualités du réseau pouvant fluctuer selon de nombreux paramètres non maîtrisés, effectuer ces mesures indépendamment ne serait pas non plus pertinent. C'est pour ces raisons que l'expérimentation a finalement été réalisée sur un réseau maîtrisé, permettant non pas d'observer les taux de pertes, mais de les définir. Nous avons ainsi pu mesurer l'efficacité du protocole en faisant varier les perturbations réseaux et obtenir des résultats qui pourront plus tard être mis en regard de mesures sur le terrain.

1.4.3 Analyse de sécurité

L'utilisation d'algorithmes connus comme fiables et de bonnes pratiques en ce qui concerne leur combinaison permet d'assurer que les opérations de chiffrement sont sûres. De plus, la mise en place du chiffrement avant la transmission d'informations sensibles par le protocole permet de protéger l'identité des utilisateurs. L'ensemble du système s'appuie sur la fiabilité du processus d'authentification, fourni par l'application appelante. Cette section traite des attaques courantes sur les protocoles de communication sécurisé rendues impossibles pour le protocole SVC de par sa construction.

1.4.3.1 Fuite d'information par retour d'erreur

Pour le protocole SVC, lorsqu'une erreur se produit, qu'il s'agisse d'un problème d'intégrité, d'identifiant invalide ou autre, il n'y a pas de notification de la cause exacte de l'erreur. L'absence de détails à ce niveau est intentionnel pour éviter toute fuite d'information. En effet, indiquer une erreur comme l'impossibilité de valider l'intégrité d'un paquet peut permettre de fabriquer de faux messages qui seraient alors considérés comme "valides"[44]. C'est pour cela que le protocole ne propose pas de mécanisme de récupération en cas d'erreur. Pour reprendre la communication, il faut établir une nouvelle connexion sécurisée. Cette approche présente l'avantage que chaque connexion utilise de nouveaux paramètres de sécurité et réduit fortement le risque de manipulation malveillante. Dans la mesure où l'étape de négociation du protocole est très rapide, même sur des réseaux de faible capacité, cette solution est acceptable.

1.4.3.2 Man in the Middle (homme au milieu)

Une attaque de type MitM est une attaque dans laquelle un attaquant s'insère entre deux entités souhaitant communiquer et peut observer et modifier le contenu des échanges. Le cœur du protocole SVC est l'algorithme d'échange de clés Diffie-Hellman[55]. Cet algorithme est vulnérable aux attaques MitM. En effet, dans la version classique, il n'y a pas d'élément d'authentification. Si l'algorithme Diffie-Hellman permet d'échanger une clé entre deux entités α et β sans en garantir l'authenticité, un attaquant γ dans une position de MitM peut alors tromper α et β en établissant deux échanges de clé : un entre α et γ, un autre entre γ et β. De cette façon il pourra intercepter de façon transparente toutes les communications entre α et β.

Dans le cas du protocole SVC, l'échange de clés est lui-même authentifié. En supposant un mécanisme d'authentification sûre (l'utilisation de cryptographie à clé publiques des cas d'usages précédents), l'algorithme Diffie-Hellman ne peut aboutir qu'avec une information d'authentification valide, ce qu'un attaquant ne peut pas reproduire. Le pire scénario dans le cas d'un MitM est celui d'un DoS dans lequel l'attaquant déclencherait un grand nombre de négociations. Ce genre d'attaque peut être détecté au niveau applicatif, car l'étape de négociation doit pouvoir se terminer en un aller-retour. Le problème se contourne en limitant le nombre de tentative de négociation qu'un client donné peut réaliser sur une courte période.

1.4.3.3 Rejeu

Il est parfois possible pour un attaquant d'enregistrer un échange à un moment donné et de construire une nouvelle "négociation" basée sur cet enregistrement,

afin de prendre la place du serveur ou du client. Lorsque ce genre d'attaque est utilisé pour prendre la place d'un client, l'attaquant peut usurper l'identité d'un utilisateur, identité obtenue lors d'une précédente session. Dans le cas de l'usurpation du serveur, cela permet de tromper les clients potentiels en leur faisant croire qu'ils communiquent avec une entité valide et peut mener à la fuite d'informations sensibles.

Ce type d'attaque n'est pas possible lors de l'utilisation du protocole SVC, l'échange de clés est authentifié durant la première étape de la négociation et ne se termine que si le client peut authentifier avec certitude le serveur. Cette authentification, combiné avec l'utilisation de l'échange de clés Diffie-Hellman, fait que même si un attaquant pouvait se placer dans la position du serveur et rejouer une négociation précédente, les informations sensibles (l'identité du client et les échanges ultérieures) restent indéchiffrables. Un attaquant ne peut donc pas usurper la position du serveur ou du client. Le risque le plus élevé ici est l'envoi par le client d'informations d'identité à un usurpateur. Cet envoi serait déjà chiffré et ne présente donc pas un risque majeur.

1.4.3.4 Récupération de clé de sessions

Les clés de session sont des clés cryptographiques utilisées pour le chiffrement effectif des données lors d'une communication sécurisée. Il s'agit de clés éphémères qui sont obtenues au terme de la phase de négociation. Elles ne sont liées à aucun élément permanent conservé par le client ou le serveur au-delà de leur utilisation au cours d'une communication.

Cependant, l'utilisation d'une même clé durant de longues périodes augmente le risque d'une récupération par un attaquant, en particulier si ce dernier dispose d'informations sur la nature du contenu échangé. Au minimum, une attaque par force brute sur une communication enregistrée "hors-ligne" est toujours possible. Quelle que soit la méthode employée, si un attaquant parvient à récupérer une clé de session, l'ensemble des données (passées et futures) chiffrées avec cette clé sont compromises.

Pour réduire l'impact de ce risque, les clés de session utilisées par le protocole SVC sont renouvelées régulièrement en fonction de deux critères : quand le volume de données traités dépasse un seuil acceptable pour une seule clé, ou lorsqu'un délai trop long s'est écoulé. Le changement de clé lors du dépassement d'un volume de données réduit l'efficacité d'une attaque sur ce jeu de données. En effet, la récupération d'une clé est une opération extrêmement coûteuse. Une expérimentation récente[13] présente l'utilisation de matériel hautement parallèle pour réaliser une attaque en force brute sur différents algorithmes de chiffrement, dont AES en effectuant 560 000 tentatives par secondes, soit une part négligeable des $2^{1}28$ clés possibles pour AES-128. Bien qu'encore non applicable de façon efficace, les

performances de ces systèmes et leur nombre ne va faire que croître, rendant réaliste la compromission de ce chiffrement par des attaquants disposant de moyens conséquents. C'est pour cela que réduire l'efficacité de cette attaque est un enjeu important ; le temps de calcul nécessaire à la récupération d'une clé est multiplié par autant de clé différentes utilisées tout au long de la communication. Pour des raisons similaires, les clés ne sont pas conservées au-delà d'un temps donné, afin de protéger les communications ultérieures.

Dans le cas du protocole SVC, le changement de clé ne nécessite pas de communication préalable entre les deux entités. Seul le changement de l'identifiant de clé, situé en début de message, est nécessaire. Cette approche permet de changer de clé aussi souvent que nécessaire en minimisant l'impact du protocole sur l'utilisation réseau.

1.5 Perspectives et recommandations

Une même application peut nécessiter les deux types de communication décrits dans la section 1.2. En effet, il n'est pas rare de déployer dans un même système une partie interopérable, communiquant avec des applications extérieures et une partie interne, permettant des échanges et des synchronisations entre différentes instances d'une application. C'est sur ce modèle que fonctionnent de nombreux services distribués. L'objet du protocole SVC n'est pas de remplacer les protocoles existants mais d'attirer l'attention sur le besoin mal satisfait des communications internes en y apportant une réponse adaptée. Une application ayant un besoin fort d'interopérabilité aura toujours la nécessité de s'appuyer sur des protocoles dédiés tels que TLS. C'est pourquoi il est important d'attirer l'attention sur les faiblesses de tels protocoles et d'apporter une solution adaptée pour les autres cas.

La version actuelle du protocole SVC a été conçue pour répondre à des problématiques strictes et réalistes pour le modèle des communications internes. Il reste certains points qui peuvent donner lieu à des améliorations sensibles du protocole, tout en restant dans l'espace de contraintes imposées. Parmi ces points, nous pouvons citer un double système d'authentification qui permettrait d'authentifier à la fois l'utilisateur et son terminal de façon efficace, ainsi que l'ajout de mécanismes de compressions pour optimiser l'utilisation de la bande passante.

Le double système d'authentification permettrait d'alléger la tâche des protocoles d'utilisation en les intégrant davantage dans le processus de négociation d'une communication sécurisée. Cette intégration permettrait au protocole d'authentification de transmettre en toute sécurité des informations sensibles, sans avoir à établir un second canal sécurisé. Pour réaliser cela, il faudrait effectuer des changements dans le déroulement du protocole SVC. Il est probable qu'il faudrait davantage d'échanges, mais surtout il deviendrait alors nécessaire de disposer d'une

authentification du terminal “client”. Cette première authentification serait indépendante de l'utilisateur et permettrait de certifier un appareil pour une utilisation sécurisée.

L'ajout de mécanismes de compression ou de traitement de l'information permettrait de réduire efficacement le coût en bande passante sans compliquer le développement de l'application appelante. En effet, un mécanisme de compression est plus efficace sur les données non chiffrées, ce qui signifie qu'il est plus intéressant de l'appliquer avant la transmission par le protocole SVC. Cependant, effectuer cette compression avant de transmettre les données au protocole de communication sécurisée signifie que les tests d'intégrité utilisés par ce dernier s'appliquent à une donnée transformée, ce qui peut introduire des redondances avec un test d'intégrité lié à l'algorithme de compression. De plus, l'intégration de mesures visant à réduire l'utilisation de la bande passante par les communications est intéressante à réaliser dans un protocole dédié aux communications, plutôt que dans le cœur d'une application métier. Afin de répondre à ces questions d'efficacité tout en améliorant le protocole SVC, il serait donc intéressant d'étudier quels algorithmes de compression peuvent s'appliquer dans cette situation.

Pour conclure ce chapitre il est important de noter qu'une partie de la sécurité du protocole SVC s'appuie sur l'utilisation de modalités d'authentification fiable, pour authentifier le côté serveur comme le côté client. Les cas d'usages proposés sont basés sur des problèmes pratiques variés, couvrant à la fois un usage de proximité et une utilisation à grande échelle. Dans le deuxième cas il s'agit de systèmes nécessitant une mise en place préalable d'éléments spécifiques (comme la dissémination de certificats de clé publique). Il est souhaitable de disposer de moyens d'authentification plus variés, tout en conservant la contrainte de compatibilité avec des systèmes cryptographiques. Le chapitre suivant présente une solution permettant d'utiliser des modalités d'authentifications variées avec un système cryptographique.

2 AUTHENTIFICATION

Dans le chapitre précédent, nous avons présenté un protocole permettant d'établir des connexions sécurisées entre systèmes communicants. Un élément de ce protocole est l'authentification qui permet de s'assurer que la communication s'établit entre entités de confiance. Dans ce cas, l'authentification s'appuie sur un mécanisme extérieur capable de garantir le processus d'authentification sans avoir à se soucier de la protection de l'échange permettant d'obtenir un résultat. Nous allons maintenant voir en détail en quoi consiste une authentification et quelles sont les modalités utilisables pour la réaliser. Nous présentons ensuite deux nouvelles approches permettant de réaliser une authentification distante de façon fiable et d'utiliser des outils cryptographiques basé sur cette authentification.

Le processus d'authentification est la première étape permettant d'utiliser un système sécurisé. L'authentification fait suite à l'identification. Ces deux éléments sont souvent confondus pour des raisons pratiques. L'identification permet à une entité d'annoncer son identité, alors que l'authentification permet de confirmer de façon irréfutable cette identification. Cela a pour but de garantir que l'entité authentifiée n'est pas remplacée par un usurpateur. Une fois une entité authentifiée sur un système sécurisé, des propriétés comme une liste d'autorisations ou des droits particuliers peuvent lui être attribuées.

Il est possible d'authentifier différents types d'entités en utilisant différentes méthodes. Parmi les éléments qui peuvent être identifiés se trouvent notamment des utilisateurs et des machines. L'authentification des machines permet de s'assurer que seuls des systèmes contrôlés et validés peuvent être utilisés notamment avec des applications distantes ou des applications distribuées. Au contraire, lorsque ce sont des utilisateurs qui sont authentifiés, la machine utilisée n'est qu'un moyen permettant l'accès aux ressources sécurisées ; ces ressources pouvant se trouver sur la machine en question, ou sur un système distant. Ces deux approches peuvent être combinées dans le cas d'une application distante, pour s'assurer que la machine utilisée est conforme et pour assurer par la suite que l'utilisateur dispose des autorisations d'accès nécessaires.

Le processus d'authentification peut être réalisé à partir de diverses modalités : utilisation d'un mot de passe, signature par un système de clé publique, mise

en œuvre d'un composant matériel (jeton cryptographique), biométrie, etc. Ces modalités d'authentification peuvent être utilisées séparément ou être combinées afin d'augmenter la fiabilité du processus ; il s'agit alors d'authentification multimodale. Chaque solution présente ses avantages et inconvénients ; cependant une distinction importante existe et permet de les séparer en deux groupes : les solutions permettant d'obtenir un élément secret cryptographique et les autres. L'objectif de ce chapitre est de proposer un protocole d'authentification permettant d'utiliser les modalités d'authentification pour fournir un secret cryptographique. Ce dernier est alors utilisé par les différents outils cryptographiques du système.

Qu'il s'agisse d'une authentification locale ou d'une authentification vis-à-vis d'un système distant, l'authentification est un processus qui pour une identité et une modalité donnée, produit un résultat binaire indiquant un succès ou un échec. En ne considérant que l'objectif initial qui est de valider une identité, ce type de résultat est suffisant. Cependant, l'utilisation d'un système mettant en œuvre des outils cryptographiques nécessite des clés et autres éléments secrets pour fonctionner. Si ces éléments doivent être liés à l'utilisateur l'authentification ne peut alors plus se limiter à un simple résultat binaire mais doit aussi fournir ces éléments secrets propres à l'utilisateur. Ainsi, l'authentification sert tout au long de l'utilisation du système sécurisée et non plus seulement comme moyen d'accès initial.

Nous nous intéressons à l'authentification des utilisateurs sur des systèmes distants. L'étape d'authentification est le premier moment où un utilisateur doit prouver son identité. Le succès de cette étape marque le début d'une "session" pendant laquelle l'utilisateur peut accéder au système. Dans le cas d'un système sécurisé mettant en œuvre des outils cryptographiques côté client, des secrets cryptographiques peuvent être utilisés tout au long d'une telle session. C'est le processus d'authentification, lien entre l'utilisateur et son identité, qui doit fournir ces secrets, contrairement aux approches basés uniquement sur du contrôle d'accès pour lesquelles une simple vérification par le système distant est réalisée.

Dans ce chapitre, nous verrons tout d'abord comment fonctionnent des solutions courantes d'authentification, ainsi que comment différentes modalités d'authentification peuvent être classées en deux groupes, séparant ainsi les modalités pouvant fournir un secret cryptographique des modalités ne le permettant pas. La suite du chapitre apporte deux solutions au problème de l'utilisation d'outils cryptographiques dans un système protégé par une authentification. Tout d'abord, nous proposons un mécanisme de gestion d'un "trousseau" de clés sécurisé, en particulier des méthodes permettant son transfert sécurisé afin d'être toujours disponible lorsqu'un utilisateur s'authentifie. Ensuite nous proposons un nouveau protocole d'authentification permettant l'utilisation de modalités d'authentification diverses de façon générique afin de fournir à la fois une confirmation d'authentification à

un système distant et le support nécessaire à l'utilisation d'outils cryptographiques côté client. Enfin, nous verrons un exemple concrèt sous la forme d'un prototype d'application. Ce prototype a été réalisé en collaboration avec la société SESIN afin de répondre à un besoin précis : l'utilisation de Cartes de Professionnels de Santé (CPS) pour authentifier les utilisateurs. La solution appliquée ici met en œuvre le protocole d'authentification décrit dans ce chapitre.

2.1 Méthodes d'authentification existantes

Il existe de nombreux outils et méthodes permettant d'authentifier des utilisateurs sur différents systèmes. Nous n'allons pas évoquer ici les solutions permettant des authentifications locales, mais uniquement des solutions permettant d'authentifier un utilisateur sur des systèmes distants. De même, les approches non basées sur des outils cryptographiques ne seront pas évoquées dans cette section, car ces dernières n'apportent pas les garanties de résultat et de confidentialité que l'on peut attendre d'un système sécurisé. Dans cet état de l'art, nous faisons la différence entre les modalités d'authentification et les protocoles d'authentification. Les premières sont les moyens utilisés vis-à-vis de l'utilisateur pour l'authentifier. Les seconds sont les outils utilisés pour transmettre l'information d'authentification au système distant.

2.1.1 Modalités d'authentification

Les modalités d'authentification sont les moyens utilisés par l'utilisateur pour fournir les informations confirmant son identité. Elles peuvent être regroupées en trois catégories : ce que l'on sait (comme un mot de passe), ce que l'on est (les solutions biométriques) et ce que l'on possède (par exemple une carte à puce). Lorsque l'on met en œuvre des solutions multi-modales il est préférable d'utiliser des modalités provenant de plusieurs groupes différents ; cela renforce le processus d'authentification. On parle alors d'authentification forte.

2.1.1.1 Approches classiques

Les modalités d'authentification classiques sont parmi les plus simples à utiliser. Elles ne requièrent pas de matériel spécifique et peuvent fournir une sécurité acceptable en fonction de la méthode utilisée.

La solution la plus courante est l'utilisation d'un mot de passe. Cette approche apporte un niveau de sécurité minimal. En effet, les mots de passe ne garantissent pas à eux seul qu'un individu donné est bien présent. Il existe de nombreuses attaques sur les mots de passe. La plus simple est l'attaque en force brute ; elle

vise à retrouver un mot de passe en tentant toutes les combinaisons possibles. Des variantes de cette attaque, comme une attaque par dictionnaire, permettent d'en améliorer l'efficacité. Même si une attaque de ce style peut facilement être contrée en limitant le nombre de tentatives autorisées, plusieurs facteurs font qu'elle représente malgré tout un réel danger. Tout d'abord, la plupart des mots de passe présentent une caractéristique commune qui est qu'ils doivent pouvoir être mémorisés par des utilisateurs ; combiné avec le fait que la plupart des utilisateurs choisissent eux-mêmes leurs mots de passe, l'information "secrète" est alors assez prévisible. De plus, de mauvaises pratiques de sécurité font que certains mots de passe sont réutilisés pour plusieurs services différents. La compromission d'un mot de passe sur un service présentant une sécurité moindre peut agir en cascade sur d'autres services mieux sécurisés. Ces raisons font que l'utilisation d'un mot de passe doit être associé à une politique de sécurité draconienne, ce qui peut aussi avoir pour effet de pousser les utilisateurs à contourner ces restrictions pour des raisons pratiques. Au final, l'utilisation d'un mot de passe seul ne permet pas d'obtenir un système d'authentification parfaitement sécurisé.

Une autre modalité d'authentification répandue est un système de cryptographie à clé publique (utilisation de paires de clés privées/publiques). Cette solution est d'un niveau de sécurité plus élevé que les mots de passe : l'élément "secret" servant à l'authentification n'est plus sous le contrôle de l'utilisateur (contrairement au choix d'un mot de passe). L'utilisation d'une paire de clés cryptographiques générées correctement rend quasiment impossible le risque d'attaque en force brute. Cependant, le problème est maintenant déplacé : au lieu de devoir découvrir le secret, il suffit de s'en emparer. En effet, pour fonctionner il est nécessaire que l'utilisateur ait accès à sa paire de clés, le plus souvent sous la forme d'un fichier. Finalement, l'authentification authentifie plus la possession de ce fichier que l'utilisateur lui-même. Pour relier ce fichier à l'utilisateur et éviter les risques de divulgation accidentelles une protection supplémentaire est mise en place sous la forme d'un mot de passe long (on parle alors de "phrase de passe"). Ce mot de passe ne sert pas réellement pour l'authentification, mais permet de déverrouiller la paire de clés. Cette approche consistant à utiliser une protection simple pour protéger l'accès à un élément de sécurité se retrouve dans des modalités utilisant du matériel spécifique. L'utilisation d'un tel système d'authentification peut se situer à la fois dans les catégories "ce que l'on possède" (le fichier) et "ce que l'on sait" (le mot de passe de déverrouillage).

2.1.1.2 Solutions matérielles

Afin de réaliser une authentification forte il est aussi possible d'utiliser des périphériques dédiés. Il y a quatre grandes catégories de modalités d'authentification s'appuyant sur des solutions matérielles : les jetons simples[40], les jetons

cryptographiques[40, 63], les solutions biométriques[34] et les "authentificateurs"[47, 48].

Les jetons simples reprennent le principe de l'utilisation d'un fichier contenant le secret permettant l'authentification et l'embarque dans un composant matériel. L'avantage est que l'on gagne en mobilité, car les informations d'authentification peuvent être transportées sans difficulté. Un tel jeton n'est pas nécessairement un appareil spécialisé. En effet, n'importe quel élément disposant de capacités de stockage peut être utilisé comme une clé USB [1]. Ce système conserve toutefois des inconvénients communs avec les solutions précédentes. L'information secrète doit transiter par la machine utilisée pour l'authentification, car les opérations cryptographiques impliquées pour en vérifier l'authenticité sont réalisées par le système "hôte". Ceci est également vrai pour le mot de passe utilisé pour garder l'information confidentielle. Dans le cas d'un système compromis, un attaquant peut alors récupérer les informations d'authentification d'un utilisateur et les reproduire sans difficulté.

Pour palier aux problèmes des jetons d'authentification simples, il est possible d'utiliser des systèmes plus élaborés embarquant des fonctions cryptographiques. L'exemple le plus courant est l'utilisation d'un système de carte à puce, mais il existe également des appareils spécialisés plus spécifiques. Ces solutions sont basées sur un même principe : pouvoir exécuter les algorithmes cryptographiques nécessaires sur le matériel externe sans jamais divulguer les informations d'authentification. La plupart embarquent une paire de clés privée/publique ainsi que les composants nécessaires pour réaliser des opérations de signature numérique en utilisant cette paire de clés. Pour fonctionner, le système souhaitant procéder à l'authentification transmet un élément aléatoire au jeton d'authentification qui retourne une signature numérique. La vérification de cette signature numérique par le système distant confirme l'authentification de l'utilisateur. Cette procédure est illustrée par la figure 2.1. Avec de tels systèmes, il est impossible de dupliquer l'information servant à l'authentification, car elle est stockée sur le composant externe et ne peut pas être lue. Le jeton simple et le jeton cryptographique sont, comme pour l'utilisation d'un système basique de paire de clés, situé à la fois dans la catégorie de "ce que l'on sait" et "ce que l'on possède".

Une troisième solution nécessitant la mise en place de matériel spécifique est l'utilisation de modalités d'authentifications biométriques. Ces systèmes rentrent dans la catégorie "ce que l'on est". Il existe plusieurs types de mesures biométriques : reconnaissance faciale, empreinte digitale, œil (iris et rétine), voix, etc. D'autres types de mesures moins courants peuvent faire intervenir une étape d'analyse plus complexe : la démarche, la façon d'écrire, les mouvements de souris d'un ordina-

1. Universal Serial Bus, Bus de communication série répandu dont un usage courant est l'utilisation avec des périphériques de stockage amovibles

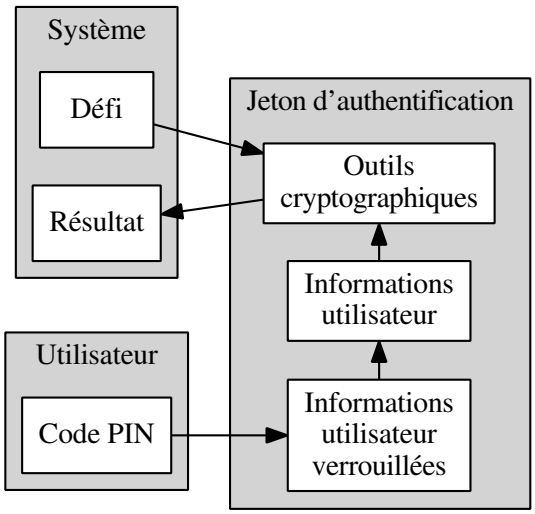

FIGURE 2.1 – Jeton d'authentification cryptographique

Sur ce schéma le "Code PIN" (Numéro d'Identification Personnel) joue le rôle du mot de passe

teur,... Tous ces types de mesures partagent le même principe de fonctionnement schématisé sur la figure 2.2. Les systèmes biométriques peuvent être considérés comme des comparateurs qui déterminent si un modèle obtenu lors de la tentative d'authentification correspond à un modèle pré-enregistré. Malgré différentes techniques consistants à enregistrer plusieurs modèles de références pour un même individu, ces solutions présentent toutes un risque non nul de faux positifs et de faux négatifs. Dans le cas des faux négatifs, l'utilisateur est pénalisé, car il doit retenter son authentification. Dans le cas des faux positifs, un attaquant est authentifié avec l'identité d'un autre, avec des conséquences sur la sécurité du système. Les systèmes biométriques doivent donc affiner leur fonctionnement pour réduire au maximum les risques d'erreurs. Une solution pratique pour réduire les faux positifs est de combiner plusieurs modalités biométriques (par exemple une empreinte digitale et une empreinte vocale). Les paramètres influant sur la reconnaissance de chacune des modalités sont modifiés afin de réduire le taux de faux négatifs individuels, ce qui a pour effet d'augmenter les faux positifs pour chaque système ; en contrepartie toutes les modalités doivent être satisfaites simultanément pour que l'authentification soit un succès, améliorant la fiabilité du mécanisme global.

La quatrième et dernière modalité d'authentification est l'utilisation de mots de passe jetables, ou OTP [2]. Il existe plusieurs approches à l'utilisation de mots de passe jetables[29, 28]. Il y a deux principes généraux pour la mise en place d'un tel système. Le premier est la génération à la demande de mots de passe par le système. Une fois générée, ce mot de passe à usage unique est transmis à

2. One-Time Password, mot de passe à usage unique généralement limité dans le temps

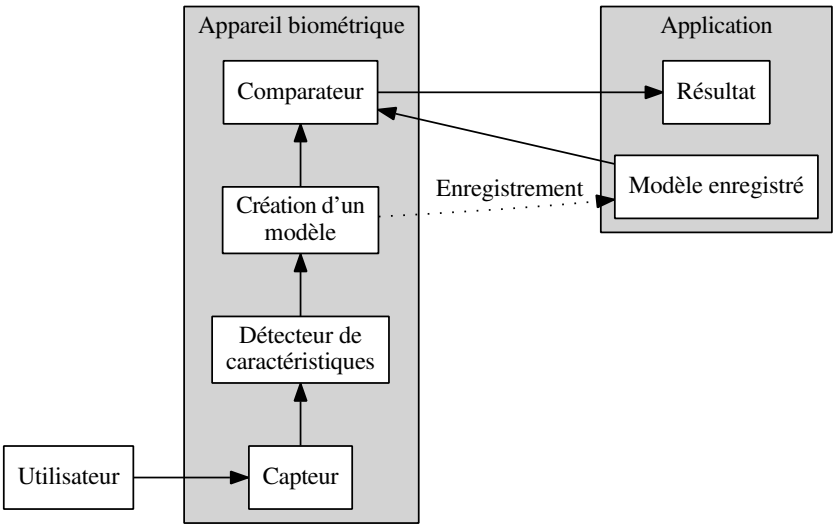

FIGURE 2.2 – Système d'authentification biométrique

l'utilisateur via un canal différent de celui sur lequel il souhaite s'authentifier, on réalise par exemple la transmission du mot de passe sur un réseau mobile pour réaliser une authentification sur un système relié à internet. Dans ce cas il est supposé que la réception de ce mot de passe par l'utilisateur souhaitant s'authentifier au même moment est une preuve suffisante. L'autre principe courant est de définir initialement un secret commun entre le système voulant authentifier ses utilisateurs et chaque utilisateur. Ce secret commun est alors utilisé pour générer une série de mots de passe à durée de validité limitée. Cette génération est déterministe et permet à chaque instant après la mise en place initiale qu'à la fois le système et l'utilisateur dispose du même mot de passe "courant". Du point de vue de l'utilisateur ce système fonctionne avec un "authentificateur". Les authentificateurs peuvent prendre différentes formes : un appareil dédié générant en continue de nouveaux mots de passe, une application pour téléphone mobile, etc. L'essentiel est qu'ils soient initialement synchronisés avec le système d'authentification. Dans les deux cas, l'utilisation du mot de passe jetable en lui-même nécessite moins de précautions qu'un mot de passe classique : même en cas de divulgation, la durée de validité d'un mot de passe dans un tel système est de quelques minutes.

2.1.2 Protocoles d'authentification

Nous nous intéressons dans ce chapitre aux protocoles d'authentifications utilisés dans le cas d'applications distantes. Cela concerne, entre autre, les solutions de type client/serveur et les applications distribuées. De manière générale, il s'agit de transmettre de façon sécurisée et fiable l'information d'authentification depuis

le poste client jusqu'à un système distant. Les approches permettant de réaliser ces transmissions sont généralement basées sur des outils cryptographiques afin d'apporter les garanties de confidentialité et d'intégrité des échanges. En fonction de la modalité d'authentification, la cryptographie peut aussi être un composant intégral de l'authentification en produisant le résultat succès/échec du processus. Bien qu'il existe de nombreuses manières de réaliser une authentification distante, la plupart des systèmes sont basés sur des principes similaires que nous allons détailler ici.

2.1.2.1 Solutions basées sur des mots de passe

Dans le cas de l'utilisation d'un mot de passe, il existe côté utilisateur un secret dont la connaissance permet de s'authentifier. Pour qu'un protocole d'authentification basé sur un tel secret puisse aboutir, tout ce qui est nécessaire est de prouver au système distant que l'on connaît ce secret. Il y a plusieurs possibilités pour prouver que l'on connaît un secret donné : simplement énoncer ce secret ou produire le résultat d'un calcul ne pouvant être fait qu'en connaissance de cette information.

Une solution basique consiste à établir une connexion sécurisée entre deux systèmes, puis à transférer les informations d'authentification sur cette connexion. Dans ce cas, il y a en fait une première authentification qui a lieu servant typiquement à authentifier le serveur. Cette première authentification permet d'établir une connexion confidentielle et authentifiée entre deux machines. Ce n'est que lorsque le serveur distant est correctement authentifié que l'utilisateur peut transmettre en toute sécurité les informations permettant son authentification. La transmission peut se faire sans aucune transformation, car nous supposons cet échange confidentiel. Ce genre d'authentification est parfois réalisé sur des connexions non sécurisées ; cette approche n'est pas acceptable, car elle entraîne la divulgation du mot de passe utilisé. Dans tous les cas, la transmission du mot de passe au serveur distant permet certes de réaliser une authentification de l'utilisateur, mais présente un défaut majeur. En effet pour fonctionner il est nécessaire que le système distant puisse comparer le mot de passe ainsi reçu avec une information qu'il aurait au préalable enregistrée. Dans ce cas, cette information est le mot de passe lui-même, ce qui signifie qu'au moins deux entités connaissent le secret de l'utilisateur : lui-même et le serveur distant. C'est un problème, car en cas de compromission de la base de données du serveur, un attaquant aurait alors tous les outils nécessaires pour pouvoir réaliser une fausse authentification.

Une approche plus avancée consiste à transmettre non pas le mot de passe, mais une transformation de ce dernier[37]. Le plus souvent, cette transformation utilise une fonction de hachage cryptographiquement sûre produisant une empreinte équivalente au mot de passe. Cette méthode n'est pas nouvelle, mais l'évolution des

fonctions de hachage fait qu'elle reste sûre en pratique. Comme il s'agit de fonctions à sens unique, le mot de passe original ayant servi à produire l'empreinte reste inconnu du système distant. Fonctionnellement cette approche est similaire à la précédente, mais elle permet de garder confidentiel le secret permettant à l'utilisateur de s'authentifier. Ce changement peut sembler anodin ; en effet le serveur distant dispose toujours de tous les éléments permettant de simuler l'authentification de ses utilisateurs. Cependant, le risque en cas d'attaque est limité au service vulnérable. L'extraction d'informations depuis un service conservant les empreintes des mots de passe plutôt que le matériel original ne met pas en danger les secrets connus uniquement des utilisateurs. Dans les deux cas, qu'il s'agisse de transmettre le mot de passe en clair ou une empreinte équivalente, le serveur distant effectue une simple comparaison et détermine le succès ou l'échec de l'authentification.

Une troisième solution permet d'utiliser de façon très sécurisée une authentification par mot de passe. Plutôt que de transmettre le mot de passe, ou une empreinte équivalente, on utilise le mot de passe dans un calcul qui pourra être vérifié indépendamment. Le serveur distant transmet un défi aléatoire au poste client, qui utilise le mot de passe pour réaliser une opération de chiffrement symétrique et transmet le résultat en retour. Le serveur connaissant le mot de passe peut effectuer la même opération et n'a plus qu'à comparer la valeur retournée par le client. Il y a des avantages à utiliser cette méthode. Premièrement, le secret de l'utilisateur ne transite jamais sur le réseau ; de cette façon il n'est pas nécessaire d'établir initialement une connexion sécurisée. Ensuite, la nature aléatoire du défi envoyé par le serveur distant élimine le risque de rejeu : un attaquant enregistrant l'échange aboutissant à l'authentification ne peut pas s'en servir pour produire une authentification frauduleuse ultérieurement. Enfin, pour obtenir la même protection contre la divulgation du mot de passe, il est possible d'utiliser le remplacement par l'empreinte du mot de passe. L'utilisation du mot de passe pour réaliser un chiffrement plutôt que sa transmission est l'approche utilisée dans la partie initiale du protocole Kerberos[50]. Kerberos est un système d'authentification en ligne permettant de faire du SSO [3] via un système de tickets distribués aux utilisateurs en fonction de leurs droits d'accès. L'authentification initiale de l'utilisateur via son mot de passe est réalisé en utilisant cette méthode de transmission de défi chiffré.

Les trois solutions décrites dans cette section sont applicables à toutes modalités disposant d'un secret côté utilisateur. Cela inclut évidemment les mots de passe, mais également l'utilisation d'une clé secrète protégée, d'un jeton d'authentification basique, d'un authentificateur et d'un système OTP. Dans tous ces cas, l'élément secret qui doit être fourni côté utilisateur peut être vu comme un bloc de données statiques dont nous devons prouver la connaissance à un moment

3. Single Sign-On, système permettant de s'identifier une seule fois pour l'utilisation de plusieurs systèmes

donné. On peut donc utiliser l'une des trois approches proposées pour prouver cette connaissance, quelle que soit l'origine du mot de passe utilisé.

2.1.2.2 Systèmes à clés publiques

L'un des problèmes récurrents des solutions basées sur des mots de passe, quelle que soit leur origine, est la nature symétrique du processus. Dans tous les cas, le serveur distant dispose de suffisamment d'informations pour pouvoir falsifier une authentification. Sans mettre en cause l'honnêteté des services utilisant ces approches, cela signifie surtout qu'en cas de fuite des informations stockées un attaquant pourra produire des authentifications frauduleuses. Une solution est d'utiliser des algorithmes asymétriques. Ces derniers, même s'ils sont plus coûteux en ressources, répondent très bien à ce problème. Qu'il s'agisse d'une utilisation personnalisée de paires de clés ou d'utiliser des certificats numériques, on parle alors de systèmes à clé publique. Une des caractéristiques de ces systèmes est de pouvoir produire, à partir d'une donnée et d'une clé privée, une signature numérique. Cette dernière peut alors être vérifie à partir de la clé publique associée à la clé privée utilisée, mais ne peut pas être reproduite sans connaissance de cette clé privée.

Quelles que soient les variations utilisées, l'approche commune s'articule toujours autour du système suivant : le serveur distant envoie un défi aléatoire au poste client, qui utilise la clé privée de l'utilisateur pour réaliser une signature numérique et l'envoyer en retour. On retrouve ici une approche similaire à l'approche avancée utilisant des mots de passe ; la différence étant qu'ici le serveur distant peut vérifier que la signature est valide, mais n'a pas les moyens de la reproduire sans l'intervention de l'utilisateur. Le problème principal de l'utilisation de systèmes à clé publique est que contrairement à un simple mot de passe, une paire de clés n'est pas un élément qu'il est possible pour un utilisateur de retenir ou de reproduire lors d'une authentification. Pour fonctionner, les clés sont stockées sous diverses formes : fichiers simples, certificats numériques, etc. Cela a pour conséquence qu'il n'est pas directement possible de s'authentifier sur un poste client quelconque ; il faut disposer de sa paire de clés à ce moment-là pour pouvoir exécuter le protocole d'authentification. La plupart des solutions s'appuient sur un transfert physique de ces informations. Le plus simple est l'utilisation d'une clé USB servant de jeton sécurisé basique ; en plaçant ses clés cryptographiques sur ce support de stockage, il est possible d'avoir un processus d'authentification nomade fiable.

Les systèmes d'authentification à clé publiques fonctionnent donc avec des paires de clés privées/publiques pouvant se trouver sur différents moyens de stockage, mais pas uniquement. La même approche est valide avec l'utilisation de jetons d'authentification avancés tels que les cartes à puces. En effet, ces dernières permettent de cacher les informations d'authentification au système utilisé par le client, mais peuvent produire des signatures numériques ayant les mêmes proprié-

tés que les solutions s'exécutant sur le poste client.

2.1.2.3 La spécification EAP

EAP [4] est un ensemble de spécifications ayant pour but de permettre le déroulement d'un processus d'authentification[2]. EAP propose une série de fonctions et d'interfaces génériques nécessaires au bon déroulement d'une authentification sur système distant. Pour réaliser l'authentification en elle-même, il s'appuie sur diverses méthodes adaptées à un grand nombre de modalités spécifiques. Parmi les plus courantes, nous pouvons citer les méthodes suivantes : EAP-TLS[64] basé sur l'utilisation de certificats X.509, EAP-PWD[30] utilisant des mots de passe et EAP-SIM[31] utilisé dans la téléphonie mobile est se servant d'une carte SIM [5] pour l'authentification,... Il existe de nombreuses méthodes d'authentification EAP non standardisées qui ont été développées pour répondre à des besoins spécifiques. Le protocole EAP n'est ni une modalité d'authentification (au contraire, il dépend de méthodes variées), ni un protocole de communication (il ne spécifie pas les formats requis pour son exécution). Il spécifie uniquement les éléments nécessaires au déroulement correct d'une authentification.

2.1.3 Extraction d'information depuis les protocoles

La majorité des protocoles et outils d'authentification existant répondent uniquement à la question d'authentification et ne permettent pas de manipulation des éléments utilisés. Même EAP qui se veut extensible ne permet pas simplement d'extraire une partie de l'information d'authentification fournie par l'utilisateur. Pour satisfaire aux exigences présentées dans la suite de ce chapitre et en particulier à partir de la section 2.3, il est nécessaire de pouvoir obtenir à partir du processus d'authentification un élément secret correspondant aux informations fournies par l'utilisateur. Cet élément secret ne doit être connu que de l'utilisateur ; cela implique un processus de transformation situé après la saisie des données, mais avant la transmission pour authentification. Cette modification doit donc se faire au niveau du protocole d'authentification utilisé.

4. Extensible Authentication Protocol, framework décrivant les éléments nécessaires à la réalisation d'une authentification

5. Subscriber Identity Module, les cartes SIM permettent d'identifier un abonné à un service

2.2 Modalité d'authentification et secret "cryptographique"

Les différentes modalités d'authentification s'appuient sur des éléments différents pour fonctionner. Dans certains cas (comme nous le verrons dans la section 2.3) il est utile de disposer d'un "secret cryptographique" côté client (sur le système de l'utilisateur). Ce secret cryptographique dépend de façon déterministe des informations d'authentification de l'utilisateur. L'objet de ce secret est de pouvoir initialiser l'utilisation de fonctions cryptographiques côté client, sans compromettre le secret utilisé pour l'authentification. Une fois ce secret "secondaire" (par rapport au secret initial de l'utilisateur) obtenu, il est alors possible de réaliser des opérations de chiffrements quelle que soit la modalité d'authentification initiale.

Les mots de passe permanents et certains jetons cryptographiques permettent d'obtenir un tel secret. En revanche, ce n'est pas le cas pour les solutions biométriques, les mots de passe jetables et les jetons cryptographiques. Cette contrainte de fourniture d'un secret "secondaire" permet de catégoriser les différentes modalités d'authentification en deux groupes.

2.2.1 Modalités incompatibles avec un secret cryptographique

La biométrie ne permet pas de produire un secret cryptographique déterministe de par son fonctionnement imprécis. Comme présenté sur la figure 2.2, les approches biométriques convertissent les données capturées en un modèle représentant cette capture. Ce modèle est ensuite comparé avec d'autres modèles préalablement enregistrés et s'il existe suffisamment de points communs l'authentification est un succès. Dans la mesure où le modèle utilisé pour la comparaison n'est jamais totalement identique entre deux authentifications, toute forme de dérivation d'informations à partir de cet élément produirait un résultat inutilisable dans un contexte de sécurité cryptographique. L'élément secret que l'on pourrait tenter de dériver en fonction du modèle présent côté client, c'est-à-dire le modèle obtenu lors de la capture des caractéristiques de l'utilisateur, ne serait pas constant.

De même, l'utilisation de mots de passe jetables est incompatible avec l'obtention d'un secret secondaire. En effet, puisque le mot de passe est renouvelé très régulièrement, dériver un secret secondaire à partir de ce matériel ne permettrait pas d'effectuer des opérations sur le long terme ; or l'intérêt ici est de pouvoir réaliser des opérations cryptographiques qui seront confidentielles vis-à-vis du serveur, mais pourront être inversées par l'utilisateur. Qu'il s'agisse d'un mot de passe à usage unique ou de l'utilisation d'un authentificateur, cette modalité ne permet pas de répondre à la contrainte de production d'un secret cryptographique.

Les jetons cryptographiques évolués masquant les informations de l'utilisateur

peuvent également être incompatibles avec cette contrainte. Selon leur principe de fonctionnement, certains ne donnent aucun moyen d'accéder aux informations spécifiques de l'utilisateur. Cela en fait des objets sûrs mais empêche de produire un résultat déterministe qui pourrait être utilisé ici pour générer un secret secondaire. C'est le cas des jetons ne fournissant qu'une possibilité de signature numérique par l'algorithme DSA [6]. Les signatures obtenues lorsque l'on signe deux fois le même élément sont toutes les deux valides, mais différentes ; nous ne pouvons donc pas en dériver un secret utilisable.

2.2.2 Modalités compatibles avec un secret cryptographique

Les mots de passe permanents sont le premier exemple permettant d'obtenir un secret secondaire côté client. Bien qu'il soit possible de les utiliser tels quels, il est préférable de se servir d'un mot de passe comme paramètre pour une fonction déterministe qui sera chargée de produire le secret secondaire. Cette étape permet de dissocier la sécurité des opérations côté client et la sécurité des opérations réalisées par le serveur distant après la réussite du processus d'authentification. Idéalement, le mot de passe n'est plus utilisé par le processus d'authentification, mais une autre fonction est utilisée afin de produire l'élément servant à authentifier l'utilisateur. Ce fonctionnement est illustré sur la figure 2.3. À partir du mot de passe de l'utilisateur, nous dérivons deux secrets différents indépendant en utilisant une fonction de hachage cryptographiquement sûre. Le premier secret est utilisé à la place du mot de passe vis-à-vis du serveur distant, qui ne connaît que ce secret (et ignore le mot de passe réel), alors que le deuxième secret sert à réaliser les opérations côté client.

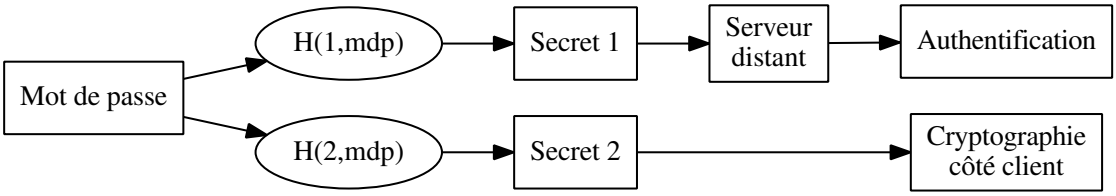

FIGURE 2.3 – Dérivation de deux secrets distincts depuis un mot de passe

Il est possible d'appliquer un principe similaire à certains jetons cryptographiques simples. Ceux se comportant comme des mots de passe "physiques" sont directement exploitables. Pour d'autres il peut exister des solutions alternatives permettant d'obtenir un élément secret reproductible. Certains jetons d'authentifications permettent d'embarquer une clé secrète, utilisée pour des opérations de chiffrement symétriques. Cette clé ne peut pas être extraite, mais il est possible

6. Digital Signature Algorithm, standard de signature numériques

d'envoyer des données au jeton et d'obtenir le résultat du chiffrement (ou déchiffrement) de ces données. En exploitant ce mécanisme, nous pouvons obtenir un secret secondaire unique lié à l'utilisateur. Pour ce type d'appareil, en envoyant un élément fixé (comme une chaîne de caractères fixe) nous obtenons bien un élément binaire chiffré qui sera identique à chaque fois.

Même en l'absence d'outils de chiffrement symétrique, il est possible de tirer parti d'autres mécanismes. Dans le cas d'un jeton ne fournissant que des outils pour réaliser des signatures numériques, il est possible de produire un secret cryptographique à la fois déterministe et lié aux informations d'authentification de l'utilisateur. Il faut pour cela que l'algorithme de signature numérique soit déterministe, ce qui est le cas notamment pour l'algorithme RSA [7] (présent sur les CPS [8] dont un usage est décrit à la fin de ce chapitre). Nous pouvons alors demander au jeton de réaliser la signature d'un élément fixé à l'avance de la même façon que l'on demandait le chiffrement symétrique précédemment. La signature produite sera bien reproductible et dépendra du secret de l'utilisateur sans pour autant l'avoir divulgué. Cette signature peut être vue comme un élément binaire unique et manipulée afin de produire une clé cryptographique acceptable servant alors de secret secondaire pour les opérations cryptographiques côté client.

Il faut noter que même si certaines modalités d'authentification ne permettent pas d'obtenir un secret cryptographique côté client, elles restent utiles pour réaliser l'authentification en elle-même. De plus, l'utilisation de processus d'authentification multi-modaux est plus que recommandée. Nous pouvons donc concevoir des systèmes tirant parti de plusieurs modalités d'authentification ; il faut simplement que l'une d'elles soit compatible avec la dérivation d'un secret côté client.

2.3 Gestion d'un trousseau de clés mobile

Nous décrivons dans cette section la première partie de la solution au problème de l'utilisation d'outils cryptographiques côté client basé sur le processus d'authentification. Pour garantir la sécurité des éléments manipulés dans une application sécurisée, il faut pouvoir fournir un certain nombre de propriétés similaires à celles décrites au chapitre 1 pour l'établissement de connexions sécurisées. Lors de la création et la manipulation de documents, comme pour des échanges d'informations comme de la messagerie, la confidentialité et l'authenticité sont primordiales.

Pour pouvoir garantir ce genre de propriétés de façon irréfutable, il est nécessaire d'utiliser des procédés cryptographiques. Cependant, ces derniers dépendent

7. Rivest Shamir Adleman, algorithme de chiffrement a clé publique parmis les plus utilisés au monde

8. Carte de Professionel de Santé, carte à puce permettant d'authentifier les professionels de santé

d'éléments secrets peu manipulable par les utilisateurs d'un système sécurisé. De plus, ces opérations doivent être réalisées sur le poste client, afin d'éviter de mettre en doute l'intégrité des opérations. En effet, une bonne solution pour sécuriser un système et d'accorder une confiance minimale à l'infrastructure, quelle qu'elle soit. En réalisant des opérations cryptographiques sûres au plus près de l'utilisateur, l'impact d'une vulnérabilité qui existerait sur le chemin des informations confidentielles est minimisé.

Afin de pouvoir réaliser des opérations cryptographiques telles que du chiffrement ou des signatures numériques, il est nécessaire de disposer d'éléments cryptographiques comme des clés et des certificats numériques. Ces éléments peuvent être stockés dans un trousseau de clés (ou porte-clés) de façon sécurisée, lui-même protégé par un système accessible pour l'utilisateur légitime, typiquement l'usage d'un mot de passe long. La protection d'un trousseau de clés sécurisé de cette façon permet de disposer d'un grand nombre de clés différentes sans avoir à retenir autant de mots de passe différents.

L'utilisation d'un trousseau de clés contenant des clés donnant à l'utilisateur différent types de droits lui permet d'appliquer les fonctions cryptographiques directement, sans avoir à accorder confiance à un tiers. Ainsi, le risque associé à l'utilisation d'un ou plusieurs intermédiaires dans le partage et la gestion des données reste sous le contrôle des utilisateurs qui décident, avant même leur communication, de la sécurité de leurs documents.

Pour qu'un processus d'authentification utilisé dans un système de contrôle d'accès classique puisse être adapté à l'utilisation de cryptographie côté client, il est nécessaire de faire trois choses. Tout d'abord, les droits qui étaient accordés à l'utilisateur par le processus d'authentification doivent être traduits sous forme de clés cryptographiques qui lui sont associées. Ensuite, il est nécessaire que ces clés puissent suivre l'utilisateur dans un contexte de mobilité : afin de ne pas régresser par rapport à un système existant, ce qu'il était possible de faire suite à un processus d'authentification classique s'exécutant sur n'importe quel système doit pouvoir rester faisable. Enfin, il faut le moins possible impacter le fonctionnement du processus d'authentification, aussi bien dans son exécution pour ne pas risque d'en affaiblir la sécurité, que dans son accessibilité pour conserver une expérience utilisateur similaire.

2.3.1 Gestion des droits

La finalité de la plupart des processus d'authentification est d'attribuer des droits à un utilisateur donné. Pour une application distante, cela signifie que les actions de l'utilisateur seront vérifiées, côté serveur, en fonction des droits dont il dispose. La conséquence est que l'ensemble du traitement des données de l'utilisateur, allant de la gestion de ses droits effectifs jusqu'à la protection des informations

manipulées, se produit sur le serveur. Dans un tel système il y a de plusieurs points vulnérables qui compromettent la sécurité du système. Un attaquant pourrait exploiter des faiblesses pour notamment : s'attribuer des droits, accéder aux données disponibles sur le serveur, usurper l'identité d'un utilisateur,... Déplacer la gestion et l'application des droits vers le client permet de réduire ces risques, mais nécessite un travail d'adaptation par rapport à l'approche classique.

Nous considérons plusieurs types de "droits" pouvant être associé à un utilisateur. Chacun de ces types donne lieu à une application cryptographique spécifique permettant de garantir que seul le détenteur d'un droit peut réaliser l'action associée. Nous allons ici examiner trois cas courants de droits utilisateurs : l'accès direct à des données, l'édition collaborative suivie d'un document partagé et l'envoi de messages.

Accès aux données : L'accès aux informations appartenant à un groupe spécifique (par exemple, un projet donné) est un type de droit d'accès classique. Dans le cadre d'un système ou la gestion des droits est uniquement réalisée sur le serveur, c'est ce dernier qui détermine s'il peut transmettre ou non l'information à l'utilisateur qui en fait la demande. Ce choix n'est imposé que par le contrôle d'accès du serveur ; une erreur, volontaire ou non, peut entraîner la délivrance incorrecte d'informations, car ces dernières ne font l'objet d'aucune autre forme de protection. Ce type de droit d'accès peut être traduit en la possession d'une clé secrète. Cette dernière est utilisée pour chiffrer et déchiffrer le document sur le poste client, avant son envoi et à sa réception. Ainsi, le fait de posséder la clé secrète devient l'équivalent de l'attribution du droit d'accès. Le serveur peut continuer à effectuer ses vérifications, mais dans le cas de l'envoi d'un document à un attaquant ayant pu perturber la gestion des droits, les informations restent confidentielles.

Édition collaborative suivie : Un autre type de droit d'accès est l'édition d'un document partagé. Dans ce cas, plusieurs collaborateurs peuvent porter des modifications à un document conservé sur le serveur, qui conserve une trace de ces changements. Si la gestion des droits est conservée côté serveur, il n'y a pas de preuve vérifiable que ces changements soient associés aux bons utilisateurs ; tout mécanisme ainsi mis en place peut être falsifié, dans la mesure ou le serveur dispose de tous les éléments permettant de réaliser le suivi. L'utilisation de fonctions de signature numérique permet de répondre à ce problème. Au-delà de son authentification classique, un utilisateur peut disposer d'une paire de clés privée/publique. La clé privée n'est connue que de l'utilisateur et permet de créer une signature numérique prouvant qu'un enregistrement est valide. La clé publique de chaque utilisateur est connue de tous, y compris du serveur et permet à quiconque de vérifier la validité des informations enregistrées.

Envoi de messages : L'envoi de messages entre utilisateurs d'un système est similaire à l'établissement d'une connexion sécurisée : nous souhaitons y retrouver des propriétés de confidentialité, intégrité et authenticité. L'utilisation d'un système classique centralisant la gestion de l'authentification et des droits sur le serveur limite les possibilités à ce niveau. Le serveur a accès aux messages et leur origine n'est vérifiable qu'en se basant sur la bonne foi de ce dernier. La traduction de ces envois via un système cryptographique nécessite l'utilisation de la cryptographie asymétrique. Pour cela, chaque utilisateur doit disposer d'une paire de clés privée/publique, comme dans le cas de l'édition collaborative. Il est d'ailleurs possible d'utiliser la même paire de clé pour ces deux usages. Un message envoyé dans ce contexte est alors chiffré à partir de la clé publique du destinataire et signé par l'émetteur. Le serveur effectue toujours la transmission, mais l'identité de l'émetteur est maintenant vérifiable et seul le destinataire peut accéder au contenu du message.

Nous pouvons ainsi traduire différents types de droits d'accès en des applications cryptographiques adaptées. Ces dernières dépendent d'éléments secrets, des clés cryptographiques, ainsi que d'éléments publics tels que les clés publiques des utilisateurs. La base permettant à ces approches de fonctionner de façon prouvable est que le serveur n'a jamais connaissance des éléments secrets. Qu'il s'agisse des clés permettant d'accéder à des documents ou des clés privées permettant de réaliser des signatures numériques, ces éléments sont toujours manipulés côté client. Une conséquence de cette décentralisation est que le serveur ne peut plus attribuer seul de nouveaux droits aux utilisateurs. Ce sont les utilisateurs eux-mêmes qui peuvent étendre leurs droits à d'autres. Le serveur ne joue alors qu'un rôle d'intermédiaire dans les communications entre clients et d'annuaire dans la gestion des clés publiques des utilisateurs. La figure 2.4 illustre de quelle façon il est possible de transférer des droits d'un utilisateur à un autre. Sont présents deux utilisateurs, le premier (A) étendant un droit d'accès au second (B). L'extension des droits passe par le serveur pour plusieurs raisons : pour des raisons de suivi et pour des raisons pratiques. Le suivi permet de garder une trace sur le serveur des droits effectifs des utilisateurs. L'aspect pratique est qu'une transmission utilisant un intermédiaire de cette façon permet une transmission de droits asynchrone. Si l'utilisateur B n'est pas présent au moment de cet envoi d'autorisation, il pourra tout de même la récupérer après coup, même si l'utilisateur A n'est lui-même plus présent par la suite.

Un des problèmes de cette approche est qu'elle nécessite un nombre important de clés. En effet, si chaque clé (ou paire de clés le cas échéant) correspond à un droit d'accès donné, leur nombre est aussi élevé que le nombre de droits possibles. La gestion d'autant de clés peut se faire de façon entièrement automatisée. Cependant, afin de garantir que seul l'utilisateur légitime peut accéder à ces clés, il

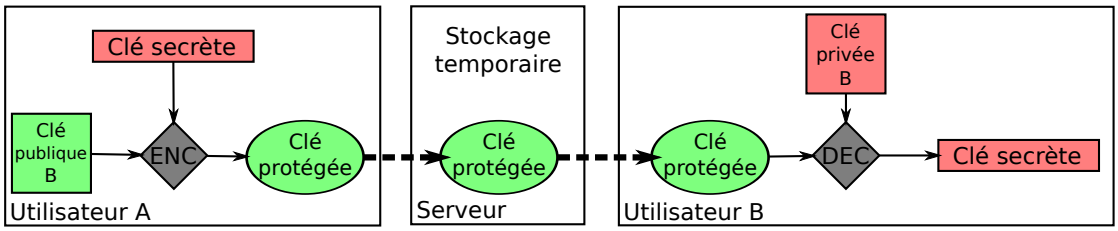

FIGURE 2.4 – Extension de droits entre utilisateurs

Les fonctions *ENC* et *DEC* permettent respectivement de chiffrer un message avec une clé publique et de déchiffrer avec la clé privée associée. La "Clé secrète" représente le droit d'accès à un document.

est nécessaire de les protéger de façon fiable à partir d'un secret connu de l'utilisateur seul. L'utilisation d'une protection unique permettant de déverrouiller un trousseau de clés est dans ce cas une solution appropriée.

2.3.2 Utilisation nomade du trousseau de clés

La démocratisation de l'informatique en général et l'omniprésence des accès à Internet a entraîné une évolution importante concernant l'utilisation de systèmes distribués. L'utilisateur n'est en effet plus limité à son poste de travail, mais peut au contraire disposer d'une mobilité accrue. Cela se traduit par l'utilisation de différents postes, mais aussi de différents types d'appareils y compris des smartphones et des tablettes. La conception de systèmes d'authentification doit prendre en compte cet aspect de mobilité et ne pas lier un utilisateur à une machine ou un groupe de machines. Un même utilisateur doit pouvoir s'authentifier sur différents systèmes et ce de façon non anticipée. De fait, toute solution se basant sur la conservation d'informations sur un système donné est inadaptée.

Ce besoin de mobilité des utilisateurs est difficilement compatible avec l'utilisation proposée d'un trousseau de clés sécurisé sur le poste client. Pour pouvoir fonctionner, il faudrait en effet que l'utilisateur puisse emporter ce trousseau de clés avec lui sur différents appareils. Dans certains cas ce n'est pas réalisable. S'il est possible d'imaginer la conservation de ce trousseau de clés sur un support de stockage amovible, il est difficile de trouver une solution qui puisse fonctionner sur tous types de systèmes. De plus, cela nécessiterait de modifier le fonctionnement de l'authentification pour les situations utilisant des modalités d'authentification non matérielles. Le problème est donc de fournir à l'utilisateur de façon transparente son trousseau de clés sécurisé sur chaque appareil où il s'authentifie.

La solution proposée ici repose sur l'utilisation d'un serveur fournissant un stockage sécurisé de trousseaux de clés. Il est important de conserver ces clés confidentielles vis-à-vis du serveur. Le chiffrement se fait donc, comme toutes les

opérations fiables, côté client. Le chiffrement du trousseau de clés dépend du secret "secondaire" décrit précédemment, de façon à ne pas le divulguer au serveur. Lors d'une authentification conventionnelle réussie, le serveur fournit à l'utilisateur son trousseau de clés sécurisé ; l'utilisateur seul peut alors le déverrouiller et utiliser les clés qu'il contient.

2.3.3 Processus d'authentification et trousseau de clés

L'utilisation d'un trousseau de clés sécurisé réduit les risques de divulgation de son contenu à un autre que l'utilisateur légitime. Le chiffrement permet en effet de restreindre l'accès aux clés qu'il contient. Cependant, aucun système n'est infaillible ; c'est pourquoi il convient de limiter au maximum les risques de divulgation du contenu, même chiffré. C'est pour cette raison que le serveur, disposant des trousseaux de clés sécurisés, ne fournit pas les trousseaux sur simple demande, même s'ils sont protégés. Nous conservons le processus initial d'authentification qui permet de confirmer l'identité de l'utilisateur sur le serveur. Ce n'est qu'une fois ce processus terminé avec succès que le trousseau de clés de l'utilisateur est envoyé. En cas d'exploitation d'une faiblesse du processus d'authentification, l'attaquant n'aura malgré tout pas accès aux droits de l'utilisateur dont il a usurpé l'identité.

Pour qu'un système mette en œuvre cette séparation de l'authentification et de l'attribution des droits via un trousseau de clés sécurisé, il est important de rendre ce changement aussi transparent que possible. En particulier, il ne faut pas demander à l'utilisateur de déverrouiller son trousseau de clés après avoir déjà procédé à son authentification. L'intégration de ces deux éléments en une seule étape est rendue possible par l'utilisation des informations saisies par l'utilisateur lors de l'authentification pour procéder au traitement du trousseau de clés par la suite.

Afin de limiter l'accès au trousseau de clés à l'utilisateur seul, il est nécessaire de mettre en œuvre des solutions telles que celle illustrée dans la figure 2.3. Pour assurer que le serveur ne puisse pas disposer du matériel secret (le mot de passe dans cet exemple) servant à dériver les deux sous-secrets, il est nécessaire de modifier le processus d'authentification. La modalité utilisée pour protéger le trousseau de clés doit être changée avant son envoi. Il faut aussi rendre disponible le deuxième secret ("Secret 2" dans la figure) à la partie cryptographique côté client. Ces changements n'impactent pas l'utilisateur, qui utilise la modalité d'authentification de manière classique. Il faut cependant tenir compte de la modification du système pour que le serveur distant soit prêt à prendre en compte le premier sous-secret ("Secret 1" dans la figure) au lieu du mot de passe initial de l'exemple. Une fois le protocole d'authentification terminé avec succès, le serveur fournit au client le trousseau de clés sécurisé. L'application dispose alors de tous

les éléments nécessaires pour utiliser les clés présentes dans le trousseau de clés sans action supplémentaire de l'utilisateur.

La figure 2.5 illustre ce processus dans le cas où on utilise un mot de passe. Cette modalité peut être remplacée par une autre modalité compatible permettant de dériver un secret secondaire. Comme pour une authentification classique, le nombre d'interactions avec l'utilisateur est limité. Malgré cela, l'ensemble du système fonctionne. Au terme de la procédure, l'utilisateur est authentifié vis-à-vis du serveur. Il dispose également de son trousseau de clés personnel, contenant tous ses droits d'accès au système.

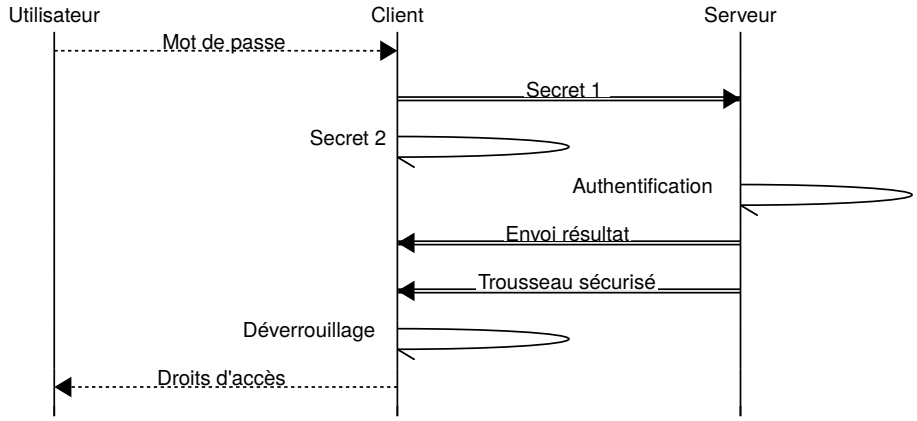

FIGURE 2.5 – Déverrouillage automatique du trousseau de clés sécurisé

L'ensemble des opérations entre la saisie du mot de passe par l'utilisateur et l'obtention des droits d'accès sont automatiques. Les pointillés représentent les interactions utilisateur, les demi flèches les traitements (hachage, chiffrement,vérification) et les flêches doubles les communications réseau.

La mise en place d'un tel mécanisme associant authentification et trousseau de clés en remplacement d'un mécanisme d'authentification existant au préalable peut se faire avec uniquement des modifications côté client. Il suffit de modifier la procédure d'enregistrement des utilisateurs pour tenir compte de la transformation des informations envoyées de la même manière que lors d'une authentification. Ainsi, dans l'exemple de l'utilisation d'un mot de passe, l'authentification réalisée sur le serveur se déroule de la même manière qu'il s'agisse du mot de passe initial ou d'une version transformée. En effet, si l'information reçue par le serveur lors de l'enregistrement initial d'un utilisateur est déjà le mot de passe transformé, il n'y a alors pas de différence dans la procédure de validation ; le secret est juste transformé avant d'être vu par le serveur dans tous les cas. Dans le cas d'une authentification multi-modale, les autres modalités ne sont pas du tout affectées par ces changements ; seule la modalité servant à déverrouiller le trousseau de clés nécessite un changement. Cela permet aussi bien d'intégrer ce système de trousseau

de clés dans des applications existantes via des modifications côté client que dans de nouvelles applications.

2.3.4 Systèmes multi-modaux

Le déverrouillage effectif du trousseau de clés sécurisé est l'élément central du mécanisme de transfert de trousseau de clés sécurisés. Comme nous avons pu le voir, cette action doit répondre à deux critères principaux : être impossible à réaliser par le serveur et ne pas imposer davantage d'interactions avec l'utilisateur. Pour cela, des solutions permettant à la fois de masquer les informations d'authentification de l'utilisateur et d'en tirer parti pour disposer d'un "secret" pouvant servir de clé de déchiffrement ont été décrites. Il reste toutefois un cas qui n'a pas été décrit : le déverrouillage du trousseau de clés sécurisé par différentes méthodes d'authentification.

En effet, le contexte de mobilité des utilisateurs peut les amener à s'authentifier sur différents systèmes pouvant utiliser des modalités d'authentifications différentes. Mais il existe des cas pour lesquels plusieurs de ces modalités peuvent être utilisées pour déverrouiller le trousseau de clés sécurisé. Dans ces conditions, il n'est pas nécessaire d'imposer une méthode partagée par tous les systèmes. L'approche la plus générique en termes d'accessibilité reste l'utilisation d'un mot de passe. Elle peut être mise en œuvre sur pratiquement tous les systèmes imaginables et ne nécessite pas de matériel spécifique.

Si un système dispose d'autres modalités d'authentification permettant de déverrouiller le trousseau de clés sécurisé, il n'est pas utile d'imposer en plus l'utilisation du mot de passe. Afin de gérer ce cas, la solution technique proposée est un mécanisme permettant d'utiliser différents éléments indépendamment pour accéder au même contenu. Le fonctionnement est illustré sur la figure 2.6. La clé principale K permet de déverrouiller l'ensemble du trousseau de clés. Les différentes modalités d'authentification devant permettre le déverrouillage du trousseau de clés produisent chacune un secret M_i. La clé K est chiffrée autant de fois que le nombre de modalités d'authentification avec les secrets M_i correspondant. De cette façon, les différentes modalités d'authentification permettent chacune de déchiffrer la clé K à partir de leur secret M_i, donnant ainsi accès au contenu du trousseau de clés.

2.4 Nouveau protocole d'authentification distant

Nous présentons dans cette section un nouveau protocole d'authentification distant. En plus de devoir fournir un résultat de succès ou d'échec, ce protocole doit pouvoir prouver ce résultat sur un système distant. Pour ce faire, il est nécessaire de transférer de façon sécurisée les informations d'authentification, obtenues sur

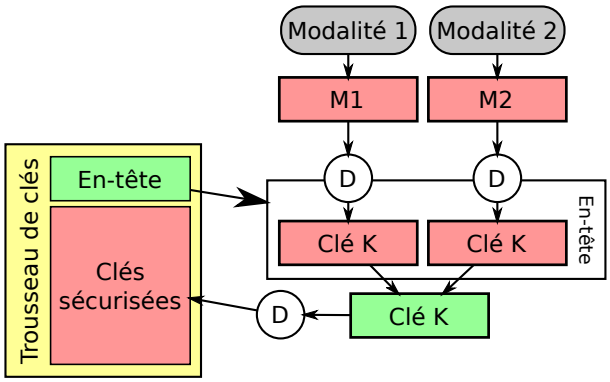

FIGURE 2.6 – Utilisation de clés multiples pour protéger un secret unique

Les cercles marqués “D” représentent une opération de déchiffrement. Les éléments rouges indiquent un contenu chiffré ou secret.

le poste client, vers le serveur. Afin de permettre un fonctionnement flexible, nous ferons la distinction entre deux serveurs : le serveur de service et le serveur d'authentification. Le serveur de service représente le système exécutant le service pour lequel l'on souhaite s'authentifier. Il n'a pas besoin de connaître les détails de l'authentification, mais uniquement de pouvoir valider qu'un client est correctement authentifié. Le serveur d'authentification dispose des informations permettant de réaliser l'authentification effective des clients. Cette approche permettant de dissocier le processus d'authentification de l'utilisation du résultat rend possible des applications de types SSO. Il est toutefois possible que ces deux serveurs soient confondus si l'authentification n'est utilisée que par un seul service. Au terme de l'exécution avec succès du protocole, le client obtient la confirmation qu'il est authentifié et le serveur de service peut vérifier cette confirmation lors du traitement des requêtes du client.

2.4.1 Présentation du protocole

La figure 2.7 illustre la direction des échanges considérés. Les interactions avec les modalités d'authentification sont limitées au poste client qui se charge d'agréger leurs résultats de façon appropriée pour transmission au serveur d'authentification. Une fois les vérifications effectuées et l'identité validée, le serveur d'authentification produit une réponse sous la forme d'un jeton sécurisé. Finalement, ce jeton sécurisé permet à l'utilisateur d'accéder au service ; la validité du jeton est vérifiable sans interactions entre les deux serveurs grâce à un mécanisme de chiffrement asymétrique. L'utilisation du jeton sur différents services permet effectivement de fonctionner selon le principe d'un système SSO.

Le protocole d'authentification décrit ici peut fonctionner avec différentes mo-

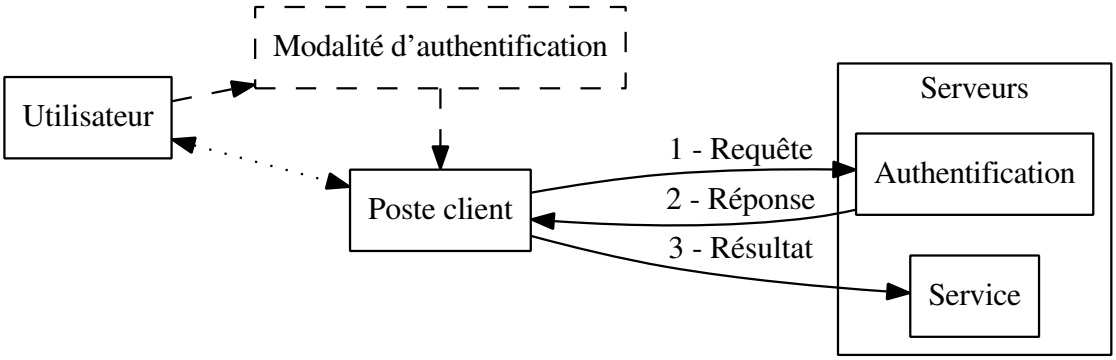

FIGURE 2.7 – Les échanges du protocole d'authentification

Les tirets représentent l'interaction initiale entre l'utilisateur et la modalité d'authentification. Les échanges entre client et serveurs se déroulent dans l'ordre indiqué.

dalités d'authentification, y compris des systèmes multi-modaux. Il a toutefois été conçu pour tirer parti de systèmes permettant de produire des signatures numériques. Bien qu'une signature numérique ne soit pas obligatoire, elle permet d'apporter un haut niveau de confiance dans le déroulement du processus. La méthode obtenue pour réaliser cette signature numérique importe peu, car celle-ci ne sert qu'à valider le bon déroulement du protocole.

Le protocole d'authentification produit lors de chaque authentification une paire de clés privée/publique temporaire utilisée pour la durée de vie de la session utilisateur. C'est sur cette paire de clés que sont basées les opérations ultérieures, y compris l'utilisation avec le serveur de services. La confiance accordée à la modalité d'authentification initiale est ainsi transmise à cette paire de clés temporairement associée à l'utilisateur. Il y a deux raisons majeures à ce transfert de confiance. D'une part cela permet la mise en place de moyens cryptographiques basés sur du chiffrement asymétrique quelle que soit la nature des modalités d'authentification utilisées initialement. D'autre part certaines modalités d'authentification peuvent s'avérer trop lente pour une utilisation régulière. C'est notamment le cas avec l'utilisation de certains jetons d'authentification cryptographiques pouvant mettre plusieurs secondes à produire une signature numérique. Pour ces systèmes, l'utilisation d'une solution de confiance plus performante permet une meilleure réactivité sans sacrifier la sécurité.

Le protocole dispose de deux modes de fonctionnement ; une version "non-confidentielle" conçue pour s'exécuter sur une connexion sécurisée par d'autres moyens et une version confidentielle, apportant sa propre sécurité. La version confidentielle est le résultat de la fusion de la version non-confidentielle et de l'établissement d'une connexion sécurisée décrite dans le chapitre 1 pour le protocole SVC [9].

9. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécuri-

Dans ce cas, il est nécessaire de fournir une méthode d'authentification permettant d'identifier le serveur uniquement ; l'authentification du client est réalisée dans la réponse faite au serveur lors de la négociation du protocole. Un inconvénient mineur de cette approche est que pour exécuter le protocole d'authentification, il est nécessaire d'avoir une troisième communication du serveur vers le client pour confirmer le résultat. Cette troisième transmission n'ayant qu'un rôle "informatif" pour le client, il est raisonnable de l'adosser à des communications utiles réalisées par la suite sur le canal sécurisé.

2.4.2 Conception du protocole

L'exécution du protocole d'authentification se fait en deux étapes : l'envoi d'une requête au serveur d'authentification et la transmission de sa réponse au serveur de service. Dans le cas où ces eux entités seraient confondues, nous pouvons évidemment nous passer des échanges 2 et 3 indiqués dans la figure 2.7.

2.4.2.1 Requête d'authentification

Lors d'une demande d'authentification, l'utilisateur applique les modalités d'authentifications requises. Les résultats obtenus à partir de ces modalités sont agrégés dans un élément appelé requête d'authentification. Cette agrégation varie d'une modalité à l'autre. L'essentiel est de transmettre les éléments nécessaires pour que le serveur puisse les valider. Cette requête d'authentification contient, en plus des informations d'authentification, différents éléments permettant de valider la demande comme l'identité de l'utilisateur, la clé publique de session, et des méta-données spécifiques au service demandé. Ces méta-données sont "libres" et déterminées par l'application mettant en œuvre le protocole d'authentification. La clé publique de session est associée à l'utilisateur pour la durée de validité de son authentification et permet de valider les requêtes de service ultérieures. Ces requêtes de services sont associées à une signature numérique réalisée avec la clé privée "de session" permettant de les lier à la requête d'authentification.

On peut classer les modalités d'authentification en trois catégories en fonction de leur agrégation dans la requête d'authentification : les modalités produisant une signature numérique, les modalités basées sur la connaissance d'un secret et les modalités nécessitant la transmission d'une information pour vérification.

Modalités produisant une signature numérique : Pour ces modalités, l'utilisateur dispose d'une paire de clés permettant de réaliser une signature numérique. La clé publique est intégrée à la requête d'authentification (sois directement, sois

sée dédiées aux communications internes des applications

sous forme d'un certificat numérique selon ce qui est disponible). La requête d'authentification est signée numériquement en utilisant la modalité. La signature ainsi obtenue ne remplace pas celle réalisée en utilisant la paire de clés de session, mais est ajoutée à sa suite. De cette façon, le serveur d'authentification peut vérifier que la modalité d'utilisation a correctement validée l'identité de l'utilisateur : la clé publique peut être vérifié par le serveur sans autres interactions et la signature numérique confirme ensuite que l'utilisateur dispose bien de la clé privée associée. La validité de ces vérifications permet de transmettre la confiance accordée à la clé publique de l'utilisateur vers la clé de session produite ponctuellement.

Modalités basées sur la connaissance d'un secret : Les modalités d'authentification basées sur la connaissance d'un secret (comme les mots de passe) fonctionnent en prouvant la connaissance de ce secret sans nécessairement le divulguer. Pour confirmer cette connaissance, nous ajoutons une empreinte d'authenticité à la suite de la requête d'authentification via un HMAC [10]. Cette opération fait intervenir l'élément secret connu de l'utilisateur et est vérifiable de façon autonome par le serveur d'authentification. Le but est le même que pour les modalités à signature numérique, c'est-à-dire transmettre la confiance accordée à la connaissance de ce secret vers la clé de session en associant les deux éléments dans une requête d'authentification intègre.

Autres modalités d'authentification : Certaines modalités d'authentification ne produisent pas de signature numériques et ne sont pas basées sur la connaissance d'un secret fixe. C'est le cas en particulier pour les modalités biométriques. Comme indiqué sur la figure 2.2 ces modalités produisent un modèle qui est ensuite comparé à une version enregistrée. Ce modèle est différent à chaque requête et ne peut donc pas servir de base pour un calcul d'intégrité ou d'authenticité. Pour les modalités d'authentifications fonctionnant sur ce principe, l'élément d'authentification doit être transmis en l'état par la requête d'authentification. Le serveur d'authentification effectue alors les vérifications sur ces données afin de confirmer leur validité. La requête d'authentification contenant ces données brutes étant signée par la clé privée de session de l'utilisateur, la confirmation de la validité des informations d'authentifications qu'elle transporte permet d'associer la paire de clés de session à l'utilisateur ainsi authentifié.

La finalité des différentes méthodes d'intégrations décrites précédemment est toujours d'associer les éléments authentifiant l'utilisateur à la paire de clés de session. C'est cette paire de clés qui sert à prouver une authentification réussie pour les requêtes de service ultérieures. De cette façon, les modalités utilisées

10. Hash-based Message Authentication Code, code d'authentification de message basé sur des fonction de hachage

initialement pour l'authentification ne sont plus sollicitées pour la durée de validité de la session.

2.4.2.2 Requête de service

Dans le cas ou les serveurs d'authentification et de service sont distincts, un mécanisme de jeton d'authentification est mis en place et permet au client de prouver une authentification réussie auprès du serveur de service. Cette approche est similaire au système Kerberos, mais s'appuie sur des modalités d'authentification différentes et un découpage différents des rôles. De plus, elle permet indifféremment de faire du SSO ou de fusionner les deux entités pour simplifier la mise en place du processus d'authentification.

Lorsque le serveur d'authentification valide une requête d'authentification, il produit un jeton d'authentification contenant toutes les informations requises par le service pour confirmer l'authentification de l'utilisateur. Pour commencer, l'identité de l'utilisateur est présente. Seul le serveur d'authentification doit connaître l'identité réelle de l'utilisateur. Pour protéger l'identité des utilisateurs, le serveur d'authentification peut utiliser une table de correspondance entre ces identités réelles et des identités virtuelles afin de masquer l'identité réelle de l'utilisateur au serveur de service. À la suite de cette identité est ajouté la clé publique correspondant à la session d'authentification. En plus de ces informations, les méta-données associées au service qui se trouvaient dans la requête d'authentification sont reproduites dans le jeton d'authentification. Pour finir, le serveur d'authentification ajoute des informations de validité telles que la durée de vie du jeton et effectue une signature numérique de l'ensemble afin de prouver qu'il a validé l'ensemble des éléments. À ce stade, le serveur d'authentification a avalisé la paire de clés de session de l'utilisateur. Cette clé est alors utilisée par le client pour contresigner le jeton d'authentification.

Lors de la transmission au serveur de service, celui-ci procède à la vérification des signatures. Tout d'abord, la signature réalisée par le client sur l'intégralité du jeton d'authentification est vérifiée. La validité de cette signature indique que l'utilisateur est bien à l'origine de cette transmission. Il ne reste plus alors qu'à valider cette paire de clés. Cela est fait en vérifiant la signature produite par le serveur d'authentification. Si ces deux signatures sont valides, cela signifie que la paire de clés de session de l'utilisateur a été validée par le serveur d'authentification après vérification des modalités d'authentification mises en œuvre. Le serveur de service peut donc associer la clé publique ainsi reçue à l'utilisateur le temps de la session. Lors de l'envoi de requêtes au serveur de service, ces dernières seront signées avec la paire de clés de session de l'utilisateur, permettant au serveur de service de valider leur origine.

2.4.3 Compatibilité avec le trousseau de clés sécurisé

La mise en place du protocole d'authentification décrit dans ce chapitre est compatible avec l'utilisation d'un trousseau de clés sécurisé décrit dans la section 2.3. Pour permettre l'utilisation d'un trousseau de clés côté client il faut considérer les points suivants : la possibilité d'extraire un secret "secondaire" du processus d'authentification, le déplacement des droits depuis les serveurs vers le client et la récupération du trousseau de clés côté client au terme de l'authentification.

Récupération de secret secondaire : Le secret secondaire nécessaire pour déverrouiller un trousseau de clés est un élément qui doit être lié aux informations d'authentification de l'utilisateur, mais confidentiel vis-à-vis des serveurs d'application. Le protocole d'authentification proposé utilise des méthodes d'agrégation des données d'authentifications permettant de masquer ces informations au serveur d'authentification. Il ne reste alors qu'à transformer ces informations d'authentification une seconde fois pour permettre le déverrouillage du trousseau de clés sécurisé. La transformation et l'agrégation réalisées pour la requête d'authentification n'est pas incompatible avec ce besoin de produire un secret secondaire.

Déplacement des droits vers le client : Avec l'utilisation d'un trousseau de clés sécurisé et d'opérations cryptographiques côté client, les droits d'utilisation du système ne sont plus conservés par les serveurs de service. Il reste le droit d'accès général à chaque service, contrôlé par le serveur d'authentification. S'agissant d'un droit ponctuel, lié au succès d'une procédure d'authentification pouvant faire intervenir des modalités variées ce droit d'accès ne peut pas être intégré au trousseau de clés. Il reste donc sous contrôle du serveur d'authentification. Les autres droits relatifs à l'utilisation du service ne sont pas affectés et peuvent toujours être conservé dans le trousseau de clés.

Disponibilité du trousseau de clés : Il y a trois possibilités pour le stockage des trousseaux de clés des utilisateurs : sur le serveur d'authentification, sur les serveurs de service ou sur un serveur dédié. La conservation sur le serveur d'authentification permet à l'utilisateur de récupérer ses droits immédiatement après le processus d'authentification et avant d'avoir contacté un serveur de service. Cette approche permet des usages ne passant pas par les serveurs de service (comme une messagerie sécurisée avec une clé contenue dans le trousseau) mais regroupe les droits de tous les services dans un seul trousseau de clés, ce qui augmente sa taille. Le stockage des trousseaux de clés sur les serveurs de services impose à l'utilisateur de contacter un service avant de disposer de son trousseau de clés mais permet de limiter la taille des trousseaux de clés en fonction des services concernés. L'utilisation d'un serveur dédié permet la centralisation de tous les trousseaux de clés

des utilisateurs, en conservant la possibilité de distinguer un trousseau par service. Cette dernière approche a comme intérêt de permettre une sauvegarde aisée de tous les trousseaux de clés, dont le contenu est critique. En cas de perte de son trousseau de clés, un utilisateur peut perdre des données de façon définitive. Dans tous les cas, la conservation du trousseau de clés s'apparente à un stockage de fichier simple, car le trousseau de clés est chiffré et déchiffré par le client. Le serveur chargé du stockage de ces trousseaux n'a donc pas de considérations de sécurité particulière à avoir en ce qui les concerne.

2.5 Implémentation d'un prototype

Un prototype du protocole d'authentification décrit section 2.4 a été réalisé dans le cadre d'une collaboration industrielle avec la société SESIN. L'objectif est de tirer parti du mécanisme d'authentification des CPS pour identifier les utilisateurs de Poseidon, une application de gestion de documents numériques. L'application fonctionne sur le principe client/serveur. Il est possible d'exécuter du code arbitraire sur le serveur, mais le client a comme contrainte d'être un client léger. Cette limitation a pour but de faciliter l'expérience utilisateur en lui permettant d'accéder au service en minimisant les étapes de configuration préalables. Cependant, du point de vue développement cela nécessite la création d'un applet spécifique permettant d'accéder au matériel. Au niveau de la modalité d'authentification, les CPS permettent de réaliser des opérations de signature numérique et disposent de deux paires de clés associées à l'utilisateur ; l'une dédiée aux étapes d'authentification, l'autre pour les signatures. Ces clés sont inaccessibles depuis l'extérieur, mais peuvent être choisies pour effectuer les opérations cryptographiques directement sur le matériel. Elles sont protégées par un code PIN [11] que l'utilisateur doit saisir initialement.

2.5.1 Architecture du prototype

S'agissant d'un prototype de protocole d'authentification, une application "minimaliste" servant de support a été développée. Puisqu'il fallait pouvoir s'intégrer à un client léger, nous avons conçu une application web invitant l'utilisateur à s'authentifier avec sa carte personnelle. La partie serveur a été entièrement écrite en langage C pour des raisons pratiques. Pour pouvoir accéder au matériel, un applet Java est automatiquement téléchargée par l'application web et contient un com-

11. Personal Identification Number, code permettant entre autre de déverouiller un jeton cryptographique matériel

posant permettant d'accéder à un jeton PKCS#11 [12] générique, en l'occurrence la CPS. Dans le cadre de ce prototype nous pouvons considérer que le serveur d'authentification et le serveur de service décrits précédemment sont confondus ; en pratique le prototype ne fournit pas de service autre que l'authentification. La figure 2.8 résume la disposition de chaque composant.

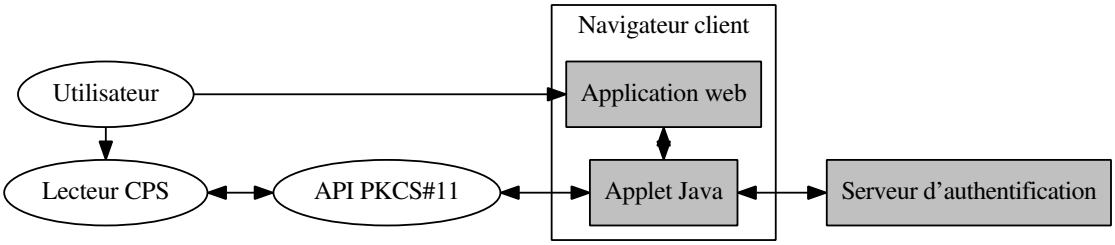

FIGURE 2.8 – Architecture du prototype d'authentification par CPS

Les cases indiquées en gris représentent les éléments développés lors de la réalisation de ce prototype.

Application web : La partie cliente du prototype est principalement constituée d'une application destinée à s'exécuter dans un navigateur internet, selon le principe du client léger. Une telle application est constituée d'une ou plusieurs pages HTML [13] liées entre elles par des transitions faisant généralement appel à un serveur pour déterminer la prochaine étape à afficher. Ces pages sont le plus souvent dynamiques et peuvent effectuer des requêtes au serveur et se mettre à jour en fonction des actions de l'utilisateur. Dans notre cas, l'application n'était constituée que d'une page dynamique. Cette dernière avait pour rôle de charger l'applet Java de façon invisible et de présenter une interface à l'utilisateur l'invitant à lancer la procédure d'authentification. Le contenu de la page changeait alors dynamiquement en fonction de la progression de l'authentification.

Applet Java : Pour pouvoir manipuler le jeton d'authentification PKCS#11 donnant accès à la CPS il est nécessaire de faire appel à une API [14] appelée Cryptoki. Cette dernière est présentée sous la forme d'une bibliothèque native s'exécutant sur le système et est différente en fonction de chaque système d'exploitation et de chaque jeton cryptographique. De plus, pour des raisons de sécurité il est impossible d'appeler directement des bibliothèques natives depuis un client léger

12. La spécification #11 définit une API générique pour l'utilisation de périphériques cryptographiques

13. Hypertext Markup Language, le langage de balises utilisé pour la construction de page web

14. Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

s'exécutant dans un navigateur. Afin de résoudre ce problème, un applet Java est utilisé. Ce dernier permet de créer une interface entre le langage de script utilisé dans l'application web (le JavaScript) et l'API Cryptoki servant d'interface avec le jeton PKCS#11, soit la CPS. Cet applet ne servant que d'interface applicative elle ne dispose d'aucun composant visible ; la totalité de l'interface utilisateur est gérée par l'application web afin de réduire au maximum la complexité de l'applet Java. En effet, pour fonctionner correctement, un applet Java qui nécessite de faire des appels à du code natif doit respecter un cahier des charges très contraignant, notamment en termes de déclarations de permissions et de signatures numériques du paquet la contenant. Minimiser la part de code présente dans l'applet permet de minimiser le nombre de changement à y apporter pour faire évoluer le reste de l'application et donc de réduire la complexité de développement.

Serveur d'authentification : Dans le cadre du prototype, le serveur n'avait qu'un seul rôle : mettre en œuvre la vérification du protocole d'authentification. À cette fin, un serveur en langage C a été écrit. L'applet Java communique directement avec ce serveur d'authentification. Pour permettre d'exécuter le protocole d'authentification, l'ensemble des outils cryptographiques nécessaires ont été implémentés. Cela inclut la vérification du certificat fourni par la CPS et les algorithmes de signature numérique requis pour réaliser les vérifications et la signature du jeton d'authentification transmis en retour à l'applet Java. Contrairement à une approche courante, nous ne passons pas ici par un langage de script pour le traitement de la requête. Le programme serveur créé pour ce prototype se comporte ainsi de façon similaire à un exécutable CGI [15]. L'autre approche possible aurait été d'inclure les fonctions de vérifications sous forme d'extensions pour un langage de script tel que PHP [16] ou Python. La conversion du prototype vers un véritable exécutable CGI ou une extension de script est triviale, mais n'a pas été réalisée dans le cadre de ce prototype.

2.5.2 Déroulement de l'exécution

Le prototype implémente l'intégralité du protocole d'authentification tirant profit de la CPS. La figure 2.9 résume l'exécution du protocole composant par composant. Une fois la modalité d'authentification disponible, ici le déverrouillage des clés par l'utilisation du code PIN, la requête d'authentification est construite. L'applet Java génère pour cette session une paire de clés privée/publique, inclue dans la requête d'authentification avec le certificat issu de la CPS. Cette requête

15. Common Gateway Interface, un standard permettant d'exécuter des programmes variés pour répondre à des requêtes web

16. PHP : Hypertext Preprocessor, un langage de script souvent utilisé pour la réalisation de sites web

d'authentification est signée par la CPS via l'API Cryptoki visible uniquement depuis l'applet Java. Le serveur vérifie la validité des signatures et du certificat présent dans la requête et produit la réponse sous la forme d'un jeton d'authentification. L'applet peut alors confirmer l'authentification à l'utilisateur par l'intermédiaire de l'application web.

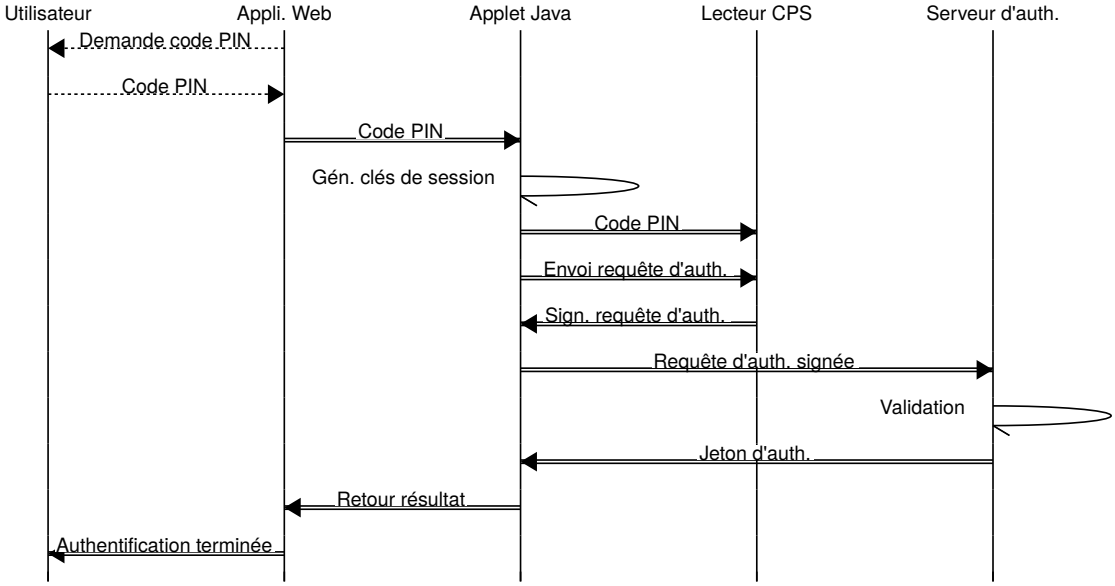

FIGURE 2.9 – Déroulement du processus d'authentification

L'ensemble des opérations entre la saisie du code PIN par l'utilisateur et la fin du processus sont automatiques. Les pointillés représentent les interactions utilisateur, les demi flèches les traitements (génération de clés, vérification) et les flêches doubles les communications.

Au terme de l'exécution du protocole, l'applet Java dispose du jeton d'authentification permettant d'utiliser un service de façon authentifié. De plus, les vérifications réalisées par le serveur d'authentification sont complètes : vérification de la validité du certificat, y compris la chaîne de certification du certificat X.509, ainsi que la vérification des signatures produites par l'applet Java et par la CPS.

Il reste un aspect non traité par ce prototype : l'envoi d'une requête de service qui fait suite à la validation de la requête d'authentification. Les outils logiciels pour réaliser ce type d'envoi sont déjà présents dans l'applet Java ; il s'agit des fonctions de signatures utilisés pour la construction de la requête d'authentification. La validation d'une requête de service par le serveur de service s'appuie également sur les mêmes outils cryptographiques que ceux utilisé pour le serveur d'authentification.

2.5.3 Possibilité d'extension du protocole

Le prototype ainsi réalisé à permis de mettre à jour un élément qui n'avait pas été envisagée lors de la conception du protocole d'authentification décrit à la section 2.4. Le fonctionnement du protocole fait qu'après l'étape d'authentification initiale de la session utilisateur, les modalités d'authentifications impliquées ne sont plus utilisées. Cependant, à l'usage la détection du retrait de la CPS de son lecteur doit déclencher un arrêt de la session authentifiée. Cet événement extérieur influence donc le fonctionnement du protocole et pourra faire l'objet d'une extension basée sur les développements de cette section.

L'une des raisons à la mise en place d'un système de transfert de confiance depuis les modalités d'authentification vers une paire de clés de session à usage unique est lié aux performances de l'application. La réalisation d'une signature numérique via la CPS prend environ deux secondes. Attacher cette pénalité à chaque requête permettrait effectivement d'obtenir la détection du retrait de la carte, mais n'est pas réaliste d'un point de vue pratique. En effet, une application peut nécessiter l'envoi de nombreuses requêtes, parfois très rapidement, afin d'être réactive. Nous parlons ici de plusieurs requêtes par secondes. La mise en place de notre protocole d'authentification permet de s'affranchir de cette pénalité à partir du moment où l'authentification initiale a réussi.

Cependant, pour la durée de vie du jeton d'authentification, le service suppose que l'utilisateur reste authentifié. Dans le cas d'un système utilisant une carte physique, en l'occurrence une CPS, il est souhaitable que le retrait de la carte invalide l'authentification de l'utilisateur. Malheureusement, par construction, la présence de la carte n'est plus nécessaire. Pour résoudre ce problème pratique, deux solutions ont été envisagées : une simple requête d'invalidation envoyée au serveur de service lors du retrait de la carte et l'utilisation d'une validation continue en parallèle.

La solution consistant à envoyer une requête d'invalidation lors du retrait de la carte peut sembler triviale. Lorsque l'applet Java détecte le retrait de la carte, elle arrête d'authentifier les requêtes qui lui sont transmises. Cette approche ne résout pas le problème, car rien dans les requêtes de service n'est lié à la présence de la carte. Si un attaquant parvient à bloquer la détection du retrait de la carte, l'applet Java va simplement continuer de fonctionner, car les requêtes de service ne dépendent pas d'un élément présent uniquement sur la carte. Afin de résoudre ce problème de détection du retrait de la carte de façon fiable il est nécessaire de réintroduire la CPS dans le processus de création des requêtes de service.

La solution utilisant une validation continue en parallèle est plus complexe à mettre en œuvre, mais permet de résoudre le problème posé. Il n'est pas possible d'utiliser une signature numérique produite par la CPS pour chaque requête, pour des raisons de performances. Cependant, seule une telle signature apporte

la garantie que la carte est toujours présente et déverrouillée. L'adjonction d'un mini-jeton d'authentification au jeton initial apporte un compromis intéressant. La durée de vie du jeton d'authentification initial peut être relativement longue (plusieurs heures) afin de permettre une utilisation sans avoir besoin de s'authentifier trop souvent. En revanche, le mini-jeton obtenu à partir de la CPS aurait une durée de vie bien inférieure, de l'ordre de la minute. Ce dernier est produit en signant régulièrement avant son expiration un élément reçu du serveur de service. Cette opération permet de garantir que la CPS est toujours présente pour la durée de vie de ce mini-jeton. Ce processus peut être totalement automatisé et transparent pour l'utilisateur. L'utilisation d'un élément aléatoire régulièrement fourni par le serveur permet d'assurer qu'un attaquant n'a pas pu produire à l'avance une série de faux mini-jetons. Le processus global d'authentification permet d'assurer au serveur de service que les vérifications sur le certificat de l'utilisateur ont été faites et permet de ne pas reproduire ces vérifications à chaque requête.

Ce besoin nouveau issu d'un simple prototype présente ainsi une extension intéressante du protocole d'authentification proposé. Les outils cryptographiques nécessaires dans ce cas sont les mêmes et il est donc possible d'implémenter cette solution à partir du code existant. Il reste toutefois à évaluer si cette solution est utilisable en pratique et à effectuer une analyse complète de sécurité pour s'assurer qu'un attaquant ne puisse pas passer outre cette mesure de protection.

2.5.4 Compatibilité avec le standard PKCS#11

L'applet Java utilise une API appelée Cryptoki pour communiquer avec la CPS. Cette interface correspond au standard PKCS#11 qui permet de manipuler différentes modalités d'authentification de façon standard en plus de la CPS. L'utilisation de ce standard fait que le prototype permet l'utilisation d'autre modalités d'authentification compatible PKCS#11.

2.5.5 Outils spécifiques côté serveur

L'implémentation a été réalisée avec la plate-forme de sécurité Arcana décrite au chapitre 3. Cependant, un certain nombre d'outils ont dû être implémenté spécifiquement pour utilisation avec la CPS : la manipulation de certificats X.509 et l'algorithme de vérification RSA. La manipulation de certificats X.509 était nécessaire pour extraire les informations du certificat produit par la CPS, c'est-à-dire l'identifiant de l'utilisateur et les informations de validité du certificat. L'algorithme de vérification de signature RSA était nécessaire pour vérifier la signature initiale produite par la CPS. Toutes les autres signatures intervenant dans ce pro-

totype sont basées sur des courbes elliptiques et l'algorithme ECDSA [17], tous deux initialement présents dans les outils utilisés.

2.5.6 Dépendance du prototype à l'environnement Java

Un problème très spécifique a impacté le développement de ce prototype. Pour fonctionner il dépend de la présence d'un environnement Java dans les navigateurs. Malheureusement, ce dernier ne s'est pas avéré aussi pérenne que prévu. Tout d'abord, les diverses évolutions de Java en lui-même compromettent régulièrement le fonctionnement de ce genre d'applet. Les changements réguliers de politiques de sécurité nécessitent une mise à jour régulière de l'applet pour continuer de fonctionner. Ensuite, la présence de l'environnement Java dans les différents navigateurs est actuellement dépendante d'une autre technologie appelée NPAPI [18]. Cette dernière est progressivement remplacée par des solutions alternatives[61, 65] avec lesquelles Java n'est pas compatible en l'état. Il est pour l'instant impossible de savoir si une évolution de Java va arriver afin de rétablir cette compatibilité. Toutefois, même si une nouvelle version de Java devait apporter à nouveau la compatibilité avec les navigateurs, la pérennité de ce support serait liée à trop d'éléments extérieures. Pour réduire ce risque, d'autres approches doivent être envisagées. Parmi les possibilités nous pouvons citer le développement de nouveaux plugins natifs, spécifiques à chaque navigateur. Cette solution pourrait présenter, en fonction des possibilités offertes par chaque navigateur, une réponse technique intéressante permettant une meilleure indépendance vis-à-vis de technologies extérieures. Une proposition pour ce type de développement est présentée dans la section 4.3.3.1. Les évolutions concernant NPAPI et son support dans les divers navigateurs étant très récentes il s'agit là d'un nouveau problème dont l'impact n'a pas encore été étudié.

17. Elliptic-Curve Digital Signature Algorithm, signatures numériques basées sur courbes elliptiques

18. Netscape Plugin API, interface d'extensions pour navigateurs web

3 PLATE-FORME LOGICIELLE

Les deux protocoles décrits dans les chapitres précédents, ainsi que les travaux d'intégration de fonctions de sécurité au sein d'un SGED [1] présenté dans le chapitre 4, nécessitent un support complet d'outils cryptographiques et ce de façon portable afin d'être utilisable sur plusieurs architectures.

Tout au long de cette thèse ont été développés des outils permettant la mise en œuvre rapide des protocoles et prototypes issus de divers travaux de collaboration industrielle. Ces outils se basent sur une bibliothèque logicielle permettant d'une part d'utiliser des algorithmes cryptographiques et d'autre part d'intégrer de nouveaux algorithmes cryptographiques dans des prototypes existants. Au-delà d'un ensemble de primitives cryptographiques de base, des protocoles de plus haut niveau sont disponibles dans la bibliothèque et s'appuient sur ces primitives. Ces protocoles permettent à leur tour le déploiement d'outils de sécurité cryptographiquement sûrs en fournissant une abstraction masquant les algorithmes cryptographiques utilisés. Nous disposons ainsi d'une solution permettant plusieurs usages : l'appel direct d'algorithmes de bas niveau, l'exécution de protocoles de sécurité de haut niveau et l'utilisation d'outils de sécurité "abstraits" permettant de se détacher de l'utilisation directe des primitives cryptographiques. Dans ce chapitre sont décrits les solutions existantes les plus courantes, ainsi que la plate-forme de sécurité logicielle que nous avons développée appelée "Arcana".

3.1 Solutions cryptographiques courantes

Il existe plusieurs solutions permettant d'utiliser des outils cryptographiques. Les deux solutions les plus courantes sont l'utilisation de fonctions intégrées au système d'exploitation et l'utilisation de bibliothèques de cryptographie dédiées. Ces bibliothèques peuvent à leur tour utiliser les fonctions du système, mais sont plus complètes en termes de fonctionnalités et proposent des abstractions de haut niveau éliminant la dépendance aux fonctions du système d'exploitation.

1. Système de Gestion Électronique de documents, permettant la création, manipulation, et distribution de documents de nature variée

3.1.1 API systèmes

Tous les systèmes modernes fournissent une interface permettant d'utiliser des algorithmes cryptographiques. Ces interfaces sont présentées sous la forme de bibliothèques logicielles fournies par le système. Elles sont plus ou moins bien fournies en termes de fonctionnalités et sont spécifiques à chaque système d'exploitation. Nous parlerons ici des solutions fournies sur les systèmes MS Windows, OS X/iOS et Linux. Ces solutions particulières ne seront pas explorées en détail, car elles sont rarement utilisées directement. En effet, lors du développement d'une application, la portabilité est un élément important. Utiliser directement les services fournis par chaque système d'exploitation n'est pas compatible avec un développement portable. Il est plus courant de faire appel à une bibliothèque qui pourra éventuellement masquer ces appels au système derrière une interface générique. Nous passons en revue ici les différents types d'outils supportés par les différents systèmes d'exploitation.

3.1.1.1 Noyau Linux

Le noyau Linux fournit des services de cryptographie à travers une API [2] appelée "Crypto API". Ces interfaces sont principalement dédiées à la cryptographie pour les modules noyaux, mais sont également accessibles depuis l'espace utilisateur dans lequel s'exécutent les applications. Le noyau peut fournir les services suivants : chiffrement symétrique, chiffrement authentifié, MAC [3]/HMAC [4] et génération d'aléas. Les algorithmes disponibles pour chaque service peuvent varier d'un système à l'autre, en fonction du support matériel disponible. À titre d'exemple, un noyau Linux version 3.16 disposant de modules courants et s'exécutant sur un processeur Intel Core i7-2600k fournit l'algorithme AES [5] dans différents modes d'utilisation (XTS, LRW, CTR [6], CBC [7], GCM [8], etc). Les modes XTS et LRW sont principalement utilisés dans le chiffrement de supports de stockage, alors que

2. Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

3. Message Authentication Code, code d'authentification de message permettant d'en vérifier l'intégrité

4. Hash-based Message Authentication Code, code d'authentification de message basé sur des fonction de hachage

5. Advanced Encryption Standard, algorithme de chiffrement par bloc le plus utilisé de nos jours

6. Compteur, cet acronyme est utilisé pour identifier le mode compteur, un mode de chiffrement par blocs

7. Cipher Block Chaining, enchaînement de blocs chiffrés : mode d'utilisation consistant à faire intervenir pour chaque bloc chiffré l'état du bloc précédent, afin de construire une chaîne

8. Galois/Counter-Mode, mode de chiffrement par blocs intégrant un mécanisme d'authentification des données

les autres modes d'utilisation sont couramment utilisés pour la protection des communications, justifiant ainsi leur présence dans le noyau. En plus d'AES, ce noyau fournit un générateur pseudo-aléatoire et quelques fonctions de hachage (CRC32, MD5 [9] et SHA1 [10]).

Les fonctions cryptographiques ainsi fournies sont plutôt limitées et n'incluent aucune forme de chiffrement asymétrique ou de signature numérique. Cependant, les versions récentes peuvent aussi inclure un mécanisme de vérification de signature RSA [11].

3.1.1.2 Services cryptographiques OS X/iOS

Les systèmes OS X et iOS incluent des services cryptographiques fournissant l'ensemble des classes d'algorithmes courantes : chiffrement symétrique et asymétrique, échange de clés Diffie-Hellman, fonctions de hachage cryptographique, génération d'aléas. En plus des primitives cryptographiques, ces services cryptographiques fournissent d'autres éléments au niveau du système d'exploitation : gestion de trousseaux de clés, chiffrement de fichiers et communication sécurisée via TLS [12]. Ces services supplémentaires présentent à la fois un avantage et un risque de sécurité. D'un côté, l'intégration au niveau du système d'exploitation de ces services permet d'en faciliter l'utilisation par différentes applications. D'un autre côté, la centralisation d'éléments de sécurité peut amplifier l'impact d'une vulnérabilité au niveau du système d'exploitation. Ce point est d'autant plus important lorsqu'on se trouve sur des systèmes pour lesquels les mises à jour sont peu fréquentes, voire impossibles. C'est le cas notamment de terminaux mobiles utilisant le système iOS, souffrant d'un arrêt des mises à jours bien avant la fin de vie effective de ces équipements, provoquant l'utilisation de terminaux vulnérables.

Il n'y a pas d'information précise sur les algorithmes supportés par les services cryptographiques d'OS X et iOS ; toutefois le support de TLS indique qu'au minimum les algorithmes actuellement reconnus fiables tels qu'AES sont disponibles.

Dans l'ensemble, ces services permettent de mettre en œuvre l'ensemble des outils cryptographiques nécessaires à la conception d'applications sécurisées, mais lient l'application à la fois à une API non portable et à un cycle de mise à jour hors de contrôle du développeur d'applications.

9. Message Digest 5, fonction de hachage simple
10. Secure Hash Algorithm 1, fonction de hachage
11. Rivest Shamir Adleman, algorithme de chiffrement a clé publique parmis les plus utilisés au monde
12. Transport Layer Security, protocole de communication sécurisée remplaçant SSL

3.1.1.3 CryptoAPI Windows

Le système Windows fournit des outils cryptographiques regroupés sous le terme "CryptoAPI". Les classes d'algorithmes fournis sont, comme pour OS X, exhaustives : chiffrement symétrique et asymétrique, échange de clés, fonctions de hachage, génération d'aléas. Au-delà des primitives cryptographiques, cette API fournit également la gestion d'un magasin de certificats de clé publique géré par le système d'exploitation. Ce magasin de certificats sert, entre autre, pour vérifier les signatures numériques de composants du système, d'applications et l'établissement de connexions sécurisées par le protocole TLS. La centralisation d'éléments critiques a pour cause de démultiplier l'impact d'une vulnérabilité. En l'occurrence, ce magasin central de certificats peut facilement être altéré pour y introduire un certificat malicieux, permettant ensuite d'installer des applications malveillantes ou de tromper la sécurité du protocole TLS sans difficulté. Les algorithmes supportés de base par CryptoAPI sont très variés, s'y trouvent : DES, AES, RSA, SHA1, SHA2 [13], Diffie-Hellman et TLS.

Les services de CryptoAPI fournissent les fondations essentielles pour la construction de systèmes cryptographiques ainsi que des outils de plus haut niveau intégrant des protocoles directement utilisables. Le problème est cependant le même qu'avec les autres solutions : en utilisant directement ces services, l'application devient liée au système d'exploitation d'une part par l'utilisation d'une API non portable et d'autre part par la dépendance aux mises à jour du système d'exploitation lors de la découverte de failles de sécurité.

3.1.2 Bibliothèques de cryptographie

En dehors des API fournis par les différents systèmes d'exploitation, il existe des bibliothèques logicielles permettant d'utiliser des algorithmes et des protocoles cryptographiques. Parmi ces bibliothèques, certaines sont particulièrement répandues. La bibliothèque OpenSSL est probablement la plus utilisée, car en plus de fournir des outils cryptographiques complets, elle est l'implémentation du protocole TLS la plus répandue. Une autre bibliothèque logicielle courante est Bouncy Castle. Cette dernière est particulièrement utilisée dans des environnements de développement Java et Android. Nous allons voir ici le contenu de ces bibliothèques et leur cas d'utilisation courants.

3.1.2.1 OpenSSL

OpenSSL[20] est l'une des implémentations les plus courantes du protocole de communication TLS. Elle est utilisée par de nombreux serveurs, dont les serveurs

13. Secure Hash Algorithm 2, fonction de hachage succédant à SHA1, plus sûr que ce dernier

web Apache. En plus de supporter le protocole de communication TLS, OpenSSL fournit également des services de cryptographie complets. Elle est constituée de plusieurs niveaux d'abstractions, permettant d'un côté l'utilisation de fonctions avancées sans se soucier des détails d'implémentation et de l'autre de fournir le support de différents algorithmes au travers d'interfaces communes. L'architecture de la bibliothèque OpenSSL, illustrée par la figure 3.1 est la suivante : Il y a tout d'abord des primitives de bas niveau ; il s'agit d'implémentations directes d'algorithmes cryptographiques. Ensuite, on trouve plusieurs moteurs cryptographiques (appelés "engine"), implémentant certains algorithmes ou permettant d'utiliser les primitives de bas niveau. Ces moteurs cryptographiques permettent d'accéder à différentes classes d'algorithmes de façon uniforme. En montant d'un niveau se trouve une API appelée EVP [14], permettant de réaliser des opérations cryptographiques "complètes" sans se préoccuper des différents composants utilisés. En plus des outils cryptographiques, OpenSSL fournit aussi le support pour le protocole TLS au niveau applicatif ainsi que des outils périphériques nécessaires à son fonctionnement, comme des fonctions de traitement ASN1 [15] permettant de manipuler les certificats numériques utilisés par TLS.

API bas niveau : L'accès direct aux primitives cryptographiques fournies par OpenSSL est possible via une interface de bas-niveau. Elles ne permettent pas d'utiliser le système de moteur cryptographiques ; il s'agit d'un appel direct à une implémentation donnée. L'utilisation des primitives cryptographiques de cette façon permet la construction de protocoles plus avancés. La documentation d'OpenSSL déconseille cependant d'utiliser cette interface directement. En effet, en dehors de la création de nouveaux protocoles et de besoins de prototypage, il est très facile de commettre des erreurs de logique dans l'utilisation de primitives cryptographiques ; qu'il s'agisse de leurs interactions, de problèmes d'initialisation, ou de vérification incomplète de cas d'erreurs, la moindre inattention nuit à la sécurité du système. Aussi, cette interface n'est pas nécessairement stable d'une version à l'autre de la bibliothèque.

Moteurs cryptographiques : Les moteurs cryptographiques sont situés un niveau au-dessus des API bas niveau. Certains moteurs peuvent utiliser les implémentations bas niveau, alors que d'autres peuvent embarquer leur propre implémentation. Chaque moteur implémente un algorithme particulier et il est possible d'avoir plusieurs moteurs pour un même algorithme. Cela est utile dans plusieurs cas ; le plus courant étant la présence d'implémentation matérielle performante

14. Enveloppe, interface haut-niveau de la bibliothèque OpenSSL
15. Abstract Syntax Notation One, syntaxe utilisée pour décrire la structure d'éléments pour les communications dans des réseaux informatiques

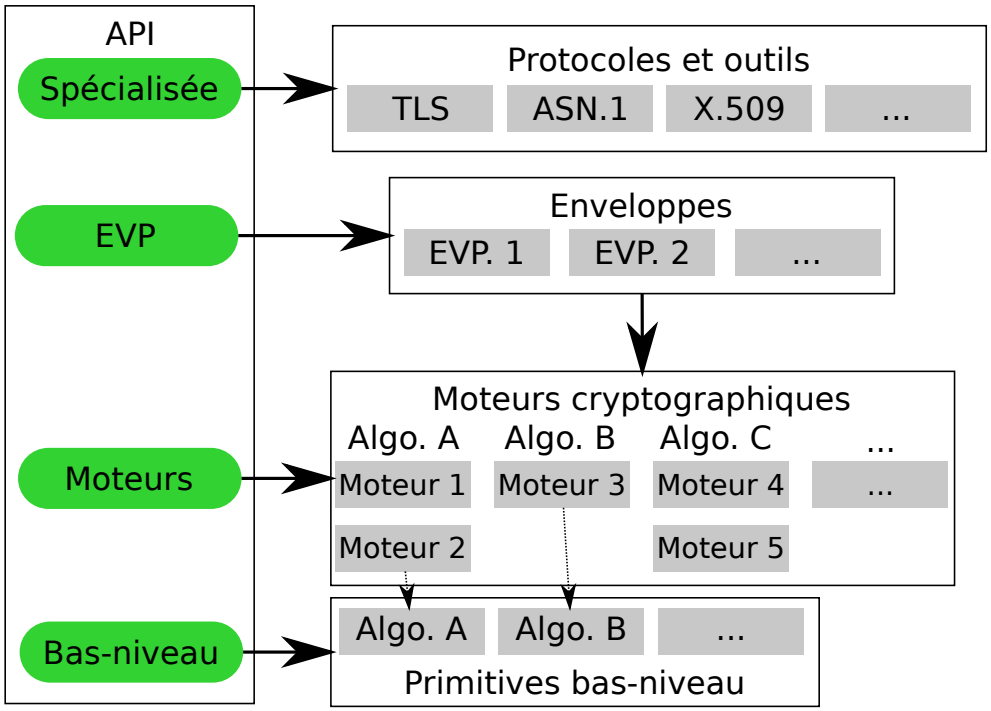

FIGURE 3.1 – Architecture de la bibliothèque OpenSSL

"Algo A,B,C..." représentent les classes d'algorithmes cryptographiques. "EVP.1,2,..." sont les différents types d'opérations cryptographiques haut-niveau. Les "moteurs" sont les différentes implémentations liés aux classes d'algorithmes.

pour un algorithme donné. Ainsi, en présence d'un générateur aléatoire matériel, la bibliothèque pourra remplacer le générateur pseudo-aléatoire logiciel par la version matérielle. Au niveau utilisation, il suffit de demander un moteur de génération d'aléa et le plus approprié est retourné. Cette abstraction permet de séparer l'implémentation effective d'un algorithme de son utilisation. Il est possible d'appeler directement un moteur, ou de choisir parmi les classes de primitives cryptographique pour sélectionner automatiquement le moteur par défaut. Cependant, il est conseillé d'utiliser l'interface de plus haut niveau EVP : l'enrobage des algorithmes cryptographique dans des séquences "vérifiées" fournies par les EVP réduit le risque d'erreur dans leur utilisation.

Interface haut-niveau EVP : L'utilisation des fonctions de haut-niveau de la bibliothèque OpenSSL permet de minimiser les risques liés à l'utilisation directe des primitives de l'API bas niveau. Cette interface présente directement des "groupes" d'algorithmes utilisables, en masquant les éléments "mécaniques" à gérer lors d'opérations cryptographiques. Par éléments mécaniques, nous parlons ici de pa-

ramètres tels que le vecteur d'initialisation ou la méthode de bourrage (padding). En effet, en prenant l'exemple d'un système de chiffrement symétrique utilisant un mode avec initialisation, les bonnes pratiques imposent d'utiliser un vecteur d'initialisation aléatoire, différent à chaque fois. En utilisant les primitives de bas niveau, cela signifierait l'initialisation d'un générateur d'aléa, le tirage d'un vecteur d'initialisation et le passage de cette valeur à l'algorithme de chiffrement. Il faudrait par ailleurs gérer la transmission de cette valeur au destinataire du message afin qu'il puisse réaliser le déchiffrement de son côté. Ces opérations sont "mécaniques". En utilisant une EVP, la question du tirage initial du vecteur d'initialisation ne se pose pas car l'EVP masque cet élément et l'intègre directement au résultat de l'opération de chiffrement. Les EVP peuvent donc être vues comme une gestion automatique et correcte de l'utilisation de primitives cryptographiques pour les opérations courantes.

Interfaces spécialisées : En plus des fonctions cryptographiques, la bibliothèque OpenSSL fournit également le support de trois éléments pour l'établissement de connexion sécurisées et authentifiées : ASN1, les certificats X.509 et TLS. Le support d'ASN1 est nécessaire pour manipuler les certificats au format X.509. Ces derniers sont le standard en matière de certification d'identité ; ils contiennent, entre autre, des clés publiques, des informations d'identité associées et un mécanisme d'authentification qui permet de valider leur authenticité. Le format de ces certificats et leurs transmissions sont régis par le standard ASN1. La bibliothèque OpenSSL fournit des fonctions permettant de créer, d'accéder et de manipuler des éléments ASN1. Notons toutefois qu'on utilise rarement ces fonctions directement, mais plutôt au travers d'une interface dédiée à la manipulation de certificats X.509. Cette interface permet directement d'accéder et de manipuler les champs d'un certificat sans se soucier de la représentation sous-jacente. Pour finir, OpenSSL fournit une interface permettant de mettre en œuvre des connexions sécurisées par TLS. Cette dernière permet d'établir une connexion sécurisée sur différents supports. Les étapes d'authentification sont gérées de manière semi-transparente en définissant un certain nombre d'objets pour la connexion (notamment un magasin de certificats servant à authentifier les serveurs).

OpenSSL est donc une bibliothèque de cryptographie très complète en termes de primitives et de types d'algorithmes supportés. Elle est aussi l'une des meilleures solutions pour bénéficier du protocole TLS. Cependant, certains problèmes d'utilisation existent. L'exemple le plus récent est la faille de sécurité "Heartbleed"[15, 27]. Si la présence d'erreurs dans un logiciel n'est pas rare, cette dernière a eu des conséquences très importantes. Cette faille permettait à un attaquant de lire une partie de la mémoire du serveur employant le protocole TLS. Malgré l'absence de risque de type élévation de privilège, accès aux périphériques de stockage, ou accès aux

autres processus du système, cela pose quand même un problème majeur : le processus vulnérable (le serveur) est logiquement en possession au minimum de la clé privée censée assurer l'authenticité de la connexion. La récupération d'une clé privée permet à un attaquant de réaliser diverses attaques en usurpant l'identité du serveur sans risque de détection. En plus d'avoir des conséquences graves, cette faille s'est avéré quasiment indétectable : un attaquant pouvait réaliser autant de tentatives d'accès à la mémoire qu'il le souhaitait sans laisser de traces, la fonctionnalité exploitée n'étant que rarement journalisée. Enfin et c'est là le point qui nous intéresse, cette faille a existé pendant une période assez longue sans être détectée ou publiée. Cela signifie que potentiellement de nombreuses entités ont pu la découvrir et l'exploiter pendant une longue période (aujourd'hui, certains serveurs accessibles depuis Internet y sont toujours vulnérables). Combiné à la popularité de la bibliothèque OpenSSL, cela signifie qu'une part très importante des serveurs utilisant TLS ont potentiellement divulgué leurs données sensibles, dont leurs clés privées.

La bibliothèque OpenSSL combine des propriétés dangereuses : très grande popularité, très grande base de code et nombreuses évolutions au fil du temps. La grande popularité n'est pas un défaut en soi, mais en termes d'implémentation d'outils de sécurité, cela a deux effets. D'un côté les "bonnes pratiques" peuvent facilement être étendue à tous les utilisateurs de la bibliothèque mais d'un autre côté, les failles potentielles et vulnérabilités impactent automatiquement un grand nombre de systèmes. Le problème du volume de la base de code d'OpenSSL et de ses évolutions successives est plus complexe. Au fil du temps, la bibliothèque a dû s'adapter aux nouveaux algorithmes et nouveaux outils cryptographiques. Malheureusement malgré un nombre important de précautions et des conventions de codage strictes, des erreurs subtiles peuvent apparaître. Lorsque ces erreurs ne sont pas détectées par les différents outils et tests mis en place, tel que cela a été le cas pour Heartbleed, seule une analyse "humaine" peut y parvenir. Lorsque la base de code devient trop importante, il n'est pas réaliste de réaliser une telle analyse sur chaque élément et chaque interaction possible entre les différents composants de la bibliothèque. C'est pourquoi avec le temps des erreurs subtiles peuvent donner lieu à des problèmes graves. À la lumière de problèmes récents ayant suivi la découverte de Heartbleed, il y a eu un effort important de nettoyage du code d'OpenSSL. Le projet reste cependant très volumineux et seuls des audits complets et réguliers pourront réduire le risque d'une faille de cette importance.

3.1.2.2 Bouncy Castle

La bibliothèque Bouncy Castle[51] est une bibliothèque de cryptographie complète initialement entièrement écrite en Java. Même s'il existe maintenant une version C#, l'essentiel de cette description est basée sur la version Java. En dehors

des spécificités de ce langage (principalement les discussions au sujet de JCE [16]), le contenu de cette section s'applique aux deux versions. Elle présente l'avantage de ne pas dépendre de binaires externes et de permettre un fonctionnement hors de l'architecture cryptographique de Java.

Java dispose d'une API appelée JCE qui fournit des services cryptographiques à partir de différents "fournisseurs". Ces fournisseurs peuvent fournir un ou plusieurs algorithmes cryptographiques et s'utilisent tous au travers d'une interface commune spécifique pour chaque classe d'algorithmes (par exemple les chiffrements symétriques ou les signatures numériques). Cependant, il existe des restrictions sur ces fournisseurs, qui peuvent en fonction de configurations locales limiter le niveau de sécurité utilisable. Un exemple est la limitation des tailles de clé à 128 bits pour la plupart des algorithmes de chiffrement. L'utilisation d'une implémentation indépendante telles que Bouncy Castle permet de se dégager de ces restrictions sur tous les systèmes.

Architecture : Les fonctions cryptographiques sont accessibles directement via une interface bas-niveau. Cette interface fournit l'accès à l'ensemble des algorithmes actuels en matière de cryptographie. Au-dessus sont construits des outils rendant ces fonctions de base plus accessibles aux développeurs. En particulier, un "fournisseur" JCE qui permet d'intégrer les fonctions de Bouncy Castle dans des applications Java standards. De nombreux autres outils cryptographiques sont supportés. Parmi ceux-là, nous pouvons citer le protocole TLS ainsi que sa variante DTLS [17], destinée aux réseaux UDP [18] (et, pour les mêmes raisons qu'OpenSSL, le support des objets ASN1 et des certificats X.509). Les autres éléments notables sont le support de l'enrobage S/MIME [19][54] et du protocole OCSP [20][59]. Le premier est notamment utilisé pour l'encapsulation sécurisée de pièces jointes dans les systèmes de messagerie électronique. Le second est un mécanisme de révocation des certificats numériques utilisé par TLS. La plupart de ces fonctions "spécialisées" sont construites au-dessus de l'interface bas-niveau ou en tirant parti d'éléments disponibles de façon standard dans l'environnement Java.

16. Java Cryptographic Extension, implémentation d'outils cryptographiques du langage Java
17. Datagram Transport Layer Security, une version de TLS fonctionnant sur des réseaux en mode datagramme tels qu'UDP
18. User Datagram Protocol, protocole de communications rapide n'ayant ni garantie de délivrance des messages, ni de garantie d'ordre d'arrivé
19. Secure/Multipurpose Internet Mail Extensions, enrobage sécurisé notamment utilisé pour les pièces jointes de messagerie éléctronique
20. Online Certificate Status Protocol, mécanisme de contrôle de la révocation de certificats numériques remplaçant les CRL

Interface bas-niveau : L'interface bas-niveau constitue les briques élémentaires de Bouncy Castle. Cette dernière, entièrement écrite en langage Java, est utilisable sur tous les systèmes pour lesquels une machine virtuelle Java est disponible. Cela va des systèmes fortement contraints utilisant l'environnement J2ME (téléphones mobiles, "set-top box", ordinateurs de bords,...) jusqu'aux systèmes informatiques classiques. Cette large disponibilité, indépendante des fonctions de cryptographie initialement disponible sur ces supports, a grandement contribué à la popularité de Bouncy Castle. En effet, la présence des interfaces JCE dans un environnement Java n'implique pas la présence de fournisseurs de base. Même sur un système "classique", il n'y a pas de garantie que le support cryptographique disponible par défaut aille au-delà d'un ensemble minimal de fonctions.

Fournisseur JCE : Même si l'accès aux fonctions cryptographiques par l'interface bas-niveau est possible, il est préférable d'utiliser le fournisseur JCE. Ce dernier permet de masquer certains automatismes liés à l'utilisation de fonctions cryptographiques. En effet, cette interface de plus haut niveau permet de traiter des objets plus abstraits au niveau de l'application et masque les détails mécaniques tels que les vecteurs d'initialisations en les choisissant automatiquement et en les incluant dans le résultat des opérations. De plus, cela permet d'appliquer un certain nombre de bonnes pratiques et d'éviter des écueils courant qui peuvent arriver en manipulant directement les primitives cryptographiques. Ce fournisseur JCE est également plus fonctionnel que les primitives de base, car il permet de supporter des formats de données standardisés, que ce soit pour la manipulation des objets cryptographiques comme les clés privées/publiques/secrètes, ou les éléments d'intégrité.

Intégration dans les systèmes Android : Une version spéciale de la bibliothèque Bouncy Castle a été intégrée dans certaines versions du système d'exploitation Android, dédié aux appareils mobiles (notamment smartphones et tablettes). Dans la mesure où les applications conçues pour le système Android sont principalement développées dans un environnement Java, l'intégration de Bouncy Castle permet de fournir des services cryptographiques sans dépendre de bibliothèques natives. Le problème avec cette intégration, comme c'est souvent le cas avec l'intégration d'outils de sécurité au niveau d'un système d'exploitation, est que la version présente sur les systèmes Android ne peut pas être mise à jour. De plus, la simple présence de cette bibliothèque au niveau du système empêche une application d'embarquer sa propre version afin de se maintenir à jour. C'est pour cela que Spongy Castle[66] a été créé. Il s'agit d'une version reconditionnée de Bouncy Castle qui permet son utilisation en parallèle avec la version fournie par le système.

En termes de fonctionnalités, Bouncy Castle est une bibliothèque logicielle

très complète. Elle supporte l'ensemble des algorithmes actuellement utilisés, ainsi que de nombreux protocoles et standards. De plus, elle apporte une suite d'outils cryptographiques dans des environnements où il n'y a peut-être pas le support adéquat en standard, notamment sur des appareils fortement limités en ressources. Cependant, cette bibliothèque est exclusivement limitée aux environnements de développements Java et C#. Cette restriction l'exclut de toute utilisation par des applications ayant un fort besoin de portabilité, car elle dépend de la présence d'un environnement Java ou C#. L'autre point négatif de cette bibliothèque est lié aux performances. Sur des systèmes où il est possible d'utiliser une bibliothèque de sécurité utilisant du code natif, on observera de meilleures performances avec celle-ci qu'avec une bibliothèque purement Java.

3.2 Plate-forme Arcana

La plate-forme de sécurité Arcana regroupe un ensemble d'outils pour la mise en œuvre des fonctions cryptographiques nécessaires à la réalisation d'applications sécurisées. Parmi ces outils se trouvent une bibliothèque logicielle de cryptographie, un ensemble de courbes elliptiques dédiées à des usages sécurisés, des protocoles avancés et des outils permettant d'utiliser cette plate-forme dans différents environnements. Nous allons présenter ici les motivations qui ont donné lieu à la création de ces outils, leur rôle et les algorithmes fournis. À l'exception des travaux sur l'arithmétique des courbes elliptiques et de la base de courbes qui ont fait l'objet de développements annexes[33], les travaux décrits dans cette section représentent le travail accompli au cours de cette thèse.

3.2.1 Présentation

Cette section apporte une présentation générale du projet. Les motivations à sa création sont décrites plus longuement au paragraphe 3.2.4.1.

La plate-forme de sécurité Arcana s'articule autour d'une bibliothèque logicielle de cryptographie appelée Arcana-Cryptobox. Celle-ci a été créée afin de fournir l'ensemble des algorithmes nécessaires à la conception d'applications sécurisées. Les objectifs lors de la création de cette bibliothèque étaient les suivants : disposer d'outils maîtrisés et vérifiables, fournir l'ensemble des primitives nécessaires à la mise en place de systèmes sécurisés et disposer de ces outils dans un maximum d'environnements d'utilisation.

Pour le premier objectif il a fallu implémenter l'ensemble des algorithmes et primitives au sein de la bibliothèque à partir de rien. Il n'y a donc aucune dépendance avec d'autres bibliothèques de sécurités. De plus, le code de la bibliothèque Arcana-Cryptobox est aussi concis que possible afin d'en permettre une analyse

rapide. Les primitives fournies correspondent à l'état de l'art en matière d'algorithmes cryptographiques. Nous disposons ainsi d'un ensemble complet de classes de primitives et chacune de ces classes implémente les algorithmes recommandés de nos jours en matière de cryptographie. Enfin, la partie logicielle a été développée avec comme objectif une grande disponibilité sur un nombre important d'architectures et d'environnements. Au travers de règles de codage strictes et de l'utilisation d'un minimum d'éléments spécifiques à un environnement donné, nous avons obtenu une bibliothèque logicielle hautement portable.

La plate-forme de sécurité Arcana intègre tous les composants élémentaires permettant de développer des systèmes cryptographiques, mais aussi des éléments plus haut-niveau comme des protocoles avancés comme des protocoles de transfert et d'échange de clés et des protocoles de protection des communications.

Les opérations basées sur des fonctions de chiffrement asymétriques s'appuient sur l'utilisation de la cryptographie sur courbes elliptiques. Ces dernières permettent d'apporter un meilleur niveau de sécurité pour des tailles d'éléments inférieures comparé aux algorithmes classiques.

La disponibilité sur plusieurs architectures et environnements, la présence d'algorithmes de pointe et l'approche concise du développement permettant un audit rapide font de cette bibliothèque logicielle un outil particulièrement intéressant dans deux domaines : la recherche et le prototypage. Tout d'abord, puisqu'il est aisé d'examiner, modifier ou ajouter des algorithmes dans la base de code, l'analyse et la conception de nouveaux éléments peuvent être rapidement mis en place. Nous pouvons ainsi aisément tester les performances et les interactions entre différents algorithmes. Pour les mêmes raisons et grâce à son interface de programmation simple, la bibliothèque Cryptobox permet un prototypage rapide qu'il s'agisse de nouveaux protocoles ou de systèmes de sécurité plus avancés, y compris pour différents environnements de développement.

Le tableau 3.1 résume les différentes classes d'algorithmes, les algorithmes et les protocoles supportés par Arcana-Cryptobox. La bibliothèque supporte l'ensemble des classes d'algorithmes cryptographiques courants. Par ailleurs, tous les algorithmes présentés sont disponibles sur tous les environnement de développement pour lesquels la bibliothèque est disponible : il n'y a pas de différence entre les versions.

3.2.2 Bibliothèque logicielle

Le développement de la partie logicielle a commencé en 2009 sous l'impulsion de Robert Rolland, qui a amené l'idée de disposer d'un ensemble d'outils cryptographiques connus et maîtrisés. Dès lors, les travaux d'implémentations et d'amélioration des fonctions de la bibliothèque Cryptobox ont commencés. L'ensemble des développements de la bibliothèque, à l'exception de la section 3.2.6 sur les opé-

Classes d'algorithmes	Algorithmes
Fonctions de hachage et HMAC	CRC24*, SHA1*, famille SHA2, Skein
Chiffrement symétrique Modes d'utilisation	famille AES, famille Twofish CBC, CTR, GCM
Chiffrement asymétrique (signature et encapsulation de clés)	ECDSA (voir ECDB, section 3.2.6)
Génération d'aléas	Basé sur SHA2 et initialisé par le générateur du système
Échange de clés	Encapsulation (KEM) basé sur ECIES ECDH Échange basé sur STS
Dérivation de clés	HKDF[36], PBKDF2[35]

TABLE 3.1 – Algorithmes supportés par la bibliothèque Arcana-Cryptobox

* supporté pour des raisons historiques, non recommandé dans un contexte sécurisé

rations arithmétiques des courbes elliptiques, ont été réalisé au cours de cette thèse à la suite d'un développement initial durant lequel la liste des outils supportés a été définie. Parmi les premiers développements ont eu lieu la création d'une API partagée par les différents algorithmes et la mise en place des protocoles d'échange de clés sûrs. Ces développements avaient pour objectif de stabiliser la base de code initiale afin de permettre par la suite des développements plus complexes.

3.2.2.1 API unifiée

La création d'une API unifiée avait pour but de permettre l'utilisation des différents algorithmes sous-jacents sans être lié à une implémentation particulière. Cela permet par exemple de développer un système de chiffrement et de choisir lors de l'exécution l'algorithme effectif. Par ailleurs, masquer les détails d'implémentation derrière une interface stable a permis par la suite de modifier la structure interne de la bibliothèque sans avoir à modifier les programmes l'utilisant. C'est pour cela que l'ensemble du code a été remanié. Nous sommes passés d'une situation initiale où il était possible de réaliser des appels "bas-niveau" référençant directement les différentes implémentations à une interface haut-niveau. Cette dernière présente un squelette partagé par tous les algorithmes du même type et sélectionne l'implémentation adaptée dynamiquement lors de l'utilisation.

La figure 3.2 présente une vue d'ensemble de cette API. La situation initiale (non représentée) permettait d'appeler directement les fonctions du niveau "primitives". L'API conserve le même niveau de fonctionnalités, mais permet de sélectionner à la volée un algorithme spécifique.

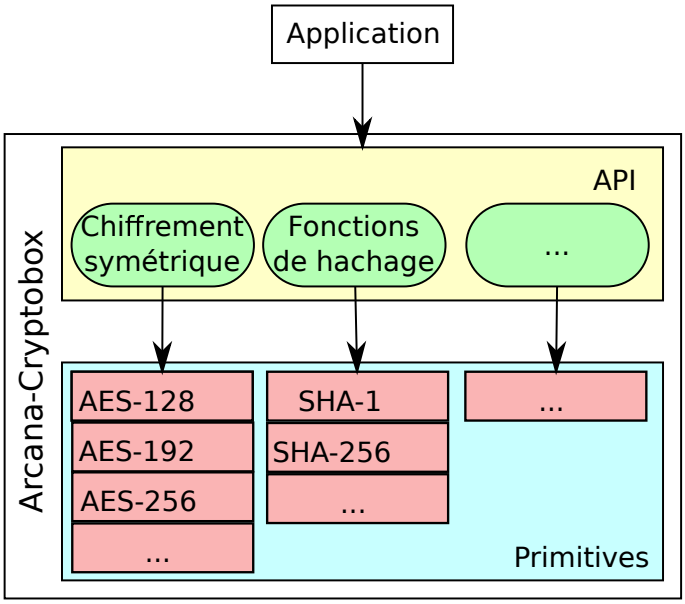

FIGURE 3.2 – API unifié Cryptobox

3.2.2.2 Échange de clé sûrs

Un protocole d'échange de clés permet à au moins deux entités de se mettre d'accord sur un secret partagé via des communications non confidentielles. L'exemple classique est le protocole Diffie-Hellman : il permet à deux entités n'ayant eu aucun contact préalable de déterminer un secret commun en échangeant des éléments publics sur un canal non sécurisé. Cet échange est basé sur le problème du logarithme discret. Le secret commun partagé par les deux entités peut par la suite servir à des opérations cryptographiques, comme du chiffrement symétrique, afin de fournir une communication confidentielle. En lui-même, l'échange de clés Diffie-Hellman a de très bonnes propriétés. Cependant, l'échange n'est pas authentifié, ce qui le rend vulnérable aux attaques de type MitM [21]. Il sert cependant de base à de nombreuses versions authentifiées de façons différentes. Il faut toutefois noter une différence avec l'algorithme de base de Diffie-Hellman ; ici les opérations sont réalisées sur des points de courbes elliptiques. C'est pour cela que l'on parle d'ECDH [22]. Cette différence permet de conserver un haut niveau de sécurité tout en réduisant le volume de données à échanger.

L'extension des protocoles d'échange de clés pour y intégrer des variantes sûres a aussi eu lieu lors du développement initial. Cela a permis la mise en place de variantes authentifiées du protocole Diffie-Hellman. L'objectif était de disposer ra-

21. Man in the Middle, l'homme au milieu : intercepter et/ou modifier les communications entre deux systèmes
22. Elliptic-Curve Diffie-Hellman, Diffie-Hellman basé sur courbes elliptiques

pidement de solutions basées sur ECDH afin de combiner les avantages des opérations sur courbes elliptiques, la fiabilité du protocole Diffie-Hellman et la résistance aux attaques de type MitM, problème majeur de l'échange de clés Diffie-Hellman. Pour répondre à ce besoin, des variantes de ECDH ont été implémentées permettant d'ajouter un élément secret supplémentaire à l'échange, nécessitant le partage initial d'un secret par un autre canal que celui utilisé par l'échange de clés. Ce secret supplémentaire n'est utilisable que dans certains cas, par exemple des échanges pour lesquels les deux systèmes sont physiquement proches. L'utilisation d'un secret partagé tel qu'un mot de passe permet alors de compléter un échange de clés avec une sécurité maximale sans risque d'attaque de type MitM. Le secret en question peut être échangé par des moyens de communications moins sensibles.

En parallèle à ces extensions du protocole Diffie-Hellman une autre approche a été développée en la forme d'un protocole basé sur STS. Ce protocole, basé sur l'algorithme ECDH, intègre un mécanisme d'authentification basé sur des signatures numériques. Ce mécanisme remplace l'utilisation d'un "secret" dans les variantes de Diffie-Hellman et permet à l'échange de clés d'aboutir sans risque d'attaque de type MitM. Cependant, pour fonctionner il nécessite qu'au moins l'un des intervenants de l'échange de clés ait un moyen de valider l'identité de l'autre. La validation d'une des deux extrémités lors d'un échange de clés est une technique similaire à ce qui est couramment utilisé pour sécuriser les communications numériques, notamment dans le cas du protocole TLS et de notre protocole SVC [23].

En plus de l'authentification, la base de code a été remaniée pour permettre l'établissement d'un échange de clés entre plus de deux entités. Cette évolution, possible avec l'algorithme Diffie-Hellman de base, prend également en compte les options d'authentification de l'échange et permet de négocier un secret partagé pour un groupe d'individus.

3.2.3 Architecture de la plate-forme Arcana

L'architecture de la plate-forme de sécurité Arcana est représentée dans la figure 3.3. L'élément central est la bibliothèque cryptographique Arcana-Cryptobox. Cette dernière est appuyée par d'autres éléments, en particulier un ensemble d'interfaces permettant d'utiliser la bibliothèque et la base de courbes elliptiques. La raison de ce découpage est de centraliser les éléments les plus critiques dans un "noyau" de primitives cryptographiques. Ce noyau est ensuite partagé par les différentes versions de la bibliothèque. Ces versions sont conçues pour fonctionner sur différents systèmes. Les opérations manipulant des courbes elliptiques utilisent les informations contenues dans la base de courbes Arcana-ECDB pour déterminer les

23. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

paramètres des courbes elliptiques. Enfin, les interfaces d'accès aux fonctions fournies par la bibliothèque Cryptobox permettent d'utiliser des conventions d'appels similaires depuis différents environnements de développements.

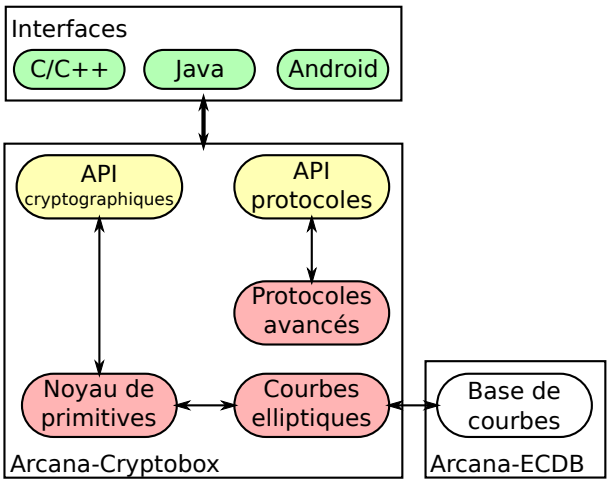

FIGURE 3.3 – Structure de la plate-forme de sécurité Arcana

Le développement a principalement eu lieu au niveau du noyau de primitives et des protocoles intégrés à la bibliothèque logicielle. L'architecture choisie pour Arcana permet de mutualiser ces développements. Cela accélère la conception et l'ajout de nouveaux éléments, ainsi que la maintenance et la mise à jour d'éléments existants. En effet, ces modifications peuvent immédiatement s'appliquer aux différents systèmes et interfaces. Du point de vue de l'utilisation, la présence d'interfaces similaires permet à l'utilisateur de maîtriser rapidement les fonctions de la bibliothèque, même si plusieurs environnements de développements différents sont utilisés.

3.2.4 Évolutions de Arcana-Cryptobox

3.2.4.1 Motivations et influences

L'ensemble du projet Arcana a été influencé par différents éléments au fil des années. La motivation initiale au projet Arcana s'est trouvé dans la nécessité de disposer d'outils cryptographiques qui regroupent plusieurs propriétés : ils devaient être utilisables rapidement, fiables, vérifiables et modifiables a loisir afin de permettre des ajouts et des évolutions fréquentes. Pour répondre à ces critères, le développement d'un outil en interne était une solution appropriée. Cela a permis de maîtriser l'outil et de pouvoir rapidement vérifier et valider son contenu. Disposer ainsi d'un outil entièrement contrôlé permet également de l'étendre sans difficulté, qu'il s'agisse d'ajouter de nouveaux algorithmes standards ou de créer

des prototypes de recherches. Tous ces points ont donné lieu à la création du projet Arcana et de sa bibliothèque logicielle Cryptobox.

L'intégration d'améliorations et d'optimisations, de nouveaux algorithmes et de nouveaux protocoles s'est faite au fil de travaux de recherche, de projets variés ou encore d'évolutions des outils cryptographiques. Ces changements ont été réalisés en plus de la maintenance régulière de la base de code de la bibliothèque logicielle. Les ajouts suivants ont été motivés par des projets externes : ajout des modes d'utilisations GCM et CBC, intégration de la fonction de hachage Skein, développement du protocole de communication sécurisée SVC et création d'une interface pour système Android.

Mode d'utilisation GCM : Le mode d'utilisation GCM[42] est appliqué aux systèmes de chiffrement par blocs et permet d'effectuer en même temps les opérations de chiffrement et les opérations de contrôle d'intégrité et d'authenticité. La partie compteur du GCM permet de tirer profit du parallélisme des opérations sur les différents blocs. La méthode utilisée pour l'intégrité et l'authentification des messages donne elle aussi lieu à des optimisations intéressantes : la possibilité de pré-calculer certains éléments et la possibilité d'appliquer ce calcul en parallèle à différents blocs. De plus, le pré-calcul de tables de résultats permet d'accélérer les opérations de calcul d'intégrité et apporte un important gain de performances sans affaiblir la sécurité du système. Ce pré-calcul dépend des paramètres de chiffrement utilisés et doit être réalisé en temps réel à chaque initialisation de chiffrement. Cette optimisation est également compatible avec la parallélisation des opérations d'intégrité, rendant le mode d'utilisation entièrement parallélisable, aussi bien pour la partie chiffrement que pour la partie authentification. Des travaux en ce sens ont été réalisés en marge du projet Arcana, mais ont été éclipsés par d'autres types d'optimisations au sein de la bibliothèque. Lorsqu'il a été intégré au projet Arcana-Cryptobox, le mode d'utilisation GCM était un nouveau mode d'utilisation peu répandu. Il est désormais utilisé dans de nombreux standards comme TLS, SSH [24], IPSec [25], etc.

Mode d'utilisation CBC : Le mode d'utilisation CBC est, comme le GCM, appliqué aux systèmes de chiffrement par blocs. Ce mode présente un avantage par rapport aux modes basés sur un système de compteur que sont GCM et CTR : il peut se resynchroniser automatiquement en cas de perte d'informations. Un usage courant de ce mode est la transmission de contenu multimédia en temps réel pour lesquels il n'est pas rare d'avoir des pertes de paquets. Dans une telle situation, il

24. Secure Shell, protocole de communication sécurisée permettant notamment l'ouverture d'une invite de commande sur un système distant

25. Internet Protocol Security, protocole de chiffrement des échanges au niveau IP

serait nécessaire de transmettre un numéro de séquence si on utilisait les modes basés sur des compteurs. Le mode CBC peut quant à lui automatiquement reprendre le déchiffrement à partir du moment ou deux paquets consécutifs sont traités. En ce qui concerne la plate-forme de sécurité Arcana, le mode CBC a été ajouté récemment pour permettre d'implémenter un protocole d'anonymisation conçu dans un projet de recherche industriel. La flexibilité de la bibliothèque Cryptobox a permis de rapidement intégrer ce nouveau mode d'utilisation et de valider immédiatement son utilisation dans un prototype du protocole d'anonymisation lié au projet de recherche.

Fonction de hachage Skein : La fonction de hachage Skein[19] est un autre exemple de cas où la plate-forme Arcana a permis l'évaluation rapide d'un nouvel algorithme cryptographique. Skein était l'un des algorithmes finalistes du concours initié par le NIST [26] afin de trouver un remplaçant à la fonction SHA2. Même si au final un autre algorithme a été choisi, Skein présentait (et présente) des intérêts de recherche. Dans sa spécification est décrite une approche parallèle basée sur l'utilisation d'arbres binaires avec une version spécialisée de la fonction de compression. De plus, les performances de l'algorithme, que ce soit dans une utilisation séquentielle classique ou dans une utilisation parallèle, méritaient d'être évaluées. À cette fin, l'algorithme Skein dans sa version séquentielle a été intégré à la bibliothèque Cryptobox. Cette intégration a permis la comparaison des performances des différentes variantes de Skein avec les autres fonctions de hachage ainsi que de point de comparaison avec une implémentation parallèle. Cette implémentation parallèle a été réalisée par une étudiante au sein du groupe de recherche et a donné lieu à une publication[17]. Au final, l'implémentation séquentielle est venue renforcer les algorithmes disponibles de la plate-forme de sécurité Arcana. Même si Skein n'a pas été choisi pour devenir à terme SHA3 (c'est un autre algorithme du nom de Keccak qui l'a été[8, 45]), il y a eu tout de même de nombreux travaux autour de cet algorithme, que ce soit des implémentations sur FPGA [27][39, 69], sur composants dédiés[68], logicielles[38, 17] ou sur smart cards[22]. Au vu des travaux réalisés autour de cet algorithme, il n'est pas impossible qu'il trouve sa place au côté de SHA3 dans des applications futures.

3.2.4.2 Fiabilité

Il existe des différences importantes entre le développement d'éléments logiciels courants (non sécurisés) et le développement d'éléments logiciels destinés à

26. National Institute of Standards and Technology, institut Américain faisant référence notamment dans le domaine des algorithmes cryptographiques

27. Field-programmable gate array, réseau reprogrammable de portes logiques permettant d'implémenter des fonctions à la demande

des usages sécurisés. Dans un contexte non sécurisé, les applications et autre bibliothèques doivent garantir que leur comportement est correct dans deux situations principales : lors d'entrées correctes et lors d'entrées incorrectes. Dans un contexte sécurisé il faut aussi tenir compte de la possibilité de saisie d'entrée malveillante par l'utilisateur. La présence d'un paramètre incorrecte dans une application peut mener à des comportements inattendus. Dans le cas d'une application développée rigoureusement, ces comportements seront vraisemblablement rapidement détectés et n'auront pas de conséquences sur le long terme. La problématique n'est pas la même dans une application où la sécurité est critique. En effet, nous pouvons nous trouver dans des situations où un attaquant produirait volontairement des entrées incorrectes. Ces entrées ayant pour but de provoquer un comportement inhabituel de l'application qui ne sera pas détecté. Dans ce contexte, il ne suffit donc pas de détecter des cas d'erreurs accidentels, mais bel et bien de verrouiller au maximum les paramètres pour empêcher tout comportement malveillant d'avoir des conséquences sur la sécurité du système.

Afin d'éviter les problèmes inhérents à la croissance de la base de code, un mélange de solutions dites "classiques" et avancées ont été utilisées. Ces solutions ont pour but de minimiser le risque d'introduction de vulnérabilités lors de la modification ou l'ajout de fonctionnalités au sein de la bibliothèque Cryptobox. Les techniques employées sont les suivantes : convention de codage stricte, vecteurs de tests, tests de non régression, analyse statique et analyse dynamique. Chacun de ces éléments permet d'assurer un niveau de qualité minimal de la base de code et des développements qui y sont faits. En complément à ces outils, des audit régulier seront nécessaires dès lors que la base de code de la plate-forme sera assez importante pour ne plus permettre un parcours rapide. En effet, les procédures d'analyse automatiques ne sont pas, pour l'instant, capables de détecter toutes les formes de problèmes. C'est en particulier ce qui s'est passé pour la faille "Heartbleed" dans la bibliothèque OpenSSL[70], qui a pu passer au travers de toutes ces vérifications.

Conventions de codage : Les conventions de codage couvrent de nombreux aspects du développement, allant de la façon de nommer les variables aux règles de découpage du code. En général, elles servent à assurer la cohérence d'un projet sur lequel différentes personnes contribuent. Dans le cadre d'une bibliothèque cryptographique, ou de tout élément logiciel ayant pour finalité des fonctionnalités de sécurité, ces conventions doivent aussi minimiser les risques liés au développement. Dans le cas présent, les aspects relatifs à la fiabilité sont des règles strictes sur la gestion des erreurs, la vérification des paramètres depuis les appels de l'API, la restriction à certains types de données et l'utilisation minimale d'éléments dynamiques. La gestion des erreurs est un élément critique dans toute application

de sécurité : toute condition d'erreur donne lieu à l'interruption d'une opération sensible. Dans la mesure où tous les protocoles et autres outils de sécurité sont construits sur des éléments plus primitifs, eux-mêmes construits depuis d'autres et ainsi de suite, une convention de détection et de propagation des erreurs cohérentes est nécessaire. La vérification des paramètres d'appel s'inscrit également dans cette gestion des erreurs. Par exemple, l'utilisation d'un paramètre invalide par l'application appelante peut fausser l'ensemble des tests qui auraient été effectués sur le reste du code. Intercepter ces erreurs dès leur point d'entrée dans la bibliothèque de cryptographie permet d'éliminer ce risque et d'éviter des comportements imprévus. Ces derniers, qu'il s'agisse de renvoyer un résultat incorrect ou de faire avorter l'application, sont bien entendu inacceptables dans tous programme, mais sont encore plus inacceptables dans le cas d'un composant lié à la sécurité. En effet, une application utilisée dans un contexte sécurisé peut se voir être la cible d'attaques volontaires ayant pour but de déstabiliser le système. La mise en place de vérifications à l'exécution permet de réduire ce risque. Les autres éléments cités, la restriction à certains types de données et l'utilisation minimale d'éléments dynamiques, sont des règles de programmation qui ont pour but d'éviter les erreurs d'inattention. L'utilisation d'un nombre restreint de types de données permet de clairement spécifier leur comportement à chaque point où ils sont utilisés et évite les confusions accidentelles. La minimisation des éléments dynamiques favorise notamment l'utilisation de variables automatiques gérées par le compilateur et empêche des problèmes courants de gestion de mémoire, dont la source est généralement une erreur humaine, pouvant causer des problèmes de stabilité. Les éléments de convention de codage relatifs aux interfaces publiques et à la sécurité du code sont présentés dans l'annexe B.

Vecteurs de test et tests de non régression : L'utilisation de vecteurs de test a pour but principal de vérifier le comportement correct des implémentations. Pour la plupart des primitives "bas-niveau" des vecteurs de test standards sont fournis par différentes sources comme l'auteur original ou les organismes de standardisation. Pour les protocoles et autres outils n'ayant pas de réels vecteurs de test, principalement parce qu'ils sont probabilistes, deux approches sont possibles : soit nous pouvons produire des résultats en fixant la source d'aléas afin de s'assurer que des versions ultérieures produisent toujours les mêmes résultats, soit nous pouvons valider les résultats en implémentant deux façons différentes de réaliser un calcul. Enfin, pour certains algorithmes réaliser des tests sur un ensemble minimal d'entrées n'a pas de sens. Dans ces cas-là, on construit des vecteurs de tests très large, combinant chacune des valeurs possibles pour chaque paramètre (ou de façon plus réaliste uniquement des valeurs représentatives comme des bornes), que l'on utilise pour confirmer que les résultats sont constants d'une version à

l'autre du logiciel. Les vecteurs de test "construits" de cette façon incluent également des paramètres d'entrées invalides afin de confirmer la bonne gestion des cas d'erreurs. Ces tests sont intégrés dans une suite permettant de valider l'ensemble des algorithmes et sont intégrés à un système de gestion de version pour valider les changements avant de les intégrer dans un dépôt central. Ils sont structurés de façon à suivre l'architecture globale de la bibliothèque Cryptobox, par classes d'algorithmes, puis par algorithmes individuels. Cela permet d'ajouter aisément de nouvelles implémentations et les jeux de test appropriés.

Analyse statique : Des outils d'analyse statique du code comme OCLint[53] permettent de détecter des problèmes potentiels depuis le code source d'une application. Ce genre d'analyse permet d'éliminer un grand nombre de fautes d'inattention et de "mauvaises pratiques" pouvant compliquer la compréhension et amener à des erreurs lors de modifications ultérieures de façon automatisée. Une analyse statique du code source combiné à l'utilisation d'une convention de codage stricte permet de s'assurer que le code produit respecte un niveau maximum de cohérence, de clarté et de correction. Cela assure que le code reste maintenable, une nécessité pour les changements et les éventuels audits qui pourraient le concerner.

Analyse dynamique : Même si la mise en place, à l'exécution, de mécanismes de contrôle peut potentiellement augmenter la stabilité de la bibliothèque cryptographique et par extension de l'application l'utilisant, cela n'est pas nécessairement souhaitable. En effet, ce genre de vérifications aurait un impact très négatif sur les performances des outils de sécurité. Nous pouvons heureusement limiter les risques de problèmes lors de l'exécution en suivant les conseils précédents, notamment le contrôle des paramètres et en validant les tests de couverture et de non régression avec une analyse dynamique. La solution retenue ici est donc d'exécuter les différents jeux de tests unitaires au travers d'un programme de diagnostic. En s'assurant que ces tests couvrent bien l'ensemble des cas d'appels par l'application utilisant notre bibliothèque, la validation de leur exécution est représentative du comportement général de la bibliothèque de sécurité. Pour ces validations l'outil Valgrind[12] est utilisé. Ce dernier permet de vérifier entre autre les allocations et libérations de mémoire, l'utilisation de mémoire non initialisée et les accès à des zones mémoire incorrectes. Ce genre d'erreurs peuvent se glisser de façon inaperçue dans une application et n'avoir aucune conséquence pendant un certain temps, mais causer des erreurs de façon aléatoire par la suite.

3.2.5 Développements

Le développement de la plate-forme de sécurité logicielle Arcana a fait l'objet d'un travail continu durant cette thèse. Pour cela, plusieurs composants ont dû être étendus, améliorés, ou intégralement créés. L'ensemble des développements, qu'il s'agisse du composant central Arcana-Cryptobox ou des outils annexes, a été fait en suivant les règles décrites précédemment. Tout d'abord, l'élément central de la plate-forme de sécurité Arcana, la bibliothèque logicielle, a fait l'objet d'ajouts et d'optimisations. Les ajouts d'algorithmes et de fonctionnalités ont été détaillés précédemment ; nous parlons ici des différentes optimisations mises en œuvre pour améliorer les performances de la bibliothèque. Une classe spéciale d'optimisation a été réalisée sous la forme d'un module de gestion avancée de la mémoire ayant pour rôle d'une part d'améliorer les performances et d'autre part de sécuriser les données sensibles. Le dernier point d'évolution de la bibliothèque logicielle est la modification du code pour le rendre portable. Grâce à des modifications spécifiques et à des contraintes d'écritures adaptées, l'ensemble de la bibliothèque logicielle peut désormais être compilée pour fonctionner sur différents systèmes et d'architectures.

Au-delà des travaux de développement centrés sur le code de la bibliothèque logicielle, d'autres travaux ont été réalisés pour améliorer la plate-forme de sécurité Arcana. Une fois le composant central rendu disponible sur différents systèmes, il a fallu créer des interfaces adaptées permettant d'utiliser cet outil depuis différents environnements de développement au travers de différentes interfaces.

3.2.5.1 Optimisations

La base de code initiale de la Cryptobox a été conçue pour être la plus fonctionnelle possible, sans nécessairement privilégier les performances. Dans un deuxième temps, afin d'améliorer les performances, des optimisations ont été réalisées. En plus des optimisations générales "classiques" incluant des changements mineurs du code, d'autres changements plus spécifiques ont été réalisés pour des algorithmes particuliers : le mode d'utilisation GCM et le chiffrement par bloc AES.

Il est important de noter que ces optimisations ont été réalisées en gardant à l'esprit les autres points de développement, dont la portabilité de la bibliothèque. Pour que cette dernière reste possible, il était essentiel que les optimisations réalisées disposent toujours d'une implémentation de secours pouvant se substituer à la version optimisée dans les cas où cette dernière ne peut pas être utilisée. De plus, dans la mesure où la portabilité cible non seulement des architectures différentes, mais aussi des systèmes ayant des ressources différentes, une optimisation particulière ne peut pas mener à un usage excessif des ressources, notamment de la mémoire disponible pour l'exécution. Afin de satisfaire ce point, dans certains

cas des versions avec une faible empreinte mémoire sont proposées, permettant d'utiliser la bibliothèque sur des systèmes fortement restreints en ressources.

Optimisations générales : Une grande partie du code initiale de la Cryptobox était constituée d'implémentations directes des algorithmes de référence en matière de cryptographie. Cela avait pour conséquence de faibles performances. Après une étape initiale visant à uniformiser les interfaces des différentes implémentations, un travail d'optimisation général du code a été entrepris. La part la plus importante a été l'élimination de nombreuses opérations redondantes de copies mémoire et la réorganisation du code afin d'éliminer des manipulations inutiles. Par ailleurs, des algorithmes "naïfs" ont été remplacés par des versions plus adaptés à un traitement informatique. Un exemple de ces changements est le traitement des données en entrée d'une fonction de hachage : l'ensemble des fonctions de hachage utilisées actuellement fonctionnent par blocs, nécessitant de découper leur entrée en blocs ayant une taille fixe. Cette opération peut être réalisée de plusieurs façons : la consommation du message en entrée, octet par octet, afin d'alimenter une mémoire tampon ayant une taille adéquate, ou le traitement direct des blocs composant le message. Cette distinction n'est pas présente lorsque l'on décrit un algorithme purement théorique, car dans la description théorique le temps de transfert est rarement pris en compte : on ne considère pas alors l'intérêt de transférer des données par blocs plutôt que par octets. Au contraire, en pratique transférer des données octets par octets peut être beaucoup plus coûteux que de réaliser des transferts par blocs ; l'approche théorique naïve est donc bien moins efficace et doit être remplacé par un algorithme adapté. Des remplacements de ce genre ont permis d'augmenter considérablement les performances de la bibliothèque Cryptobox. En particulier, l'exemple du découpage en entrée des fonctions de hachage a permis d'obtenir des performances à peu près sept fois supérieures à l'implémentation initiale.

GCM version logicielle : L'algorithme GCM peut être vu comme deux composants opérants ensemble : le mode CTR classique et une fonction de calcul d'intégrité appelée GHASH permettant d'authentifier un message chiffré, comme illustré sur la figure 3.4. Sur cette figure, la partie supérieure représente les opérations usuelles du mode CTR alors que la partie inférieure représente le calcul de la fonction GHASH. Cette dernière produit une étiquette (TAG) permettant de confirmer l'authenticité du message lors de sa réception. Cette étiquette dépend des paramètres de sécurité, de données d'authentification additionnelles et de la clé du chiffrement. Elle ne peut donc pas être falsifiée sans la connaissance de ces éléments. En termes d'optimisation, deux approches non mutuellement exclusives sont possibles pour cet algorithme. Tout d'abord, il est possible d'étendre le

parallélisme du mode CTR au calcul de l'empreinte d'authenticité. Cela permet d'effectuer simultanément les calculs sur plusieurs blocs afin d'accélérer le temps de calcul global sur les systèmes disposant de plusieurs unités de calculs. Une autre optimisation porte sur la fonction de multiplication impliquée. Cette dernière fait intervenir entre autre de la clé de chiffrement, ce qui signifie qu'elle est différente pour chaque opération. Cependant, pour une opération de chiffrement donné, il est possible de pré-calculer des tables de résultats de cette multiplication. L'optimisation porte donc sur le pré-calcul de ces tables, afin d'obtenir un temps d'exécution global inférieur à la version non optimisée. Cette optimisation est décrite dans la spécification du GCM[42]. L'implémentation de cette optimisation est disponible dans la bibliothèque Cryptobox. Puisqu'il s'agit d'une implémentation logicielle écrite en langage C elle est disponible de manière portable sur l'ensemble des systèmes supportant la plate-forme de sécurité Arcana. Comme indiqué précédemment cette optimisation fait partie des optimisations augmentant considérablement l'empreinte mémoire de l'algorithme ; c'est pourquoi une version alternative existe pour les systèmes restreints en ressources. Moins performante, elle permet l'utilisation sur des systèmes pour lesquelles la mémoire disponible à l'exécution est très restreinte.

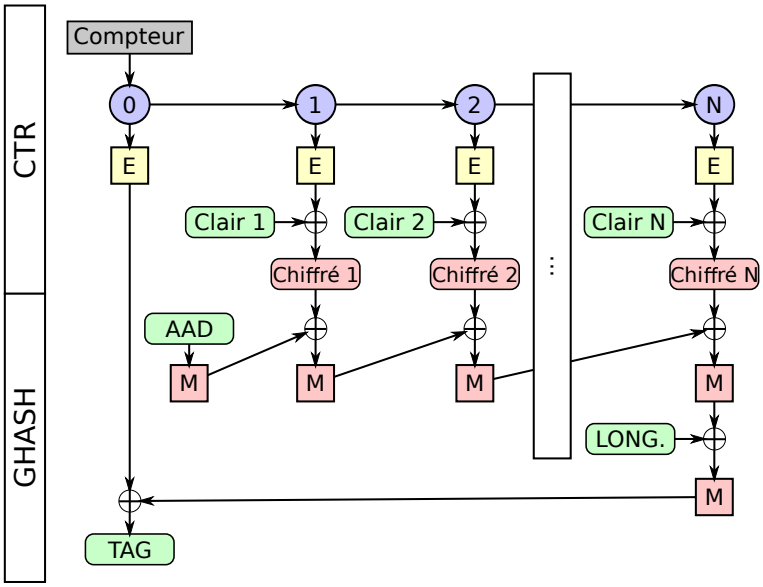

FIGURE 3.4 – Fonctionnement du mode de chiffrement par blocs GCM

Sur cette figure N est le nombre de blocs du message original, AAD des données à authentifier non confidentielles, E la fonction de chiffrement (par ex. AES), M la fonction de multiplication de GHASH, LONG. la longueur du message et de l'AAD et TAG représente l'empreinte d'authenticité.

GCM version matérielle : Il existe une extension au jeu d'instructions x86 appelé CLMUL [28][26]. Cette dernière a été proposée début 2008 par Intel et permet d'accélérer les opérations de multiplications sur les corps finis. Ces accélérations sont notamment utiles pour le calcul de la fonction GHASH du GCM basée sur ces multiplications. S'agissant d'une extension au jeu d'instructions classique x86, jeu d'instructions de quasiment l'ensemble des processeurs d'ordinateurs personnels de nos jours, un programme compilé utilisant cette extension pourra s'exécuter aussi bien sur un système disposant de ces instructions que sur un système n'en disposant pas. Bien évidemment, si l'on fait appel à ces instructions sur une machine pour laquelle elles ne sont pas disponibles, cela va provoquer une erreur d'exécution et causer l'arrêt brutal du programme. C'est pourquoi le code de la bibliothèque Cryptobox réalise une vérification à l'exécution afin de déterminer si le système courant supporte ces instructions. Si c'est le cas, une version optimisée tirant profit de ces instructions est utilisée. Dans le cas contraire c'est l'implémentation logicielle optimisée décrite précédemment qui prend sa place. Ce choix d'implémentation réalisé à l'exécution permet d'obtenir les meilleures performances possibles sur une large gamme de matériels sans avoir à déployer des versions différentes de la bibliothèque logicielle Cryptobox. La détection de la présence d'une extension du jeu d'instructions se fait via l'instruction CPUID qui est toujours disponible. Cette dernière renvoie (entre autre) un champ de bits permettant de déterminer le support par le processeur courant d'instructions optionnelles.

Support matériel pour AES : De la même façon que pour les instructions CLMUL, un ensemble d'instructions spécifiques pour l'implémentation d'AES a été proposé en 2008[25] sous le nom AES-NI (pour "New Instructions"). Ces instructions permettent d'effectuer l'ensemble des opérations d'AES de façon accélérée : la génération des clés de tours et le chiffrement pour chaque tour (y compris le dernier tour, différent des autres). Comme pour l'extension CLMUL, les instructions AES-NI sont une extension au jeu d'instructions x86 et leur présence sur un système donné est optionnelle. La solution consistant à tester à l'exécution le support d'un ensemble d'instructions est donc utilisé ici de la même façon que pour CLMUL dans l'optimisation du mode GCM. Comme pour les optimisations matérielles du mode GCM, en cas de non-support des instructions spécifiques AES-NI l'implémentation logicielle est utilisée.

28. Carry-Less Multiplication, une extension au jeu d'instruction x86 permettant d'accélerer les multiplications sur des corps finis

3.2.5.2 Gestion mémoire

La gestion de la mémoire dans le contexte d'une bibliothèque de cryptographie présente deux aspects majeurs. Concernant les performances, les algorithmes utilisés doivent manipuler des quantités importantes de données. Qu'il s'agisse de mémoire tampon pour les opérations de chiffrement et déchiffrement, ou les opérandes de calculs pour les opérations manipulant des points de courbes elliptiques, l'allocation dynamique de mémoire est nécessaire. Même si cet aspect dynamique est minimisé autant que possible grâce à l'allocation de mémoire sur la pile de façon non dynamique, il reste des cas d'usage pour lesquels ce remplacement n'est pas possible. Le problème de performance lié à l'utilisation de mémoire dynamique est le suivant : les allocations et libérations de mémoire doivent passer par des appels systèmes qui ralentissent l'exécution du programme. Sur la plupart des systèmes, ces allocations dynamiques fonctionnent par "pages", des sections de mémoire d'une taille donnée, afin de limiter les appels systèmes nécessaires. Malheureusement, la taille des données manipulées dans le contexte d'outils cryptographiques peut dépasser la taille de ces pages et entraîner malgré tout des appels systèmes réguliers. En fonction des usages liés à cette mémoire dynamique, nous pouvons donc nous retrouver dans des situations alternant entre un grand nombre d'allocations et de libérations, causant des allers-retours incessants entre le programme et le système. Dans ces conditions, il est intéressant d'étudier le ou les modèles d'utilisation de la mémoire dynamique afin de proposer une approche plus adaptée que la gestion générique consistant à simplement fournir des pages mémoire à l'application lors des demandes et les rendre au système lors de leur libération.

Pour le problème de sécurité, l'ensemble des données, qu'il s'agisse d'élément publics comme les valeurs chiffrées ou les éléments secrets comme les clés de chiffrement transitent par la mémoire du système. Dans le cas de données sensibles il est souhaitable que leur présence en mémoire soit la plus brève possible. Pour s'en assurer un système d'allocation et de contrôle de la mémoire doit être mis en place. Il permet de spécifier qu'une zone mémoire contient des données sensibles. Lorsque ces zones mémoire sont rendues au système, c'est-à-dire lorsqu'elles sont libérées par l'application, leur contenu est effacé afin de ne pas causer de fuite d'information. De cette façon, une donnée sensible n'existe en mémoire que le temps de son utilisation et ne persiste pas.

L'utilisation d'heuristiques spécifiques permet à la gestion dynamique de la mémoire de mieux répondre à l'exécution des fonctions de la bibliothèque en améliorant les performances sans augmenter l'empreinte mémoire de façon significative. Pour allouer dynamiquement de la mémoire, l'ensemble de la bibliothèque utilise une fonction similaire à la fonction `malloc()` standard. Cette dernière permet d'obtenir un bloc de mémoire en spécifiant sa taille comme paramètre. La fonction de remplacement au sein de la bibliothèque Cryptobox, nommée `xmalloc()`, prend

en plus un second paramètre indiquant le scénario d'utilisation de la mémoire allouée. Les scénarios suivants ont été implémentés : mémoire tampon pour données non-sensibles, mémoire tampon pour données sensibles, opérandes de calcul pour courbes elliptiques et contexte d'utilisation.

Tampon pour données non-sensibles : Il s'agit de la mémoire utilisée pour conserver les données non sensibles, comme le résultat d'une opération de chiffrement ou d'une fonction de hachage. Ces données sont considérées comme non sensibles, car leur divulgation ne met pas a priori en danger les éléments secrets. Ce type de mémoire ne nécessite pas d'opérations de nettoyage particulières lorsqu'elle est rendue au système. Ces opérations de nettoyage ayant un impact négatif sur les performances, il est préférable de s'en dispenser lorsqu'elles n'apportent pas d'avantages significatifs. Ce scénario d'allocation dynamique de mémoire se contente donc de garder une trace de la taille de chaque bloc alloué, afin de pouvoir satisfaire plus rapidement les demandes d'allocation suivantes. Si l'application alloue et libère régulièrement deux tampons de N octets, le mécanisme de gestion de mémoire va rapidement déterminer que l'application a un besoin de $2N$ octets. Pour cela, on utilise une variable retenant la quantité totale de mémoire allouée, ainsi que des compteurs retenant le nombre de blocs alloués, et un bref historique de ces opérations. Lors d'une opération d'allocation, l'état du gestionnaire de mémoire est mis à jour immédiatement. Lors d'une opération de libération, l'état du gestionnaire de mémoire est poussé dans l'historique. Dès que le nombre de libération dépasse le nombre d'allocation successive précédentes, le gestionnaire considère que l'application change de comportement, et commence à réduire la quantité de mémoire qu'il conserve lors des opérations de libérations suivantes. Cette information permet de décider lors de la libération d'un tampon mémoire s'il est pertinent de le "conserver" au niveau de l'application pour une réutilisation ultérieure, ou de le rendre au système. Dans le premier cas, une allocation de mémoire dynamique ultérieure conduira à la réutilisation directe de cet élément conservé sans avoir à passer par un appel système. Pour une utilisation régulière (typiquement une application effectuant en boucle les mêmes opérations), la conservation de mémoire est optimale. Pour des utilisations plus atypiques, le gestionnaire de mémoire ne conservera qu'une quantité minimum de mémoire "superflue" pour une durée limitée. L'algorithme s'ajuste ainsi automatiquement à la demande pour ne pas bloquer une quantité trop importante de mémoire, décidée en fonction de l'utilisation.

Tampon pour données sensibles : Les tampons mémoire utilisés pour conserver des données sensibles sont dédiés à la conservation d'éléments secrets comme les clés de chiffrement ou les données avant leur chiffrement. L'heuristique générale

pour leur allocation et libération est la même que pour les données non sensibles. La différence se situe dans le traitement du contenu de ces tampons. Dès leur libération, qu'ils soient rendus au système ou conservés au niveau de l'application, leur contenu est effacé par écrasement. Une séquence d'octets aléatoires y est écrite plutôt qu'une simple mise à zéro. En effet, la mise à zéro des données sensibles peut donner une information à un attaquant : l'analyse de la mémoire rendue au système par une application sécurisée après que son exécution se soit terminée pourrait, entre autre, mener à la découverte des emplacements susceptibles de contenir des données sensibles en identifiant les suites suspectes de zéro. Ce genre d'informations, dans le cas d'applications ayant une exécution relativement régulière, pourrait permettre de récupérer ces parties sensibles du jeu de mémoire d'une application durant son exécution via d'autres vulnérabilités.

Opérandes de calcul : Lors des opérations de calcul sur les points de courbes elliptiques, la bibliothèque a besoin d'allouer un certain nombre d'opérandes de façon dynamique pour conserver des résultats intermédiaires. Ces opérandes ont des tailles très similaires, mais dépendent des définitions de courbes utilisés. De plus, en fonction d'optimisations dans les algorithmes de calcul eux-mêmes, il est difficile de savoir à l'avance le nombre de ces allocations ainsi que leur taille. Pour répondre à ce problème, le fonctionnement suivant a été défini : les blocs mémoire alloués dynamiquement pour les opérandes de calcul sont conservés lors de leur libération, créant ainsi une réserve de zones mémoire pour ces calculs. La taille de ces zones mémoire converge rapidement vers la taille maximum requise pour les calculs effectués par l'application. Le nombre de ces zones mémoire correspond au schéma d'utilisation de l'application. En effet, typiquement une application effectue la même série d'opérations régulièrement. Dans ce cas, on atteint rapidement un équilibre sur le nombre et la taille des opérandes de calculs. Lorsque le gestionnaire de mémoire détecte plusieurs cycles d'allocations/libérations pendant lesquels certaines zones mémoire conservées ne sont plus sollicitées, ces dernières sont progressivement rendues au système afin de ne pas bloquer inutilement les ressources.

Contextes d'utilisations : Les contextes d'utilisation contiennent toutes les informations pratiques nécessaires à l'utilisation des "outils" de la bibliothèque. Par exemple, le contexte d'utilisation d'un chiffrement symétrique contiendra des informations concernant le volume de données chiffrées, les algorithmes utilisés, les contextes associés de chaque algorithme et la direction des opérations (chiffrement ou déchiffrement). Ces données sont relativement peu sensibles et en temps normal ne font pas l'objet de cycles d'allocation et libération réguliers. Ils conservent donc un fonctionnement identique au mécanisme standard de gestion dynamique de

mémoire.

Ces différentes gestions en fonction du contexte de l'utilisation de la mémoire dynamique ont permis d'améliorer les performances globales de la bibliothèque Cryptobox. Les gains en temps d'exécution obtenus initialement avec l'approche "naïve" peuvent atteindre 8 % en comparaison avec l'utilisation directe des fonctions standards d'allocation mémoire. En utilisant la gestion des différents scénarios décrit dans cette section, les gains en temps d'exécution sont de l'ordre de 12 % sur l'exécution d'un jeu de test équivalent à celui utilisé pour l'évaluation précédente. L'analyse de l'empreinte mémoire est plus compliquée à évaluer, car le jeu de test ne correspond pas à des cas typiques d'applications ayant en général une unique finalité. Nous pouvons noter toutefois qu'à l'exécution le gestionnaire de mémoire dynamique libère plus de mémoire que l'approche naïve, pour de meilleures performances. Cela indique que les heuristiques utilisées permettent de conserver au niveau de l'application la mémoire nécessaire à son fonctionnement tout en monopolisant moins de ressources systèmes.

L'autre côté de l'implémentation d'un système de gestion dynamique de la mémoire est la gestion des appels concurrents. Même si le fonctionnement interne des algorithmes gère les problèmes de parallélisation là où c'est applicable, il est possible que l'application utilisant la bibliothèque Cryptobox produise elle-même des appels en parallèle aux fonctions cryptographiques fournies, causant ainsi des appels parallèles imprévus des fonctions de gestion de mémoire.

Situation initiale : Au début de la thèse, la bibliothèque Arcana-Cryptobox utilisait un gestionnaire de mémoire dynamique appliquant une approche similaire aux différents "scénarios". L'approche considérée permettait une augmentation arbitraire du nombre de scénarios à l'exécution. Le support d'appels concurrents était réalisé en multipliant le nombre de "scénarios" par le nombre d'appels concurrents effectués par l'application. Cette approche permettait d'obtenir de bonnes performances et semblait par construction éliminer les risques d'interblocages. Cependant, des défauts existaient dans cette implémentation. Le problème majeur résidait dans le fait que les gestionnaires assignés à chaque fil d'exécution (thread) étaient indépendants les uns des autres. Si cela élimine effectivement les problèmes d'interblocage, cela pose des contraintes masquées sur les appels de la bibliothèque. Il faut en effet que les allocations et libérations se fassent dans le même thread. Ce genre de contraintes va à l'encontre du besoin initial de permettre le support transparent des appels dans un contexte de concurrence.

Pour résoudre ce problème, la séparation entre les threads a été modifiée en profondeur. La conservation des zones mémoire au niveau de l'application lors d'une libération est bien gérée individuellement pour chaque thread, mais les paramètres sur le nombre de zones mémoire disponibles et les autres valeurs utilisées par les

heuristiques décrites précédemment sont partagées. Cette nouvelle approche permet d'affiner plus efficacement les statistiques d'allocation mémoire dans le cas d'une application répartissant ses opérations sur plusieurs threads de façon dynamique. En plus de fournir une analyse statistique plus cohérente, cela élimine le problème suivant : un thread pouvait décider individuellement de conserver un grand nombre de zones mémoire alors qu'un autre thread pouvait en avoir besoin. Le partage des statistiques permet d'empêcher cette situation de dégénérer. En effet, si lors de la libération la zone mémoire est conservée indépendamment pour chaque thread, lors d'une allocation il est possible d'interroger les réserves des autres thread au besoin. Cette séparation entre libération et allocations permet de réduire l'impact des mécanismes de synchronisations entre thread. Elle favorise les performances lors de cycles réguliers d'allocation et de libération au sein d'un même thread, tout en permettant, si besoin est, de transférer des zones mémoire réservées d'un thread à l'autre afin de s'adapter aux besoins de l'application. Même si au final nous avons réintroduit des éléments de synchronisation, synonyme d'impact négatif sur les performances, ces dernières restent similaires aux résultats précédents, car la synchronisation n'a lieu que dans de rares occasions.

L'implémentation de ces concepts, leur amélioration et l'adaptation à des problèmes plus larges ont fait l'objet d'un important travail de développement tout au long de la thèse. Ces réalisations sont basées sur des développements initiaux qui visaient à mettre en place la différenciation des différents scénarios de gestion dynamique de la mémoire sans s'intéresser finement à leurs usages. La contribution ici est l'utilisation d'heuristiques précises pour la gestion dynamique de la mémoire et la prise en charge correcte et efficace des appels concurrent.

3.2.5.3 Portabilité

Le terme portabilité signifie ici la possibilité de mutualiser au maximum une même base de code pour différentes architectures. La bibliothèque Arcana-Cryptobox est entièrement écrite en langage C. Ce choix permet de supporter autant d'architectures qu'en supporte les compilateurs C. Au-delà de la disponibilité de versions compilées pour différentes architectures, des interfaces dédiées rendent accessibles les fonctions de la bibliothèque aux différents environnements de développement. Un exemple concret est la présence d'une version de la plate-forme Arcana pour systèmes Android. Cette version est constituée d'un binaire pour les architectures Android (sur architecture ARM et x86) et d'une interface Java permettant à une application de réaliser des appels aux fonctions fournies par le binaire.

Cependant, l'utilisation du langage C n'est pas une garantie en soi que le code sera portable. Il existe des contraintes à prendre en compte pour s'en assurer : l'absence de dépendances externes, la validation du code utilisant des manipulations au niveau des bits, l'isolation des optimisations matérielles et l'isolation du

code spécifique à un système d'exploitation donné.

Limitation des dépendances externes : La présence de dépendances avec des bibliothèques extérieures peut compromettre la portabilité d'un projet. Il faut donc se restreindre autant que possible à l'utilisation de fonctions présentes dans le standard du langage C. Cette restriction permet de s'assurer que la présence d'un compilateur conforme au standard choisi permettra de compiler l'ensemble de la bibliothèque (le standard actuellement utilisé par la bibliothèque Cryptobox est le C99[32]). Dans le cas de la plate-forme Arcana, il y a une dépendance à une bibliothèque extérieure : la bibliothèque de manipulation de grands nombres GMP [29][23]. Elle est utilisée pour la partie arithmétique sur les courbes elliptiques. L'utilisation de cette bibliothèque peut réduire les possibilités de portabilité de la bibliothèque Cryptobox ; toutefois la bibliothèque GMP est elle-même disponible sur un large éventail d'architectures et n'est donc pas un facteur limitant. Au final, en dehors de cette dépendance la quasi-totalité du code constituant la bibliothèque Cryptobox est entièrement portable.

Manipulations binaires : L'utilisation d'algorithmes implémenté en langage C standard permet de limiter au maximum les dépendances sur la gestion des données par le matériel. Cependant, certaines opérations manipulent l'information à un niveau plus bas que l'abstraction présentée par le langage C avec ses types de données. En particulier, des manipulations au niveau des bits sont courantes. Ces manipulations peuvent avoir un résultat différent en fonction de la représentation native des informations. Il est ici question de problèmes d'ordre des octets (en anglais "endianness") qui peuvent altérer la représentation en mémoire des types de données constitués de plusieurs octets. Un exemple est l'utilisation d'un entier de 32 bits, soit quatre octets, pour représenter une donnée. La manipulation octet par octet d'un tel entier aura un résultat différent selon l'ordre des octets du système. Les implémentations dépendantes d'un ordre d'octets particulier doivent donc être adaptées pour fonctionner correctement sur des architectures différentes. Ce genre de changement peut avoir des conséquences importantes sur les performances, car certains algorithmes sont particulièrement optimisés pour tirer profit d'un ordre d'octets donné.

Optimisations matérielles : Les optimisations matérielles réduisent de façon inhérente la portabilité de la bibliothèque, car elles sont dépendantes d'une architecture donnée. Cependant, leur usage permet d'augmenter considérablement les performances. Le compromis choisi ici est de limiter les optimisations matérielles en fonction de deux critères : la présence d'une implémentation de secours

29. GNU Multiple Precision Arithmetic Library, bibliothèque de calcul sur des grands nombres

purement “C” permettant de supporter les autres architectures et, le cas échéant, la possibilité de déterminer à l'exécution la présence d'optimisations matérielles quand une même architecture peut les supporter de manière optionnelle. C'est cette seconde possibilité qui est appliquée dans les optimisations décrites pour le mode GCM et l'algorithme AES. En effet, ces optimisations matérielles sont disponibles en option sur les systèmes x86 et un binaire prévu pour cette architecture doit pouvoir s'exécuter à la fois sur des systèmes disposant de ces instructions spéciales et sur des systèmes n'en disposant pas. La première possibilité visant à fournir des implémentations spécifiques à certains matériels est plus générique et s'applique principalement aux optimisations utilisant des langages de bas niveau. Un exemple d'algorithme concerné par ce point est le mode CTR classique. Ce dernier nécessite d'itérer une fonction sur un tampon mémoire découpé en blocs. Les performances obtenues en réalisant une implémentation en langage d'assemblage de cet algorithme sont supérieures à l'implémentation en langage C. Cependant, l'utilisation d'une implémentation en langage d'assemblage limite son utilisation à une architecture donnée. C'est pourquoi lors de la compilation, une implémentation en langage C est nécessaire afin de fournir les mêmes fonctionnalités sur les architectures ne disposant pas d'une version optimisée spécifique.

Code spécifique à un système d'exploitation : Certaines fonctions nécessaires au bon fonctionnement de la bibliothèque Cryptobox ne sont pas portables. Un exemple est l'initialisation du PRNG [30]. Un PRNG cryptographiquement sûr permet de fournir des séquences aléatoires “imprévisibles”, c'est-à-dire qu'en connaissant la séquence de bits déjà produite, il est impossible de déterminer le prochain bit qui sera fourni par le PRNG. Un PRNG est dit “pseudo-aléatoire” car la séquence produite est en fait déterministe et est basée sur une valeur initiale ayant un rôle de “générateur”. La sécurité du PRNG dépend de l'aléa de ce générateur. Il est très difficile pour un système informatique de produire de “vraies” valeurs aléatoires. Pour s'en approcher, différentes approches sont utilisées. Elles mettent en œuvre la collecte d'un grand nombre d'événements provenant de sources différentes et les agrègent afin de produire des valeurs aléatoires. Cette collecte d'événements et cette agrégation sont réalisées par le système d'exploitation et ne sont pas directement accessibles par une application ou une bibliothèque logicielle. Pour profiter de cette génération d'aléa il est nécessaire de passer par des interfaces spécifiques aux différents systèmes d'exploitation. Sur un système Windows, la récupération de valeurs “aléatoires” se fait au travers de l'API CryptoAPI du système. Pour des systèmes de type UNIX tels que Linux ou OS X, la récupération de cet aléa système se fait au travers d'un fichier spécial nommé `/dev/random` qui fournit à chaque lecture des octets aléatoires. La conséquence de cette diffé-

30. Pseudo-Random Number Generator, générateur pseudo-aléatoire

rence fait qu'il est nécessaire au niveau de la bibliothèque Cryptobox de disposer d'éléments propres à chaque système pour fournir ces services. En effet, le fonctionnement du PRNG présent dans la bibliothèque Cryptobox dépend de cette initialisation et de la fourniture régulière de nouvelles valeurs "réellement" aléatoires. C'est pour cela que certaines fonctions dont la fonction d'initialisation du PRNG possèdent du code strictement non portable. Les différentes variantes sont sélectionnées à la compilation en fonction du système cible. L'ensemble du code spécifique est volontairement conservé aussi minimal que possible. Toujours dans l'exemple de l'initialisation du PRNG, le code commun fait appel à une fonction nommée `ktb_prng_entropy_source()` dont le seul rôle est de remplir un tampon mémoire avec des octets aléatoires provenant d'une source fiable. La partie spécifique à un système d'exploitation est ainsi isolée autant que possible du code commun et est fournie par un unique fichier regroupant ces spécificités.

3.2.5.4 Différents langages

Si le cœur de la plate-forme de sécurité Arcana est une bibliothèque écrite en langage C, les différents environnements de développement peuvent utiliser d'autres langages de programmation. La plupart des langages permettent de faire appel à des bibliothèques natives via des interfaces spécifiques exposant les fonctions des bibliothèques dans un format approprié. Le travail pour rendre la bibliothèque accessible via d'autre langages s'est appuyé sur deux éléments : la restriction de l'API publique à des fonctions et des types de données basiques pouvant aisément se traduire dans des types supportés par de nombreux langages et le développement d'outils permettant la génération automatique d'interfaces pour les différents langages ciblés.

Les restrictions sur l'interface ont pour but d'éviter des situations où la création d'interfaces vers d'autres langages serait compliquée par des conversions entre différentes représentations. En effet des étapes de conversions actives peuvent présenter un risque de sécurité. Si le processus de conversion effectue des copies non contrôlées en mémoire il est possible qu'une fuite d'informations sensibles se produise lors de cette conversion. Afin d'éviter ce problème seul des types basiques sont utilisés dans l'API publique. Cela inclut les types de nombres entiers, les tableaux d'octets, les chaînes de caractères et des pointeurs. En ce qui concerne les trois premiers types, les conversions sont explicites. Pour les pointeurs de données, il s'agit toujours de types opaques. La seule contrainte sur ces pointeurs est qu'ils soient conservés entre les appels au code natif. Le code de l'application appelante, quel que soit le langage utilisé, n'a jamais à manipuler les données référencées par les pointeurs.

Une fois l'API publique simplifié pour permettre une bonne compatibilité avec d'autres langages, il est nécessaire de fournir des interfaces spécifiques à chaque

langage ciblé permettant de faire appel à la bibliothèque native. Plutôt que de concevoir ces interfaces manuellement, un processus de génération automatique a été mis en place. Ce processus automatique permet de produire à partir des fichiers d'en-tête C de la bibliothèque Cryptobox tous les fichiers nécessaires à l'utilisation depuis d'autres langages. Pour l'instant, seule une interface Java est disponible. Cette interface utilise la bibliothèque Java JNA [31][18], qui permet d'établir un lien entre une application Java et une bibliothèque native en fournissant simplement les profils de fonctions utilisées. La bibliothèque JNA se charge des opérations de conversions entre les deux langages, ainsi que du chargement de la bibliothèque native. Afin de pouvoir utiliser ce système d'importation, un logiciel spécialisé a été conçu pour traduire les profils de fonctions C en profils de fonctions Java compatible avec JNA. Cette approche permet une utilisation directe de l'API publique de la bibliothèque Cryptobox depuis des applications développées en Java. Lors des éventuelles mises à jour de l'API publique, il suffit de régénérer automatiquement l'interface Java ainsi créée.

Il y a cependant deux problèmes à l'utilisation de JNA dans un contexte d'application sécurisée. Tout d'abord, les performances obtenues avec JNA ne sont pas optimales. Plus il y a d'informations à transférer entre la partie Java et la partie native, plus les performances se dégradent. Or, dans le cadre d'outils cryptographiques comme des fonctions de chiffrement, il faut parfois traiter des quantités importantes de données. Cet usage n'est donc pas compatible avec l'utilisation de JNA. L'autre problème est lié à la sécurité des données sensibles. Afin de fonctionner de façon transparente, JNA dispose de mécanismes permettant de convertir les types natifs tels que les tampons mémoire vers des types Java plus flexibles. Ces conversions sont susceptibles de laisser des données sensibles en mémoire. Ces fuites de données, combinées à des vulnérabilités systèmes permettant de récupérer le contenu de la mémoire d'autres processus ou de processus terminés, représentent un risque réel.

La solution retenue pour répondre à ces problèmes est l'utilisation d'une interface JNI [32] au lieu de JNA. Contrairement à JNA, JNI est un composant standard du langage Java. JNI permet aussi d'appeler des fonctions natives de façon directe. Les performances obtenues avec JNI sont supérieures à celles obtenues avec JNA. Cela s'explique par l'absence de code automatique "facilitant" le passage de données entre le langage Java et la bibliothèque native appelée. Ces deux points semblent répondre parfaitement aux problèmes soulevés par JNA. Malheureusement, certains types de données ont toujours besoin d'une étape de conversion pour pouvoir être manipulés depuis le langage Java ; principalement les tampons mémoire. La différence avec JNA est qu'ici, les appels de fonctions natives passent

31. Java Native Access, bibliothèque Java permettant de simplifier les appels au code natif
32. Java Native Interface, composant du langage Java permettant d'appeler du code natif

des types spécifiques du Java au code natif au lieu d'effectuer des conversions automatiques n'exposant au code natif que des type "C". Pour réaliser une interface JNI il est donc nécessaire de fournir à la fois les profils de fonctions Java et un composant natif servant de raccord entre les appels JNI et la bibliothèque Cryptobox. C'est dans ce composant natif que les conversions rendant les données accessibles au langage Java ont lieu. L'avantage de cette solution est que nous avons une maîtrise totale de la conversion, qui peut alors avoir lieu de façon sécurisée. Dans le cas spécifique des tampons mémoire, nous utilisons directement des objets Java capables de référencer les tableaux natifs d'octets. Nous évitons ainsi les copies et le risque de fuite de données.

Tout comme pour JNA, un outil de traduction automatique a été réalisé pour permettre la traduction des en-têtes C à la fois vers les profils de fonctions Java adaptés et vers le composant agissant comme interface entre JNI et la bibliothèque Arcana-Cryptobox.

3.2.6 Courbes elliptiques

La plate-forme de sécurité Arcana contient une base de courbes elliptiques adaptées à la cryptographie appelée Arcana-ECDB. Cette base de courbes a été obtenue selon un procédé documenté permettant de garantir la transparence des choix et la fiabilité des courbes[33]. Elle contient des courbes de deux formes différentes : des courbes d'Edwards et des courbes de Weierstrass. Ces courbes permettent de disposer d'une alternative au nombre réduit de courbes proposées par le NIST tout en assurant que les propriétés de sécurité fournies sont corrects. De plus, le processus de sélection étant aléatoire le risque de manipulation ou de choix volontaire d'une courbe disposant de propriétés cachées est écarté.

Afin d'utiliser cette base de courbes, la bibliothèque Cryptobox intègre le code nécessaire à la manipulation de points sur ces courbes. Cette implémentation est capable de gérer les deux formes (Edwards et Weierstrass). Elle est réalisée avec la bibliothèque GMP évoquée précédemment. Il s'agit d'une implémentation purement logicielle, donc complètement portable. L'ensemble des opérations de cryptographie asymétrique de la bibliothèque Arcana s'appuient sur les opérations sur courbes elliptiques. Cela inclut les signatures numériques, l'encapsulation de clé cryptographique et les protocoles d'échange de clés de type ECDH.

3.3 Compatibilité et standards

Afin d'être utilisable dans des projets industriels, la plate-forme de sécurité Arcana doit répondre aux standards du domaine. Les implémentations de l'ensemble des algorithmes et protocoles sont faites conformément aux différents standards

qui les régissent. Par ailleurs les vecteurs de test standards sont utilisés dans les jeux de tests quand ils sont disponibles. Il y a toutefois certains éléments qui présentent une compatibilité potentielle avec certains standards, mais nécessitent des adaptations.

L'échange de clés ECDH disponible dans la bibliothèque Cryptobox est basé sur Diffie-Hellman (le standard PKCS#3). Il ne présente toutefois pas de propriété d'interopérabilité, car la structure des échanges n'est pas compatible. Les formats standards de stockage et d'échange de données de la famille PKCS [33] ne sont pas tous supportés directement. En particulier, PKCS#7 qui spécifie le format de messages signés et chiffrés, PKCS#8 qui spécifie le format de stockage des paires de clés et PKCS#12 qui spécifie le format de stockage des clés privées. Cependant, les outils cryptographiques essentiels pour leur support sont présents dans la bibliothèque Cryptobox. Une application désirant suivre ces standards peut les implémenter à un plus haut niveau si elle le souhaite. Le standard PKCS#11 spécifie une interface de programmation permettant d'utiliser des jetons cryptographiques. Ces derniers sont le plus souvent des composants matériels permettant de réaliser des opérations comme des chiffrements et des signatures. Il n'y a pas de raison de supporter ce standard au niveau de la bibliothèque Cryptobox, car le plus souvent les implémentations PKCS#11 sont accessibles via des bibliothèques dédiées spécifiques. En revanche, les données produites (chiffrements, signatures numériques,. . .) peuvent être traitées et manipulées par la bibliothèque Cryptobox.

Un autre élément standard critique est le support des certificats X.509. Ces derniers permettent d'associer une clé publique à des informations d'identité et d'authenticité, afin de pouvoir prouver son identité. Il n'y a pas dans la bibliothèque Cryptobox de fonctions de manipulation de ces certificats. Cependant, les éléments cryptographiques qui les composent peuvent être obtenus à partir des fonctions cryptographiques de la bibliothèque. Seul reste à la charge de l'application la mise en forme à l'aide d'un traitement ASN1. La gestion complète des certificats est restée hors périmètre pour deux raisons. Tout d'abord, beaucoup des propriétés présentes dans un certificat X.509 n'ont pas de restrictions et peuvent avoir un grand nombre de représentations, sans aucun lien avec l'aspect cryptographique du certificat. En particulier, les informations d'identités, de signataire et de période de validité sont indépendantes des opérations de chiffrement et de signature. Ensuite, le standard actuel présente des lacunes qui compliquent son intégration, notamment au niveau des courbes elliptiques. En effet, s'il est possible de spécifier dans un certificat X.509 des courbes standardisés comme celles fournies par le NIST aussi bien que des courbes personnalisées, cette spécification reste limitée aux courbes de Weierstrass. Le standard actuel ne permet donc pas

33. Public-key cryptography standard, ensemble de standard décrivant des éléments communs pour l'application d'outils cryptographiques

d'utiliser les courbes d'Edwards disponibles dans la plate-forme de sécurité Arcana.

3.4 Travaux futurs

Comme nous avons pu le voir, il y a beaucoup de contraintes à prendre en compte lors du développement d'une plate-forme de sécurité portable. Toute modification doit pouvoir passer une série de tests complets et tout ajout doit être fait avec son jeu de test, respecter les conventions de codage et répondre aux contraintes de portabilité et de performances. De plus, il est essentiel qu'un tel outil continue d'évoluer ; de nouveaux algorithmes sont régulièrement publiés et les algorithmes existants peuvent faire l'objet de révisions pour plusieurs raisons, allant de la vulnérabilité au remplacement total. C'est pour cela qu'il reste plusieurs améliorations à réaliser sur le projet Arcana. Nous présentons ci-dessous quelques-unes de ces perspectives d'amélioration.

Améliorations de l'existant : La bibliothèque Cryptobox dans son état actuel permet d'effectuer toutes les opérations requises en matière de cryptographie et de fonctions annexes. Cependant, des améliorations sur l'accessibilité et la transparence de ces fonctions sont nécessaires. Deux éléments ont été envisagés : la création d'interfaces permettant de manipuler des flux de données et la création d'une API simplifiée. Une interface permettant de manipuler des flux permettrait la construction rapide de protocoles en composant des enchaînements d'algorithmes cryptographiques et des transformations intermédiaires. Un tel schéma pourrait ainsi être enregistré pour être réutilisé comme une "boite noire" disposant de plusieurs entrées et sorties. Un exemple serait l'association rapide d'un algorithme de chiffrement, d'une opération de signature et de leur composition dans un format standard. Une fois imbriqué les uns dans les autres, ces trois éléments seraient alors vus comme un seul algorithme dont l'entrée est constituée le message original, de la clé privée du signataire et de la clé publique du destinataire. Cet algorithme produirait un fichier chiffré, signé et déchiffrable uniquement par le destinataire. L'autre approche de simplification est la création d'une API de plus haut-niveau permettant de masquer les aspects "mécaniques" de l'application de fonctions cryptographiques. Une telle API permettrait à un utilisateur de ne pas se préoccuper des choix d'algorithmes, des paramètres aléatoires et du format des données. Seules les données utilisateur seraient nécessaires et les résultats seraient encapsulés dans des formats intégrant les informations périphériques nécessaires générées automatiquement. Ces deux évolutions possibles permettraient de construire des fonctions plus opaques au niveau de l'utilisateur, mais garantissant que les algorithmes cryptographiques sont utilisés correctement.

Nouveaux algorithmes : De nouveaux algorithmes ont été publiés depuis le lancement du projet Arcana. En particulier la fonction de hachage SHA3 a été choisie. Cette dernière a fait l'objet d'une spécification formelle en août 2015[45]. Il serait intéressant de l'intégrer dès maintenant dans le noyau cryptographique du projet Arcana, car même si elle n'est pas pour l'instant utilisée, les prochains protocoles et algorithmes s'appuieront probablement dessus. D'autres outils cryptographiques apparaissent par ailleurs, notamment des modes d'utilisation pour les chiffrements symétriques ou des fonctions d'authentification nouvelles.

Versions pour usages spécifiques : Lors du développement des optimisations, une attention particulière a été fournie pour que ces changements disposent d'une version alternative ayant une empreinte mémoire réduite, pour permettre l'utilisation de la bibliothèque dans des environnements fortement contraints. Cependant, certaines implémentations de base des algorithmes peuvent avoir des variantes moins gourmandes en certaines ressources, mémoire ou temps de calcul. L'intégration de ces variantes et d'un moyen permettant de créer au besoin une version de la bibliothèque Cryptobox ayant une consommation de ressources plus ciblé permettrait d'accroître les possibilités d'utilisation sur des systèmes embarqués. Dans la même optique, il existe certaines applications pour lesquelles l'approche choisie de fournir un noyau cryptographique en langage C agrémenté de différentes interfaces n'est pas utilisable, notamment pour des clients légers n'ayant accès qu'à des technologies de type JavaScript. Pour ces usages, le développement d'une alternative est nécessaire. Il reste encore à déterminer quels algorithmes seraient pertinents à intégrer dans un tel projet et comment ces développements pourraient tirer parti de la base de code existante.

Optimisations d'algorithmes : Certains algorithmes utilisent actuellement leur implémentation de référence. Pour d'autres, une approche générique a été choisie afin d'assurer un fonctionnement correct dans tous les cas d'utilisation possible. Nous pouvons envisager des améliorations de certains de ces algorithmes. En effet, le protocole d'échange de clés basé sur Diffie-Hellman intégré à la bibliothèque Cryptobox supporte un nombre d'intervenants quelconque. Pour que l'échange puisse se dérouler, tous les intervenants sont placés dans un anneau et les communications se font de façon circulaire, jusqu'à ce que tous les participants disposent de toutes les informations requises. Un tel échange peut être optimisé en utilisant d'autres structures qu'un anneau et en utilisant la redondance obtenue lors des échanges intermédiaires. Ce type d'optimisations est plus complexe à mettre en œuvre et peut poser des problèmes dans certains cas d'utilisations. Mais pour les cas acceptables, présenterait un gain de performances. Au lieu d'avoir une complexité de l'ordre de $O(n)$, n étant le nombre de participants, nous pourrions

obtenir dans le meilleur des cas une complexité de l'ordre de $O(log_2(n))$ en baissant aussi bien le nombre d'échanges à réaliser que le nombre de calculs nécessaires.

Meilleur support matériel : Il y a déjà certaines optimisations tirant profit de matériel spécifique. Il reste cependant beaucoup d'angles possibles pour tirer le meilleur parti du matériel, qu'il s'agisse d'améliorer les performances, de réduire le coût en ressources, ou d'obtenir une meilleure sécurité. Tout d'abord, en plus des instructions "optionnelles" décrites précédemment permettant d'optimiser certains algorithmes, il existe des composants matériels dédiés à l'accélération de primitives cryptographiques. Ensuite, certains composants sont dédiés à la gestion des clés cryptographiques et à leur utilisation. Ce type de composants permet d'éviter que la clé de chiffrement ne transite par la mémoire vive du système, éliminant tout risque de divulgation. Ce genre de matériel peut présenter des avantages, mais doit être soigneusement évalué et se présente trop souvent comme une "boite noire" rendant une telle évaluation difficile. Il existe d'autres possibilités d'optimisations par l'utilisation de matériels ayant une puissance de calcul importante. En particulier les cartes graphiques des systèmes modernes disposent d'un nombre important d'unités de calculs et pourraient être utilisées pour accélérer les opérations cryptographiques. Enfin, une approche intéressante est l'utilisation de composants FPGA. Ces derniers donnent la possibilité de créer à la demande une implémentation matérielle d'un algorithme donné. Le support de ce type de matériel pourrait considérablement améliorer les performances de tous types d'algorithmes comparé à une exécution sur processeur classique. Pour aller plus loin, ce genre de matériel pourrait être programmé non pas avec une implémentation générique d'un algorithme donné, mais une version spécialisée intégrant les éléments secrets comme les clés de chiffrement. Une telle approche (pouvant rappeler la cryptographie en boite blanche) permettrait non seulement d'obtenir de meilleures performances, mais aussi de masquer les informations sensibles.

4 INTÉGRATION DE SÉCURITÉ DANS UN SGED

Les SGED [1] sont des applications permettant la création, la manipulation, le partage et la conservation de documents numériques. La notion de document numérique est ici très large et représente un groupement cohérent de données. Un document peut ainsi être constitué d'un ou plusieurs fichiers de formes variées. La gestion, le stockage et le partage de ces documents par le système peut se faire sur plusieurs périphériques et entre plusieurs utilisateurs.

Les travaux de ce chapitre sont basés sur l'étude du fonctionnement du logiciel Poseidon, produit de la société SESIN. Cette application intègre notamment un SGED servant de support à de nombreuses applications. C'est à partir des fonctionnalités et des cas d'utilisations de ce dernier que nous avons choisi d'étudier la sécurisation de ce type d'applications.

Un SGED est un bon exemple d'application informatique distante, car il regroupe plusieurs problématiques dans les domaines de la communication, du contrôle d'accès et du stockage. Les SGED présentent tout d'abord des problématiques réseaux avec un fonctionnement distribué suivant plusieurs modèles possibles comme le modèle client/serveur et le Cloud. Ensuite, comme ces systèmes sont conçus pour manipuler des documents de toute nature et pouvant présenter un caractère sensible, la sécurité doit être omniprésente. Cela signifie des problématiques d'authentification forte des utilisateurs, de protection des communications et de chiffrement pour le stockage des informations. Enfin, les SGED ont un niveau d'interaction important avec les utilisateurs. Ce dernier point est essentiel dans la conception de systèmes sécurisés. Au-delà de la fiabilité d'un système, son accessibilité est un facteur important. Une mauvaise accessibilité a pour effet de détourner les utilisateurs du système sécurisé et rend inutile toutes les solutions de sécurité mises en œuvre.

Les SGED présentent donc un grand intérêt pour l'application de méthodes de sécurisation des outils numériques.

1. Système de Gestion Électronique de documents, permettant la création, manipulation, et distribution de documents de nature variée

Dans ce chapitre nous étudions étape par étape l'intégration d'une sécurité complète dans un modèle générique de SGED. Ce modèle présente les caractéristiques et fonctionnalités courantes d'un SGED ayant une approche de sécurité initialement basée sur le contrôle d'accès. Les points sensibles à sécuriser sont détaillés avec les solutions théoriques à appliquer et leur intégration dans notre modèle de SGED. Un découpage en étapes de la mise en place de ces solutions est proposé, décrivant un processus de migration partant d'une application initialement non sécurisée et menant à une application entièrement sécurisée.

4.1 Modèle du SGED

Nous considérons un modèle générique de SGED doté d'un mécanisme de sécurité basé sur un contrôle d'accès. Ce modèle suit un principe de fonctionnement client/serveur classique, dans lequel toutes les informations transitent par le serveur et les postes clients peuvent être de nature variés. Les utilisateurs ne sont pas liés à un poste client spécifique, mais peuvent au contraire être mobiles.

L'architecture et les fonctionnalités de ce modèle sont basées sur la structure et le fonctionnement de l'outil industriel Poseidon proposé par la société SESIN. Poseidon est décliné en de nombreuses applications s'articulant autour d'un SGED. Il est utilisé comme support pour la création d'applications personnalisées en fonction de différents besoins. Le modèle décrit ici concerne la partie SGED de Poseidon et les solutions proposées prennent en considération les contraintes d'exploitations réelles qui lui sont associées.

4.1.1 Architecture

L'architecture du modèle est présenté dans la figure 4.1. Le serveur centralise l'ensemble des informations du système : il a accès à toutes les données. L'index des méta-données contient toutes les informations associées aux documents conservés. Il permet de retrouver un document particulier en fonction de différents critères de recherche comme le nom, la date, des mots-clés, etc. Le stockage des documents conserve l'ensemble des documents connus du SGED, corps et méta-données. La base d'utilisateurs regroupe la liste de tous les utilisateurs du système avec leurs droits d'accès. Cette base contient également les informations d'authentification (un mot de passe dans cet exemple).

Nous considérons un unique serveur "logique". Différents scénarios sont possibles : serveur unique ou multiples, synchronisations temps réel ou a posteriori, réplication pour sauvegarde, etc.

Du point de vue de la sécurité de fonctionnement de l'application, ces différents scénarios peuvent être regroupés sous un seul serveur "logique" même si physique-

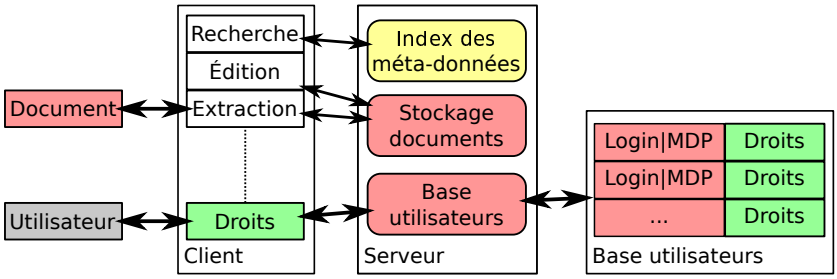

FIGURE 4.1 – Composants du modèle de SGED générique

Les éléments rouges sont les données sensibles conservées par le système. Les autres éléments ne sont pas confidentiels.

ment ce n'est pas le cas. En effet, dans tous les cas l'interaction entre le poste client/l'utilisateur et le serveur se fait de façon point à point. Les éléments de sécurités présentés sont conçus pour minimiser l'importance de l'infrastructure et ne sont donc pas impacté par la différence entre un serveur central et une architecture multi-serveurs.

L'application cliente est une interface permettant à un utilisateur d'accéder au système. Elle permet la transmission des informations d'authentification au serveur, qui associera alors les droits spécifiques de l'utilisateur à la session ainsi créée. En fonction de ces droits, l'utilisateur pourra effectuer différentes actions sur le système : recherche de document, modification des méta-données, récupération du corps d'un document, etc.

Afin de pouvoir manipuler le contenu d'un document, ce dernier est extrait du SGED pour pouvoir être utilisé dans un logiciel d'édition adapté, puis réintégré après modification. Le document ainsi mis à jour peut remplacer l'ancienne version ou être ajouté comme une nouvelle version via un changement dans les méta-données associées.

Le client peut prendre différentes formes en fonction de contraintes d'utilisations variées. Les deux modèles considérés sont l'utilisation d'un client applicatif lourd, principalement dédié à l'utilisation sur poste de travail classique et l'utilisation d'un client web léger, permettant un accès depuis plus de systèmes différents y compris en mobilité, mais limitant les possibilités techniques. Un client web léger dispose d'un ensemble de fonctionnalités plus réduit que les solutions client lourd. Cependant, cette différence tend à s'estomper avec la possibilité de plus en plus courante d'utiliser des portions d'applications "lourdes" intégrées aux clients légers de façon transparente. Les solutions techniques permettant de sécuriser le système que nous proposons s'appuient sur des possibilités de manipulation avancées des données tout en tenant compte des limites actuellement imposées par les applications web. Une approche avancée pour permettre ces manipulations est décrite dans la section 4.3.3.1.

La partie serveur n'est pas exposée à l'utilisateur et offre donc beaucoup de libertés sur les possibilités de développement logiciel. Nous partons ici d'un serveur applicatif écrit en langage C permettant l'intégration avec des outils provenant d'environnements variés. L'application cliente quant à elle pose plus de limitations. Qu'il s'agisse du client lourd ou de fonctionnalités avancées intégrées au client léger, les développements proposés seront également basés sur des environnements C pour des raisons de portabilité. Les communications entre le serveur et le client n'imposent initialement pas de contraintes particulières autres que celles d'un réseau TCP/IP [2] classique ; c'est-à-dire la délivrance ordonnée et unique des messages.

4.1.2 Fonctionnalités

Dans la mesure où il s'agit d'un SGED modèle, nous ne décrivons ici que les fonctionnalités pertinentes pour la suite du chapitre. Le modèle de sécurité déployé restant compatible avec des fonctions plus avancées non détaillées. En considérant l'ensemble du SGED comme une unique application, les fonctionnalités du point de vue de l'utilisateur sont les suivantes : authentification, création et édition de documents, récupération/mise à jour de contenus, recherche indexée et messagerie. Ces fonctionnalités ont pour but de rendre accessible et utilisable des "documents".

Les documents sont les éléments principaux manipulés par le système. Ils sont constitués de deux parties : les méta-données et le "corps" du document. Les méta-données contiennent toutes les informations permettant l'indexation du document. Il peut s'agir de différentes propriétés comme des dates, des indicateurs de catégories, des renvois à d'autres documents, etc. Ces éléments sont utilisés par le SGED pour permettre notamment la recherche et le traitement automatisé des données. Le corps du document peut prendre différentes formes ; si la plus courante est un fichier, il peut également être constitué de plusieurs fichiers, ou être simplement un ensemble de propriétés (paire nom/valeur) similaires aux méta-données. La distinction essentielle est que le corps du document peut contenir des données plus sensibles que les méta-données en termes de confidentialité et de contrôle d'accès.

L'authentification est l'étape initiale qui conditionne l'accès aux autres fonctionnalités. Elle fonctionne selon un schéma basique : l'utilisateur transmet ses informations d'authentification, typiquement un mot de passe, au serveur qui les compare alors avec les informations enregistrées. En cas de succès, la session de l'utilisateur se voit associé les droits correspondants stockés sur le serveur. Les utilisateurs peuvent également être membres de groupes spécifiques ; ces groupes pouvant eux-mêmes disposer de droits qui sont alors associés à tous les membres. Une fois l'authentification réalisée, les requêtes ultérieures seront validées par le

2. TCP over IP, suite de protocoles utilisant TCP sur un réseau IP

serveur qui décidera de répondre ou non en fonction de ces droits individuels ou de groupe.

Du point de vue du SGED, la création et l'édition de document concernent la manipulation des méta-données et leur association avec le corps du document. Les méta-données sont constitués d'éléments de nature variée et libre, mais contiennent aussi les informations d'accès au document afin que le serveur puisse en restreindre la manipulation. Ainsi, seuls les utilisateurs autorisés pourront accéder à un document donné. De même, la création initiale d'un document est également limitée par les droits utilisateurs.

La récupération et la mise à jour de contenu concerne le corps des documents. Elle permet de télécharger les éléments constituant le document (le plus souvent un fichier) afin de pouvoir le manipuler en dehors du SGED. Le contenu ainsi mis à jour peut alors, selon les droits de l'utilisateur, être intégré dans le SGED afin de mettre à jour le document archivé.

La recherche indexée de contenu permet de créer des listes de documents répondant à des critères spécifiques. Cette recherche peut se baser sur les méta-données des documents ou leur contenu. La recherche à partir de méta-données permet de spécifier finement le type de champs à examiner, alors que la recherche sur le contenu est une simple recherche de correspondance. Ces opérations de recherche sont conditionnées par les droits de l'utilisateur, d'une part le droit d'accéder au système de recherche et d'autre part le droit d'accéder aux documents retournés par ces recherches.

La fonction de messagerie permet des communications entre utilisateurs du SGED. Ces communications peuvent prendre une forme arbitraire, mais présentent les caractéristiques d'une messagerie hors-ligne permettant l'envoi de messages à des utilisateurs temporairement indisponibles qui seront rapatriés dès que possible.

Le SGED ainsi décrit peut également être considéré comme un serveur de fichiers augmenté permettant d'effectuer des opérations de recherche, de journalisation et de transformation sur le contenu manipulé.

Les fonctionnalités décrites ici sont toutes impactées par la mise en place d'outils de sécurité. Certaines de ces fonctionnalités sont modifiées ou mises à profit pour permettre l'utilisation des outils de sécurité, alors que d'autres comme les fonctions de recherches sont impactées négativement. La section 4.3 détaille ces changements et comment minimiser l'impact négatif sur les fonctions existantes du système.

4.1.3 Faiblesses de sécurité

Pour des raisons de simplicité, notre modèle générique de SGED n'utilise initialement aucun moyen cryptographique, même si en pratique tous systèmes récents utilisent un minimum d'outils sécurisés. Ces outils n'apportent qu'un faible niveau

de sécurité lorsqu'ils sont utilisés individuellement ou incorrectement. Il s'agit essentiellement de solutions de protections de communications (de type TLS [3]) appliquées sans maîtrise des vérifications associées, de l'utilisation de moyens d'authentification basiques sur des canaux sécurisés ne prévenant pas les risques de compromission côté serveur et de méthodes génériques de chiffrement des données au repos ne protégeant pas effectivement contre la plupart des attaques sur le serveur. Nous allons voir tout d'abord quels éléments présentent des points faibles dans le fonctionnement d'un SGED, puis nous décrirons les limites des solutions de protections individuelles à chaque point de faiblesse.

4.1.3.1 Points faibles d'un SGED

Dans le modèle de SGED considéré, la sécurité des documents n'est garantie que par un seul élément : le contrôle d'accès limitant les droits d'un utilisateur authentifié. Il y a plusieurs points de faiblesse qui peuvent permettre à un attaquant de compromettre la sécurité des données et des utilisateurs du système : le stockage des documents, le processus d'authentification, les communications et l'utilisation du document côté client. Chacun de ces points est expliqué ci-après ; les solutions permettant de répondre aux problèmes soulevés sont détaillées dans la section suivante. La figure 4.2 résume les points faibles du modèle initial de SGED considéré ici.

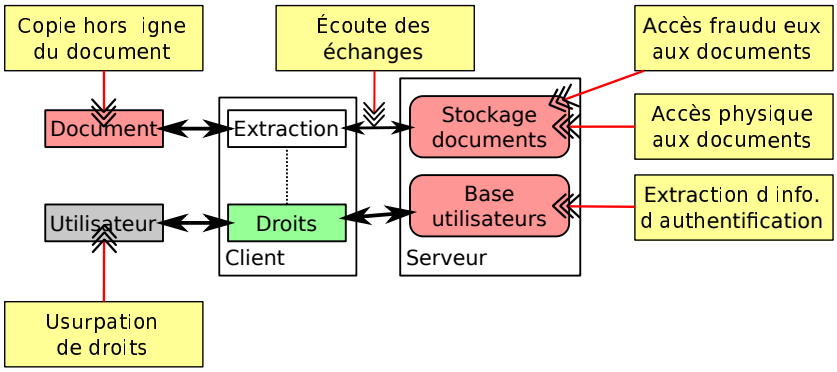

FIGURE 4.2 – Faiblesses du modèle initial de SGED

Les triples flèches indiquent les points d'attaque du système.

Conservation des documents : Les documents sont tous archivés sur le serveur qui en contrôle l'accès. Cependant, lorsqu'on parle de sécurité il est essentiel de considérer qu'un attaquant cherchera à contourner les systèmes mis en place.

3. Transport Layer Security, protocole de communication sécurisée remplaçant SSL

Dans le cas du stockage des documents, ou de toutes autres données sensibles, leur existence sous forme "lisible" sur un point central comme un serveur de données est un problème. En effet, il existe plusieurs moyens d'accès à des données stockées allant de la simple compromission physique du support de stockage à l'accès frauduleux au serveur permettant de récupérer les données sans passer par le processus habituel. Il est également possible d'exploiter directement d'autres failles du serveur afin de donner des droits à un attaquant, lui permettant de récupérer des données sensibles. De manière générale, le fait que le serveur ait accès aux données en clair les rend vulnérable, quelle que soit la méthode utilisée pour en assurer la protection. Il est donc nécessaire de mettre en place une solution pour laquelle le serveur n'a qu'un accès restreint aux données qui lui sont nécessaires pour fonctionner, c'est-à-dire les méta-données.

Processus d'authentification : L'authentification doit répondre à des critères de sécurité strictes pour garantir qu'elle ne sera pas manipulée. Il existe plusieurs méthodes d'authentifications "peu sûres" n'impliquant ni protection du processus d'authentification, ni garantie contre le rejeu ou d'autres formes d'attaque. Dans le cas de notre modèle de SGED, il s'agit d'une authentification par mot de passe simple. Cela signifie que le serveur a connaissance des mots de passe des utilisateurs et donc que ce secret n'est pas connu de l'utilisateur seul. Dans ces conditions, il ne peut pas y avoir de garantie qu'une authentification ait réellement été réalisée par un utilisateur donné. Toutes les actions qui pourraient en découler, ainsi que leur journalisation, ne sont pas fiables, car la compromission de la base utilisateurs peut donner les moyens à un attaquant d'usurper l'identité des utilisateurs. Ce problème est résolu par l'utilisation de moyens d'authentification cryptographiques et l'utilisation d'authentification forte.

Connexions sécurisées : Si l'utilisation de connexions non sécurisées pour établir les communications entre le client et le serveur présente un risque évident, l'utilisation de moyens de sécurisation dépendant de paramètres externes n'est pas non plus une solution appropriée. Afin d'assurer la protection de toute la chaîne de fonctionnement entre l'utilisateur et les documents manipulés, il est nécessaire non seulement d'établir des communications sécurisées, mais aussi d'assurer la maîtrise des éléments le permettant. En particulier, l'utilisation de protocoles de type TLS sans s'assurer de la fiabilité des certificats utilisés peut permettre une compromission totale des communications (voir la section 1.1.1). Une solution consiste à maîtriser l'établissement de connexions sécurisées entre le client et le serveur afin de ne pas compromettre la confiance accordée au système.

Diffusion de documents non protégés : L'utilisation qui est faite des documents par le poste client présente un risque de fuite de données. En supposant une gestion entièrement sécurisée de tout le chemin entre le poste client et le serveur, la manipulation du document final pose un problème de confidentialité. En effet ce dernier doit être rendu accessible aux différents logiciels permettant de le manipuler, or l'intégration de ces logiciels dans un processus sécurisé n'est pas toujours possible, en particulier à cause de leur nombre et de contraintes techniques individuelles. Il est donc nécessaire d'extraire le document du système en le sortant de la partie "sécurisée" afin de permettre sa manipulation avant son retour dans le système. Cette période d'existence non sécurisée présente un véritable trou de sécurité ; de nombreuses causes involontaires (gestion de la mémoire par le système d'exploitation, réplication des supports de stockage,...) et volontaires (copie des données lors de leur manipulation,...) peuvent mener à la fuite d'informations confidentielles. La protection des données lors de leur utilisation est un problème complexe ; la section 4.4 présente une réflexion sur ce point.

4.1.3.2 Limites des solutions de sécurité classiques

Les outils de protection classiques couramment utilisés pour répondre à des problèmes spécifiques montrent des limites importantes dans le cas de l'utilisation d'un SGED. L'interaction entre les différents besoins de sécurité et la versatilité du système peut provoquer des effets de bord négatifs.

Protection des communications : Les protocoles permettant d'établir des connexions sécurisées tels que TLS imposent de respecter certaines contraintes sans quoi leur efficacité est limitée. En particulier, afin d'établir une communication sûre entre deux systèmes il est nécessaire de pouvoir procéder à l'authentification d'au moins l'un d'entre eux. Dans l'exemple de TLS cela signifie disposer de certificats dont on peut prouver la validité. Cependant, le système de certificats s'appuie quasi-systématiquement sur un mécanisme d'autorités de certification tierces ainsi que sur un magasin de certificats géré par le système d'exploitation de l'utilisateur. Ces deux points peuvent être à l'origine de différentes attaques réduisant la confiance que l'on peut accorder à cette approche. À cela s'ajoute des vulnérabilités propres aux protocoles eux-mêmes. La conséquence est que l'utilisation de protocoles tels que TLS basé sur ces éléments tiers ne permet pas d'avoir une confiance totale dans la sécurité des communications.

La faible confiance dans l'établissement de communications sécurisé peut avoir un effet en cascade sur les autres éléments du SGED. Certains processus d'authentification distants peuvent s'appuyer sur la sécurité du support de communication utilisé afin de protéger leur fonctionnement. De même, l'échange de documents entre un client et un serveur peut n'avoir comme seule protection que celle fournie

par le protocole de communication sécurisé. Un problème de confiance dans ce dernier peut donc réduire la confiance accordée à l'ensemble de l'application.

Méthodes d'authentification : Les moyens d'authentification mis en œuvre par un SGED doivent pouvoir prouver de façon fiable que l'utilisateur n'est pas un usurpateur. Des approches classiques basées sur la communication d'un mot de passe présentent de nombreux inconvénients. Tout d'abord, ces communications s'appuient souvent sur un canal de communication protégé et ne fournissent pas leur propre sécurité. Ensuite, pour fonctionner elles nécessitent une connaissance des informations privées par le serveur, ce qui signifie que l'utilisateur n'est pas la seule personne à en disposer. Même en considérant l'utilisation d'un moyen de communication sécurisée de confiance, ces approches présentent des vulnérabilités en termes de sécurité, car le vol d'informations sur le serveur permet d'usurper l'identité d'un utilisateur. Dans le cas d'un moyen de communication peu sûr, un attaquant peut récupérer les informations d'authentification comme le mot de passe lors d'une tentative d'authentification et les utiliser directement. Quelle que soit la méthode utilisée, dès lors qu'un usurpateur parvient à réussir une authentification auprès du serveur, il dispose des droits de l'utilisateur en l'absence d'autres mécanismes en place pour valider ces droits.

Données au repos : Les informations conservées sur un serveur comme les informations conservées sur un poste client présentent le même problème : elles doivent être accessibles au SGED pour qu'il puisse fonctionner, mais doivent être inaccessibles à toute entité ne disposant pas de droits d'accès. Une solution courante est l'utilisation de supports de stockages chiffrés. Une fois le système arrêté, les supports de stockages sont inaccessibles en l'absence de connaissance des clés permettant de les déchiffrer. Une autre approche est de réaliser un chiffrement au niveau des différentes applications. Dans ce cas, le SGED peut lui-même chiffrer ses données avant de les envoyer au support de stockage afin d'en protéger le contenu. Les deux approches partagent la même vulnérabilité : le système sur lequel les opérations de chiffrement se produisent est le même que celui sur lequel les données sont manipulées. Dans le cas du serveur, cela signifie que pendant le fonctionnement les informations permettant de déchiffrer les données sont disponibles, sans quoi le système ne pourrait pas fonctionner. En termes de sécurité, tant que le système fonctionne la différence entre un stockage chiffré et un stockage non chiffré est mineure ; un attaquant pouvant accéder au système aura à disposition tous les éléments nécessaires pour passer outre cette protection. Notons toutefois que le chiffrement des données reste efficace en cas de vol de données "à froid", notamment la récupération physique des supports de stockage, à condition que les clés de déchiffrement ne soient pas elles-mêmes conservées sur les systèmes

concernés.

4.2 Solutions de sécurité

S'il existe des solutions d'appoint pour chacun des problèmes présentés, elles ne prennent pas en compte des éléments inhérents à la mise en place d'un système tel qu'un SGED. Aux contraintes de sécurité s'ajoutent des contraintes d'utilisation, d'ergonomie et de flexibilité qui ne sont pas entièrement couvertes par les solutions individuelles de sécurité. Un système de sécurité adapté à l'utilisation d'un SGED doit donc tenir compte de plusieurs points.

Tout d'abord, l'ergonomie et l'accessibilité imposent des outils de sécurité aussi peu intrusifs que possible. En particulier, l'approche du SSO [4] doit s'appliquer à l'utilisation du système et ne pas nécessiter que l'utilisateur s'authentifie auprès de différents composants. Ensuite, les fonctions de sécurité doivent être intégrées aux outils de gestion des documents ; l'accès à un document et la manipulation de ce dernier doivent se faire de façon transparente, avec le minimum possible d'altérations de l'interface à laquelle est habitué l'utilisateur. Enfin, la mobilité des utilisateurs n'est pas toujours compatible avec des systèmes d'authentification forte ou de cryptographie côté client. Cette limite est particulièrement importante dans notre contexte, car l'objectif à terme est de déplacer la protection des contenus sensibles du serveur vers le client. Ces points font que l'application directe de solutions courantes à chaque point de faiblesse n'apporte pas une réponse adaptée au besoin de sécurité du SGED.

Afin d'apporter une réponse appropriée, différents éléments de sécurité doivent être développés et il doit exister une collaboration entre eux. Ces éléments utilisés ensemble permettent de sécuriser un SGED en impactant le moins possible son fonctionnement initial. Les outils de protection des communications et d'authentification décrit aux chapitres 1 et 2 font partie de cette réponse. Nous détaillons ici comment leur intégration est faite au sein d'un SGED existant.

La finalité du dispositif est la mise en place de solutions cryptographiques autant que possible côté client afin de limiter l'implication du serveur dans le dispositif de sécurité. L'utilisation correcte d'outils cryptographiques permet d'apporter des garanties quant à la fiabilité du système. Ils permettent également la mise en place de solutions asymétriques évitant le partage d'informations sensibles notamment avec le serveur. Le déplacement des moyens de protection du serveur vers le client a un double rôle. D'une part, en impliquant le moins possible le serveur une compromission de ce dernier entraîne beaucoup moins de risques pour les données des utilisateurs. D'autre part, les vulnérabilités liées à l'infrastructure,

4. Single Sign-On, système permettant de s'identifier une seule fois pour l'utilisation de plusieurs systèmes

qu'il s'agisse des communications ou des aspects de réplication et de sauvegarde côté serveur, n'impactent pas la sécurité des documents protégés avant même leur transmission.

4.2.1 Connexions sécurisées

L'établissement d'une connexion sécurisée entre le serveur applicatif et les clients est un élément clé du SGED. La protection des communications permet de protéger les éléments non sensibles, mais devant néanmoins rester confidentiels vis-à-vis d'entités extérieures au système. Cela inclut les méta-données des documents, les requêtes de recherche, etc.

La communication entre les clients d'une application et le serveur applicatif associé est un exemple typique du cas des communications internes décrit dans la section 1.2. Il est donc possible d'utiliser un protocole dédié comme le protocole SVC [5] à la place de protocoles de sécurité classiques. Dans tous les cas, il est primordial de s'assurer de la validité des éléments permettant d'établir la connexion sécurisée. Dans le cas d'une application client/serveur, cela signifie contrôler l'élément d'identité du serveur, que ce soit sous la forme de certificats numériques ou de clés publiques pré-enregistrées. Cette information doit être sous le contrôle de l'application afin d'éviter les manipulations extérieures.

L'approche courante consistant à appliquer une solution générique de sécurité (généralement sous la forme du protocole TLS) ne répond pas aux exigences posées par l'analyse globale du SGED. Ces solutions génériques n'établissent pas de liens entre la mise en place de connexions sécurisées et les informations de sécurité utilisées, qu'il s'agisse des informations d'authentifications de l'utilisateur, du système distant ou des éléments de protection des données transportées. Il est raisonnable d'envisager ces communications sécurisées comme un simple média sur lequel transiteront des informations sécurisées par ailleurs. Toutefois l'intégration de cet élément de sécurité avec les autres peut permettre d'en renforcer la sécurité en le liant plus fortement à l'authentification de l'utilisateur. De plus, cette intégration a un impact positif sur les éléments d'interface utilisateur en créant une continuité entre l'étape primordiale qu'est l'établissement de connexions sécurisées et l'authentification de l'utilisateur par des moyens cryptographiques. Bien qu'on ne puisse pas qualifier ce dernier point de propriété de sécurité à proprement parler, il a un impact bien réel sur l'utilisation effective d'autres outils de sécurité et doit donc être considéré lors de la mise en place de solutions de sécurité. Ce sont ces points qui sont mis en avant ici en proposant un protocole permettant à la fois d'augmenter la fiabilité du système et d'intégrer les éléments d'authentifications à

5. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

la négociation de connexions sécurisées.

4.2.2 Gestion de clés et partage de droits

Tout système cryptographique s'appuie sur des clés pour fonctionner. Ces clés peuvent prendre plusieurs formes et être présentes en grand nombre. Dans le cas de l'utilisation proposée pour un SGED ces clés représentent les droits d'accès au système des utilisateurs. Idéalement, l'utilisateur ne manipule pas directement ces clés, mais agit sur leur rôle dans le système via différentes interfaces. Un trousseau de clés est créé pour les conserver de façon sécurisée et faciliter leur transfert. La section 2.3 détaille la distribution de ce trousseau de clés.

Contrairement aux systèmes de gestion de droits se basant sur des validations de règles pour déterminer la légitimité d'un accès, le système proposé ici est de lier toute action sur les données du SGED à des clés cryptographiques. L'objectif est de garantir la protection et le contrôle d'accès aux données indépendamment de la confiance accordée à l'infrastructure. En effet, même si le service considéré est "de bonne foi", il n'est pas à l'abri d'une vulnérabilité permettant l'accès aux données. Remplacer la gestion des accès par des protections cryptographiques dites "de bout en bout" transforment cette infrastructure en simple intermédiaire ne pouvant pas permettre un accès illégitime aux données, que ça soit volontairement ou suite à l'exploitation d'une faiblesse. Cette approche nécessite pour fonctionner de pouvoir traduire les différents types de droits d'accès en clés cryptographiques, ainsi que leur gestion et leur transfert. C'est pour cela que l'on utilise un trousseau de clés sécurisé intégré au fonctionnement du SGED.

Le processus d'authentification initiale est utilisé afin d'identifier de façon vérifiée l'utilisateur auprès du système. Une fois cette étape réalisée, le trousseau de clés sécurisé est rapatrié par le client depuis le stockage du SGED. Il est alors déchiffré à partir des informations d'authentification de l'utilisateur. L'interface du client permet de manipuler les éléments du trousseau sous la forme de droits utilisateurs.

La récupération et la mise à jour du trousseau de clés sécurisé conservé sur le serveur tire parti de la fonction de partage de document du SGED. Le trousseau de clés sécurisé est vu par le SGED comme un document auquel l'accès est restreint à l'utilisateur par les moyens de contrôle d'accès classiques. L'application cliente peut donc rapatrier ce "document" depuis le serveur comme s'il s'agissait d'un document normal. Une fois récupéré, le déverrouillage du trousseau de clés sécurisé peut se faire sur le client sans interaction supplémentaire avec le serveur.

La protection par le système de contrôle d'accès non cryptographique a pour but de limiter la diffusion du trousseau de clés sécurisé. La sécurité effective du trousseau de clés est basée sur la connaissance d'une information connue de l'utilisateur seul, issue du protocole d'authentification, comme décrit dans la section

2.3. Une fois déverrouillé, le trousseau de clés présente à l'utilisateur les différents droits qu'il contient.

Chaque droit d'accès à un document est matérialisé sous la forme d'une clé associée à ce document. La possession de cette clé permet le déchiffrement et le chiffrement du document associé.

La modification des éléments non chiffrés, comme les méta-données, nécessite la possession d'une paire de clés privée/publique associée. Cette paire de clés permet de produire des signatures numériques validant ces changements. Le serveur garde de son côté un annuaire de clés publiques afin de pouvoir vérifier ces signatures.

Enfin, le trousseau de clés possède une paire de clés privée/publique associée à l'identité de l'utilisateur. Cette paire de clés permet à l'utilisateur d'échanger des messages authentifiés et de recevoir des messages qu'il sera seul à pouvoir déchiffrer. Comme précédemment, le serveur conserve une liste des clés publiques des utilisateurs afin que l'authentification et l'envoi de messages secrets puisse fonctionner même en l'absence d'un utilisateur donné.

Afin de permettre une gestion à grande échelle des droits, il est également possible de créer des trousseaux de clés qui n'appartiendront pas cette fois à un utilisateur, mais à un groupe, de la même façon que l'on pouvait associer un utilisateur à un groupe dans le système de contrôle d'accès. Dans ce cas, le trousseau de clés de l'utilisateur dispose de la clé secrète permettant de déchiffrer le trousseau de clés du groupe. Ce dernier peut être rapatrié de la même façon, via la fonction de récupération de document du SGED.

L'utilisateur dispose de tous les éléments nécessaires pour diffuser les droits dont il dispose à d'autres utilisateurs. L'envoi de clés associées à un droit particulier vers un autre utilisateur est fait en utilisant un système de chiffrement asymétrique, illustré par la figure 4.3. Cet envoi utilise la clé publique du destinataire, connue de tous via l'annuaire conservé par le serveur. Ce dernier sert d'intermédiaire au transfert et permet l'envoi de droits d'accès à des utilisateurs hors-ligne. Seul le destinataire en possession de sa clé privée personnelle est en mesure de déchiffrer cette transmission. Ce mécanisme d'envoi chiffré à destination d'un utilisateur spécifique est similaire à l'utilisation de la messagerie sécurisée décrite section 4.2.5.

L'ensemble des éléments cryptographiques du système s'appuient sur les clés disponibles dans le trousseau de clés de l'utilisateur. Cette approche permet d'utiliser un unique système de clés indépendant des modalités d'authentification des utilisateurs. En effet, en fonction de ces modalités, il pourrait être possible de les utiliser comme mécanismes cryptographiques. Cependant, l'implication de mécanismes d'authentification variés dans la génération des clés des utilisateurs pose un problème dans un contexte de mobilité : l'accès aux informations sur des systèmes ne permettant pas d'utiliser toutes les modalités d'authentification serait limité.

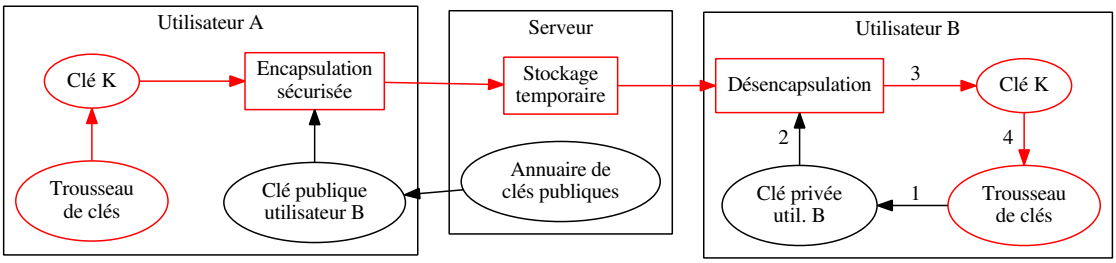

FIGURE 4.3 – Transfert de droits d'accès du système sécurisé

L'utilisateur A donne à l'utilisateur B le droit d'accès au document protégé par la clé K. Le chemin en rouge représente le transfert de la clé.

L'utilisation d'un trousseau de clés sécurisé résout ce problème : les moyens d'authentification variés permettent tous de déverrouiller le trousseau de clés, qui est ensuite accessible quel que soit le système utilisé.

4.2.3 Protection des documents

Les documents sont le premier élément à sécuriser dans un SGED. Leur protection est assurée par des méthodes de chiffrement cryptographiques. Lors de sa création dans le système, un document se voit attribuer une clé secrète unique. Cette dernière est utilisée pour chiffrer et déchiffrer le corps du document. Les méta-données ne sont pas chiffrées ; en revanche elles contiennent les informations permettant d'associer la clé au document.

Les opérations de chiffrement et déchiffrement sont réalisées sur le poste client et la clé du document est conservée de façon sécurisée dans le trousseau de clés de l'utilisateur. Ce fonctionnement est illustré par la figure 4.4. À aucun moment l'infrastructure, qu'il s'agisse du support de communication ou du serveur, n'a accès au corps du document. Seule l'information référençant la clé utilisée est conservée dans les méta-données partagées avec le serveur. La diffusion du droit d'accès au document auprès d'autres utilisateurs se fait de la façon décrite dans la section 4.2.2, en transmettant la clé protégée afin que seul le destinataire puisse la déchiffrer.

L'utilisation de cette approche a un impact sur le fonctionnement du SGED. Du point de vue du client, le processus est quasiment transparent. Il n'y a pas de changement vis-à-vis des possibilités de manipulation du corps du document ; de même le rapatriement et l'envoi de nouvelles versions au serveur mettent automatiquement en œuvre les opérations de chiffrement et déchiffrement. Le changement majeur est la façon dont les droits sont gérés ; seul un utilisateur ayant le droit de manipuler le document peut donner ce droit à un autre. Il n'est donc plus possible d'ajouter des droits à un utilisateur uniquement à partir du serveur.

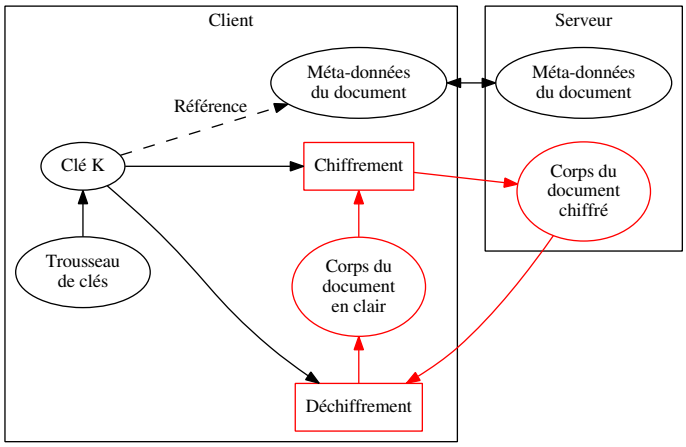

FIGURE 4.4 – Protection des documents

Le document est protégé par la clé K. Le serveur n'effectue aucune opération de chiffrement. Le chemin en rouge représente le cycle de transfert du corps du document.

L'autre fonctionnalité modifiée par cet apport de sécurité concerne la possibilité de manipuler le document par le serveur. En particulier l'analyse du contenu, la recherche directe sur ce dernier ou sa manipulation afin de créer d'autres documents automatiquement n'est plus possible. Enfin, les documents ainsi protégés sont définitivement inaccessibles en l'absence des clés associées. Dans certains cas d'usage, en particulier lorsque des contraintes légales existent, il est souhaitable de pouvoir lever le secret, ce qui est rendu impossible par l'application de procédés cryptographiques. La sous-section 4.2.6 présente des solutions permettant de conserver certains de ces aspects dans un contexte sécurisé.

4.2.4 Protection des méta-données

Les méta-données associées aux documents ne sont pas chiffrées, afin de permettre la conservation des fonctionnalités d'indexation du SGED. Leur modification doit toutefois rester limitée aux seules utilisateurs autorisés. Afin d'assurer que seuls des utilisateurs légitimes puissent produire des modifications sur ces méta-données un mécanisme de signature numérique est utilisé. De la même façon que la clé secrète permettant le chiffrement du document est générée lors de la création du document, une paire de clés associées aux méta-données est produite et la clé publique est enregistrée sur le serveur, associée au document. La clé privée est conservée dans le trousseau de clés de l'utilisateur. Ainsi, les modifications apportées aux méta-données sont associées à une signature numérique attestant de la possession du droit de modification. Le serveur peut vérifier cette signature avant d'enregistrer ces changements.

En termes d'altération des fonctionnalités du SGED, l'impact est nul. Côté client, les opérations sont encore une fois transparentes ; un utilisateur disposant de la clé privée dispose du droit de modification et l'étape de signature numérique est transparente. Pour le serveur, l'étape de vérification de signature est également transparente. Les méta-données n'étant pas chiffrées, leur accès par le serveur n'est pas modifié et n'impacte donc pas les fonctionnalités associées.

4.2.5 Messagerie sécurisée et envoi de droits

L'envoi manuel de messages entre utilisateurs et l'envoi de droits, c'est-à-dire de clés, utilisent le même mécanisme basé sur de la cryptographie asymétrique. Chaque utilisateur dispose d'une paire de clés privée/publique permettant la mise en œuvre d'un tel système. Un message est protégé de deux façons : d'une part, le contenu du message est chiffré avec un chiffrement asymétrique en utilisant la clé publique du destinataire ; d'autre part le message est signé par l'émetteur. La clé publique du destinataire est disponible via un annuaire conservé par le serveur. Cet annuaire permet également au destinataire de vérifier la signature du message reçu. Ce fonctionnement est illustré par la figure 4.5.

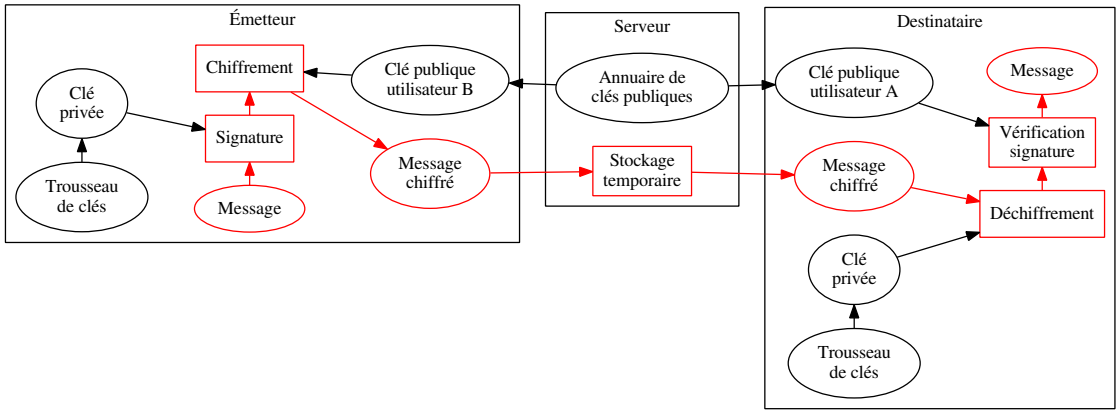

FIGURE 4.5 – Encapsulation et envoi de messages sécurisés

L'utilisateur A envoi un message sécurisé à l'utilisateur B. Le serveur joue le rôle d'intermédiaire pour permettre une délivrance asynchrone. Le message transite par le chemin indiqué en rouge.

L'envoi d'éléments entre utilisateurs est le même qu'il s'agisse de messages personnalisés ou de l'envoi de droits d'accès au système sous forme de clés cryptographiques. Ces dernières font cependant l'objet d'un traitement automatisé afin d'être intégrés de façon transparente au trousseau de clés de l'utilisateur concerné.

La protection ainsi mise en place des fonctions de messagerie n'impacte en rien l'utilisation du SGED. Les opérations de chiffrement, déchiffrement et de signature

numérique sont toutes traitées sans intervention active de l'utilisateur. L'action du serveur reste la même qu'en l'absence de chiffrement et consiste au transfert des messages vers les utilisateurs dès que ceux-ci se connectent. L'envoi de droits d'accès utilise le même mécanisme de protection, mais s'intègre dans différentes parties de l'interface utilisateur dédiées à la gestion des droits. Le comportement du serveur dans cette diffusion de droits n'a pas besoin d'être distinct de la fonction de messagerie. Il peut toutefois être utile de conserver une trace de ces transferts d'autorisations à des fins de journalisation. La section 4.2.6 présente une possibilité d'extension du système pour permettre cela.

4.2.6 Maintien des fonctionnalités

Sur certains points, les solutions de sécurité proposées entraînent une perte de fonctionnalités. La perte de certaines fonctionnalités est inévitable, car liée au besoin initial, comme l'exclusion du serveur de la liste des entités ayant accès aux données. Il existe cependant des modifications périphériques permettant de restaurer certaines fonctionnalités. Nous allons décrire ici des pistes permettant de rétablir des fonctions de traitements automatisés, de recherche avancée, de gestion des droits et de confidentialité révocable.

Traitement automatisé : Le traitement automatisé des documents, que ce soit lors de leur envoi initial ou lors d'un traitement par lots, est une fonctionnalité importante d'un SGED. Cet aspect est rendu impossible à réaliser sur le serveur du SGED par l'utilisation du chiffrement. En effet, les corps de documents sont rendus confidentiels avant leur émission par le client et ne sont donc pas accessibles sur le serveur. Une solution permettant de rétablir ce type de fonctions en ayant un impact minimum sur la sécurité mise en œuvre est l'utilisation d'un agent automatique disposant des droits d'accès aux documents. Cet agent agit en position de client dans le système et doit être distinct du serveur central du SGED. Cette séparation permet de conserver le contrôle des droits d'accès par les utilisateurs, sans devoir de nouveau accorder une confiance totale au serveur central. Pour conserver une bonne granularité sur la gestion des droits, le déploiement de plusieurs serveurs de traitement automatisés ayant tous un accès limité uniquement aux données pertinentes est possible.

Recherches sur le contenu : La recherche de documents basés sur les métadonnées est toujours possible, car ces dernières sont toujours disponibles pour tous les éléments du SGED. Cependant, toute recherche sur le contenu des documents est impossible pour le serveur. Une solution naïve serait de permettre le rapatriement des documents sur le client afin de permettre leur recherche. En pra-

tique cela n'est pas possible ; les volumes de données impliqués ne permettent pas raisonnablement d'appliquer cette solution. Des méthodes de recherche pouvant s'appliquer sur des documents chiffrés existent ; certaines solutions utilisent des filtres de Bloom[7], des mécanismes de chiffrement dédiés[10], alors que d'autres s'intéressent à la recherche de mots-clés multiples sur données chiffrées[21]. Ces méthodes nécessitent pour la plupart la création d'index spécifiques et n'ont pas toujours un taux de réussite parfait. Ces index spécifiques pourraient être conservés dans les méta-données du SGED. Pour le problème du taux de réussite, il est possible d'utiliser des méthodes ne produisant que des faux positifs ou des résultats corrects. Cela permet d'appliquer l'approche "naïve" mais cette fois en n'envoyant qu'un ensemble réduit de documents pouvant correspondre à la requête de recherche. Cette solution est illustrée par la figure 4.6.

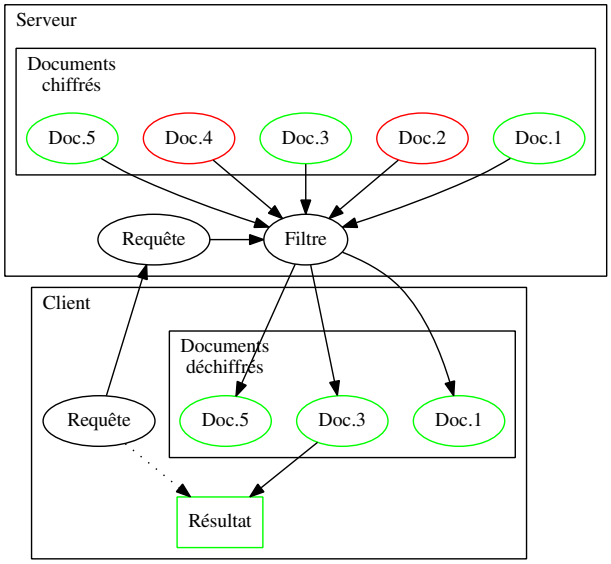

FIGURE 4.6 – Recherche sur documents chiffrés avec filtre

Les documents rouges sont exclus par le filtre ; les documents verts sont des candidats possibles après application du filtre de recherche sur données chiffrées.

Journalisation des droits utilisateurs : Les apports de sécurité décrits ne font plus intervenir activement le serveur dans l'attribution des différents droits d'accès. Pour des raisons variées (analyses, suivi de projets,...) il peut être important de conserver une trace des différents droits d'accès de chaque utilisateur du système. Deux approches sont possibles : la conservation d'informations en clair dans les trousseaux de clés sécurisés et la notification du serveur en cas de transfert de droits. La première approche nécessite l'association de données d'identification en clair à chaque clé présente dans un trousseau de clés sécurisé. Cela permet au

serveur d'analyser, à partir de ce trousseau de clés, l'ensemble des droits d'un utilisateur. L'autre approche consiste à doubler chaque transfert de droits d'un message spécifique destiné au serveur afin qu'il conserve une trace de ces transferts. Si la première approche permet une analyse a posteriori, la seconde permet de conserver une trace de quel utilisateur diffuse un droit donné à un autre. En fonction des besoins, des solutions faisant intervenir les deux approches sont envisageables.

Notion d'administrateur central : Contrairement à un système limité par un système de contrôle d'accès centralisé, il n'est pas possible pour un administrateur central de "créer" des droits pour les attribuer à un utilisateur sans le concours d'un autre utilisateur possédant déjà le droit d'accès en question. Si une telle fonctionnalité est souhaitable, une solution immédiate consiste à ce que chaque client transmette automatiquement les droits d'accès, lors de leur création, à un utilisateur spécial. Cet utilisateur n'est pas nécessairement un véritable utilisateur, mais il centralise l'ensemble des droits dans son trousseau de clés sécurisé. De cette façon, la séparation de sécurité entre le serveur central et l'accès aux documents est conservée. Les administrateurs du SGED ne sont pas nécessairement les administrateurs des données. Ces derniers disposent alors d'un accès au système de la même manière qu'un utilisateur classique ; une fois authentifié sur le système, le trousseau de clés sécurisé récupéré permet à ce compte administrateur de diffuser les droits d'accès aux autres utilisateurs.

Confidentialité révocable : Les SGED utilisé dans des contextes spécifiques peuvent présenter un besoin particulier en opposition directe avec l'apport de confidentialité proposé ici. Certaines applications nécessitent l'existence d'une option permettant sous certaines conditions de révoquer la confidentialité d'informations sensibles sans la participation du détenteur légitime des droits d'accès. L'ensemble de la sécurité du système s'appuie sur l'utilisation de clés cryptographiques générées à partir de sources aléatoires fiables. Cependant, il est possible d'utiliser un système de génération de clés alternatif compatible avec cette nouvelle contrainte. À l'utilisateur responsable de la création d'un document et au serveur central nous ajoutons un serveur tiers dit "de confiance". L'utilisation de ces trois entités permet la mise en place du mécanisme de génération de clé décrit dans la figure 4.7. Le client produit un identifiant unique nommé i lié au document et le transmet à la fois au serveur central et au serveur tiers. Chacun de ces serveurs produit un élément secret, respectivement E_i^S et E_i^T qui sont retournés au client, et conservés par chaque serveur. Le client produit la clé K_i à partir d'une fonction de hachage cryptographique H : $K_i = H(E_i^S || E_i^T)$. C'est cette clé K_i qui est attribuée au document pour les opérations de chiffrement. Ce mécanisme permet de reconstituer K_i sans l'intervention d'un utilisateur en cas d'accord entre le serveur central et le

serveur tiers.

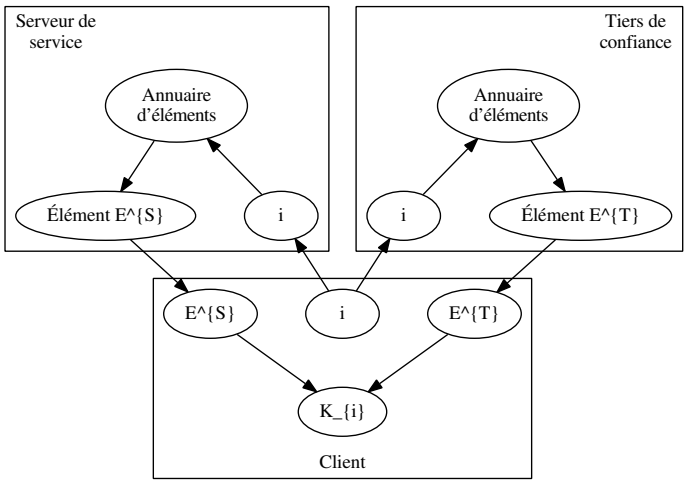

FIGURE 4.7 – Génération d'un secret révocable

La procédure commence par l'envoi de l'élément i par le client.

4.3 Mise en œuvre des solutions

Cette section décrit un ordre de mise en place des outils de sécurité au sein du SGED en trois étapes. L'objectif est de minimiser les modifications requises au sein du système tout en apportant progressivement davantage de fonctions sécurisées. La première étape permet l'intégration des bases de cryptographie nécessaires dans le système en modifiant le processus d'authentification. La deuxième étape sécurise l'ensemble des données sensibles avec des fonctions de chiffrement. Enfin, la troisième étape vise à rétablir les fonctions impactées par la mise en place de moyens de sécurité. Les solutions proposées peuvent imposer de nombreux changements à différents niveaux du SGED. Cependant, un des objectifs initiaux qui est de conserver l'accessibilité du système est omniprésent ; c'est pour cela que la majorité des solutions ont un impact réduit sur l'interface utilisateur et favorisent des opérations transparentes en termes de manipulations.

4.3.1 Protection de l'authentification

Cette première étape vise à mettre en place les outils sur lesquels toutes les applications de sécurité seront basées. Les deux solutions de sécurité concernées sont l'utilisation de connexions sécurisées et la gestion de l'authentification des

utilisateurs. Une fois intégrées, ces solutions apportent une confidentialité forte vis-à-vis des entités extérieures au système en assurant la confidentialité des échanges, ainsi que le retrait des informations sensibles des utilisateurs stockées sur le serveur.

Ces changements n'ont aucun impact sur les fonctionnalités du SGED. Leur implémentation ne nécessite que des changements mineurs sur le serveur. En revanche, ils impliquent des changements plus conséquents côté client. Ces changements sont essentiellement limités au fonctionnement interne du processus d'authentification et ne présente pas de changement d'interface utilisateur.

4.3.1.1 Protection des communications

La mise en place de moyens de communication sécurisée entre le serveur et les différents clients implique des changements qui peuvent impacter ces deux éléments. Il est nécessaire d'implémenter une sécurité de ce type au niveau applicatif. En effet, s'appuyer sur un protocole de sécurité qui n'est pas sous contrôle de l'application, comme l'utilisation de VPN [6] ou d'IPSec [7] présente un risque de sécurité ; la configuration de ces derniers n'étant pas contrôlée, ils ne peuvent pas garantir qu'ils fonctionnent de façon adéquate à l'application. L'utilisation d'un protocole de communication sécurisée au sein de l'application réduit le risque d'intervention externe en assurant de façon contrôlée que les données sont rendues confidentielles dès l'instant où elles quittent le système, qu'il s'agisse du client ou du serveur.

Deux solutions sont proposées afin de fournir une protection contrôlée des communications au niveau du SGED : l'utilisation maîtrisée de TLS et l'utilisation du protocole SVC dédié à ce cas d'usage (voir section 1.3). Dans les deux cas, il est nécessaire de centraliser les différents points de communications sur le serveur et sur le client afin d'assurer que tous les échanges tirent profit de la solution de sécurité mise en œuvre. Que la sécurité initiale soit absente, ou qu'elle soit basée sur une utilisation basique du protocole TLS, il est nécessaire de s'assurer que la solution finale soit sûre.

TLS : La sécurité effective du protocole TLS dépend de nombreux éléments dont certains ne sont pas toujours maîtrisés. Parmi les recommandations indiquées dans la section 1.1.1, les deux plus importantes sont l'authentification du serveur de façon fiable et la configuration correcte du protocole pour interdire les algorithmes peu sûrs. Pour être fiable, l'authentification du serveur ne doit pas dépendre d'informations non contrôlées par l'application. Cela implique de limiter cette authentification à des certificats connus à l'avance par le client au lieu d'utiliser un magasin de certificats fournis par le système d'exploitation. En ce qui concerne

6. Virtual Private Network, réseau privé virtuel, réseau logique pouvant regrouper des machines situées sur différents réseaux physiques distants

7. Internet Protocol Security, protocole de chiffrement des échanges au niveau IP

la configuration du protocole, il est nécessaire de choisir avec soin les algorithmes autorisés lors du processus de négociation. Afin de faire ce choix, le mainteneur de l'application doit effectuer une veille technologique sur les recherches en sécurité afin de suivre l'évolution des vulnérabilités. Si l'utilisation "naïve" du protocole TLS peut présenter des risques, une utilisation ainsi maîtrisée est beaucoup plus sûre. Elle nécessite néanmoins une attention importante de la part du développeur du SGED en charge de cet aspect.

SVC : Le protocole SVC est conçu pour répondre aux contraintes d'une application distribuée comme un SGED. Le processus de négociation est limité à un ensemble d'algorithmes sûrs et son fonctionnement est intégré dans l'application afin de garantir un support équivalent de ces algorithmes aussi bien par le serveur que par le client. De plus, le mécanisme d'authentification du serveur est lui aussi directement lié à l'application qui se charge de cette validation suivant des méthodes classiques, afin de limiter toute interaction avec des éléments externes non contrôlés. Une fois implémentées, les évolutions du protocole sont liées aux mises à jour de l'application, évitant ainsi les problèmes de conflits de versions. Ces évolutions intégreront les changements d'algorithmes rendus nécessaires par l'évolution des recherches en sécurité et en cryptographie. Les restrictions d'algorithmes et l'intégration du protocole dans le code de l'application réduisent la complexité d'utilisation par rapport à d'autres solutions externes.

4.3.1.2 Processus d'authentification

Le processus d'authentification initiale de notre modèle de SGED utilise l'envoi d'un mot de passe comme preuve de l'identité d'un utilisateur. S'il est utilisé sur un canal de communication non sécurisé, ce procédé est extrêmement vulnérable. Cependant, même dans le cas d'une utilisation sur un canal sécurisé, il existe des problèmes rendant cette approche inadaptée à un système pour lequel on souhaite pouvoir prouver de façon fiable le fonctionnement, notamment pour des raisons de preuve de la présence effective d'un utilisateur donné lors du processus d'authentification. Afin de disposer d'un système fiable à chaque étape, le processus d'authentification doit utiliser des méthodes permettant d'une part de garantir que leur résultat est correct et d'autre part que seul l'utilisateur dispose des informations permettant de l'authentifier. Pour cela on met en place un processus d'authentification basé sur des outils cryptographiques, notamment des fonctions de hachage.

Au niveau du SGED, cette mise en place initiale implique des changements importants sur le fonctionnement interne du client, mais peut être mis en place sans modification du serveur. En effet, l'élément essentiel est ici la confidentialité de l'information d'authentification de l'utilisateur vis-à-vis du serveur. Cela implique

une transformation cryptographique mise en œuvre par le client. La mise en place du protocole d'authentification complet décrit dans la section 2.4 aura lieu à la fin de cette étape. Ici nous mettons en place la partie présentée dans la figure 2.3.

Modification du client : Il est nécessaire de modifier le comportement du client après la saisie des informations par l'utilisateur. Le chemin de départ est le suivant : saisi du mot de passe M, envoi de M au serveur, réception d'une réponse indiquant le succès ou l'échec. Afin de préparer l'utilisation de fonctions de sécurité basées sur un trousseau de clés sécurisé et pour isoler la sécurité du mot de passe vis-à-vis du serveur, il faut intercaler un processus entre la saisie du mot de passe et son envoi. Le mot de passe M est transformé par un procédé cryptographique : une fonction de hachage. Un élément M_1 est produit par le calcul $M_1 = H(1||M)$, où H est une fonction de hachage cryptographique et $||$ est l'opération de concaténation. C'est cet élément M_1 qui est alors envoyé au serveur et joue le rôle de mot de passe. Cette étape permet aussi à la récupération d'un élément M_2, calculé de façon similaire en préfixant M par 2, qui sert au déverrouillage du trousseau de clés sécurisé.

Une fois le processus d'authentification validé par le serveur, le client doit alors rapatrier son trousseau de clés sécurisé. Cette opération utilise les mécanismes habituels du SGED qui considère le trousseau de clés comme un document quelconque dont le contenu lui est inconnu. Une fois récupéré, l'élément M_2 est utilisé pour déverrouiller le trousseau de clés sécurisé. Ces opérations n'ont pas d'effets visibles sur l'interface utilisateur, car elles sont effectuées de façon transparente. La dernière modification à opérer sur le client est la mise en place d'une interface permettant l'accès aux clés du trousseau de clés par les autres modules sécurisés.

Impact sur le serveur : Dans cette mise en place initiale le serveur ne nécessite pas d'adaptation particulière. Le "mot de passe" reçu par le serveur, qu'il s'agisse de l'élément M ou de l'élément M_1 décrit précédemment est traité de la même façon. Dans le cas d'une base d'utilisateurs existantes, il est nécessaire de demander aux utilisateurs de modifier leur mot de passe afin que le serveur puisse enregistrer leur nouvel élément M_1 à la place du mot de passe initial. Afin de faciliter cette transition, l'ajout d'une information dans la base indiquant les comptes utilisateurs n'ayant pas effectué cette mise à jour peut être utile, mais n'est pas strictement nécessaire. Dans le cas d'une nouvelle base d'utilisateurs, il n'y a pas de situation initiale à mettre à jour ; la seule information connue par le serveur est l'élément M_1 de chaque utilisateur.

4.3.1.3 Protocole d'authentification distant

Afin de pouvoir bénéficier de façon sécurisée de différentes modalités d'authentification côté client, en particulier des systèmes non basés sur des mots de passe, la mise en place d'un protocole générique d'authentification adapté est nécessaire. Ce protocole, décrit dans la section 2.4, permet l'utilisation de façon unifiée de moyens variés comme des mots de passe, des jetons d'authentification PKCS#11 (de type CPS [8]), etc. L'authentification mise en place permet d'authentifier un même utilisateur à travers différentes modalités, dont la disponibilité peut varier sur différents périphériques, tout en conservant l'ensemble des fonctionnalités auxquelles il a accès.

En se plaçant dans le contexte des changements réalisés dans la section 4.3.1.2, le protocole d'authentification s'insère entre l'envoi de l'information d'authentification au serveur et son traitement par ce dernier. Les informations d'authentification sont encapsulées dans un format particulier permettant d'assurer cryptographiquement la fiabilité de l'échange via une signature numérique spécifique à la session courante. La réponse du serveur est elle aussi signée numériquement afin d'assurer le client que la réponse provient bien d'un serveur légitime.

Le protocole d'authentification distant se place à l'extrémité du processus d'authentification, juste avant l'envoi des informations au serveur. Tous les éléments nécessaires à son fonctionnement (notamment le secret transformé M_1) sont alors disponibles. Le protocole gère également le traitement de la réponse du serveur, afin de vérifier la validité de celle-ci. Côté serveur, un traitement analogue est réalisé : le protocole d'authentification extrait les informations de l'utilisateur depuis la requête reçue, transmet ces informations à l'application pour qu'elle puisse les vérifier et encapsule la réponse de façon sécurisée pour le retour vers le client.

Ainsi, que ce soit pour le client ou pour le serveur, le reste du processus d'authentification (récupération des informations et envoi de la réponse) reste non modifié. Le protocole d'authentification s'assure en arrière-plan que les informations échangées sont authentiques.

4.3.2 Protection des données

Une fois le processus d'authentification sécurisé et le mécanisme de trousseau de clés sécurisé mis en places, la protection des contenus est possible. Le corps des documents et les méta-données associées nécessitent deux mécanismes différents : un système de chiffrement symétrique et un système de signature. Le système de messagerie intégrée au SGED peut lui aussi être sécurisé lors de cette étape.

8. Carte de Professionel de Santé, carte à puce permettant d'authentifier les professionels de santé

4.3.2.1 Corps des documents

La protection du corps des documents s'applique essentiellement du côté client du système. Pour l'appliquer il est nécessaire de modifier la façon dont sont envoyés et reçus les documents. Chaque document chiffré dispose dans ses méta-données de l'identifiant de la clé secrète utilisée pour le protéger. Cette clé secrète est créée lors de la création initiale du document et n'est présente que dans les trousseaux de clés des utilisateurs ayant accès au document concerné. Le cycle chiffrement/déchiffrement est illustré par la figure 4.4 page 135.

Pour récupérer le corps d'un document, le client envoi une requête classique au serveur. Ce dernier renvoi le corps du document chiffré sans aucune action spécifique de sa part. Une fois reçu par le client, la clé secrète identifiée dans les méta-données est extraite du trousseau de clés et utilisée pour déchiffrer le corps du document. Le résultat de cette opération est alors présenté à l'utilisateur comme s'il n'y avait pas eu d'opération intermédiaires : elles sont totalement transparentes et gérées par le système.

Lors de l'envoi initial ou de la mise à jour du corps d'un document, le processus inverse est mis en place. Avant son envoi au serveur, le corps du document est chiffré avec la clé secrète associée et le résultat de cette opération est envoyé au serveur. Ce dernier manipule le corps du document chiffré de la même façon qu'un document générique dans un format qui lui est inconnu. Il n'est pas nécessaire de mettre en place un traitement particulier des documents chiffrés côté serveur.

Une fois qu'un document existe dans le SGED, la clé secrète qui lui est associée représente le droit d'accès au document. La création de cette clé secrète peut se faire de deux façons. En considérant que la clé secrète doit être effectivement connu des seuls utilisateurs autorisés, elle peut être générée par le client à l'origine du document. Si au contraire un mécanisme de confidentialité révocable est envisagé, la production de cette clé secrète doit faire intervenir le serveur et un tiers en charge de la confidentialité du système. Dans les deux cas, l'impact sur l'interface utilisateur est nul : le processus d'obtention de la clé secrète est transparent. Cependant, l'implémentation sous-jacente doit prendre en compte cette possibilité. En particulier, le cas de l'intervention d'un tiers nécessite la mise en place d'interactions sécurisées avec ce dernier.

4.3.2.2 Méta-données

Les méta-données ne sont pas rendues confidentielles pour le serveur. En effet, elles servent à différents traitements essentiels et ne présentent pas la même importance que le corps du document. Afin de limiter leur modification par des utilisateurs non autorisés, un mécanisme de signature numérique validant le droit de modification est mis en place. Cette signature numérique utilise une paire de

clés privée/publique propre au document. Cette dernière est générée au même moment que la clé secrète utilisée pour protéger le corps du document. Puisqu'elle n'est utilisée qu'à des fins de validation de droit et n'est pas liée à un utilisateur donné, cette génération n'a pas besoin de bénéficier d'un éventuel mécanisme de levée de secret. Une fois produite, la partie publique de cette paire de clés est envoyée au serveur qui la conserve dans un annuaire spécifique liant clés publiques et identifiants de documents.

Il n'y a pas de changement sur le fonctionnement des méta-données : elles sont accessibles au serveur en clair et leur échange utilise le moyen de communication sécurisée mis en œuvre par ailleurs. Lors de la mise à jour de ces méta-données, la requête de modification est associée à une signature numérique produite à partir de la clé privée associée. La récupération de la clé privée depuis le trousseau de clés ainsi que le processus de signature numérique se font de façon transparente à partir de l'identifiant de document et trousseau de clés déverrouillé. Du côté du serveur, un processus de vérification doit être mis en place basé sur l'annuaire de clés publiques associés à chaque document. Cette vérification permet de valider l'enregistrement ou non des modifications des méta-données.

La mise en place des opérations cryptographique associées côté client s'appuie sur les composants déjà utilisés par ailleurs et n'impacte pas l'interface utilisateur. En revanche, un travail spécifique doit être réalisé sur le serveur du SGED. Ce dernier doit implémenter l'annuaire de clés publiques associées aux identifiants de document. Cette base de données spécifiques ne nécessite pas d'aspects de confidentialité, les clés publiques n'étant pas des données sensibles de ce point de vue. En revanche, l'intégrité des données doit pouvoir être assurée, une corruption de ces informations pouvant empêcher toute manipulation ultérieure des méta-données des documents. En plus de cet annuaire, le serveur doit également mettre en place un mécanisme de validation de signature pouvant bloquer la mise à jour des informations en cas de problème. Dans ce cas, un simple retour négatif à l'utilisateur suffit ; en cas de réussite la récupération par le client des nouvelles méta-données enregistrées confirme que l'opération s'est déroulée avec succès.

4.3.2.3 Messagerie

La sécurisation de la messagerie interne s'appuie sur un élément commun à la protection des méta-données : un annuaire de clés publiques. Cette fois cependant, il s'agit des clés publiques associés à chaque utilisateur et pas aux documents. Trois éléments sont donc nécessaires à la mise en place d'une messagerie sécurisée telle que décrite précédemment : une paire de clés privée/publique propre à chaque utilisateur, l'annuaire de clés publiques conservés sur le serveur et les mécanismes de chiffrement et de signatures numériques utilisés par les clients. En plus des éléments indiqués ci-dessus, l'approche sécurisée s'appuie sur un mécanisme de

boîtes aux lettres existant préalablement sur le serveur et permettant déjà des échanges de messages hors-ligne. Nous rappelons que le système de gestion de la messagerie est également utilisé pour l'envoi de clés représentant des droits d'accès, qui sont dans ce cas des messages spéciaux.

Paire de clés des utilisateurs : Chaque utilisateur doit posséder une paire de clés privée/publique qui lui est propre. Sa clé privée est conservée dans son trousseau de clés sécurisé, alors que la clé publique est connue de tous, grâce à l'annuaire présent sur le serveur. Cette paire de clés permet de mettre en œuvre des mécanismes de cryptographie asymétrique utilisés notamment pour la signature numérique et l'envoi d'information confidentiels hors-ligne. La clé privée ne doit être connue que de l'utilisateur ; elle doit donc être générée par ce dernier et ne jamais transiter en clair par un tiers. Dans le cas de nouveaux utilisateurs du SGED comme pour la migration d'anciens utilisateurs, la génération de cette paire de clés se fait lors de leur première connexion après la mise en place du système. Si le client détecte, lors du déverrouillage du trousseau de clés sécurisé, que l'utilisateur ne dispose pas de sa paire de clés personnelle, il peut en générer une sans intervention de l'utilisateur. Elle est alors ajoutée au trousseau de clés et la partie publique envoyée au serveur via la session authentifiée de l'utilisateur.

Annuaire de clés publiques : Comme pour la protection des méta-données, le SGED doit conserver un annuaire des clés publiques de chaque utilisateur. Cependant, contrairement à l'annuaire utilisé pour la modification des méta-données, les clés publiques des utilisateurs doivent être rendues accessibles à tous. Cette accessibilité peut se faire de plusieurs façons, comme la mise en place d'un protocole de requêtes simple permettant d'interroger le serveur pour récupérer la clé d'un utilisateur, ou l'utilisation de "méta-documents" non confidentiels dont l'identifiant serait lié aux identifiants utilisateurs. Cette deuxième proposition présente l'avantage de mettre à profit les fonctions existantes du SGED, en stockant les clés publiques des utilisateurs comme des documents publics. Dans tous les cas, la propriété essentielle des annuaires de clés publiques est de pouvoir garantir l'authenticité des clés fournies. Une solution permettant de garantir cette authenticité est l'utilisation d'une structure de données de type dictionnaire authentifié[5]. Une telle structure de données permettrait au SGED de pouvoir authentifier la validité des clés qu'il fournit aux utilisateurs tout en réduisant le coût en bande passante de cette preuve d'authenticité.

Mécanismes cryptographiques : Pour fonctionner, le système illustré sur la figure 4.5 page 136 doit pouvoir réaliser deux types d'opérations sur les messages : leur chiffrement/déchiffrement et leur signature/vérification. Ces opérations sont,

comme pour les autres opérations, transparentes pour l'utilisateur. L'interface courante d'envoi de messages présente deux éléments principaux : le contenu du message et le destinataire. Ces informations sont suffisantes pour la mise en place des outils de chiffrement. L'envoi d'un message fait donc l'objet, sur le client, des opérations de signature numérique et de chiffrement faisant intervenir respectivement la clé privée de l'émetteur et la clé publique du destinataire. La réception d'un message fait l'objet d'un déchiffrement et d'une vérification de signature utilisant cette fois, respectivement, la clé privée du destinataire et la clé publique de l'émetteur. Le client doit donc disposer des mécanismes cryptographiques adéquats et les appliquer au moment de l'émission et de la réception d'un message. Le serveur ne nécessite aucune modification à ce stade : le système initialement présent de boîte aux lettres est utilisé et contient les messages chiffrés, sans incidence sur leur transmission.

4.3.2.4 Transfert de droits

Le changement le plus impactant tant sur l'interface utilisateur que sur le fonctionnement général du SGED est celui de la gestion des droits d'accès. En effet, chaque utilisateur devient responsable de la propagation des droits dont il dispose. Ce changement nécessite une modification de l'interface utilisateur pour présenter cette possibilité. De même, le serveur ne peut plus attribuer de droits aux utilisateurs par lui-même.

Une fois l'interface des clients adaptée à ce nouveau fonctionnement, il reste à effectuer le transfert effectif de droits. Ce dernier se fait en utilisant le système de messagerie sécurisée. L'encapsulation d'une clé donnée, correspondant à un droit d'accès, permet un envoi effectivement confidentiel à un autre utilisateur. Ce changement nécessite, côté client, un traitement spécifique : l'encapsulation et la récupération de clés doit se faire sans intervention de l'utilisateur. En revanche, le serveur n'a qu'un rôle dans cet envoi de droits : la gestion de la boîte aux lettres du destinataire afin de lui faire parvenir le message spécial.

4.3.3 Fonctions avancées

4.3.3.1 Mise en œuvre d'outils avancés pour clients légers

Les applications de type "client léger", s'exécutant dans un navigateur Web, utilisent généralement des solutions basées sur la technologie Java pour accéder à des fonctionnalités non disponibles depuis les langages de scripts comme JavaScript. Cette utilisation de Java dépend essentiellement d'une autre technologie appelée NPAPI[9], permettant l'écriture d'extensions de navigateurs utilisant du

9. Netscape Plugin API, interface d'extensions pour navigateurs web

code natif. Ces interfaces NPAPI sont progressivement retirées des navigateurs modernes[61, 65], rendant impossible l'utilisation de Java à moins d'importants développement visant à le rendre compatible avec les solutions remplaçant NPAPI. La figure 4.8 présente la dépendance des clients légers aux deux technologies Java et NPAPI.

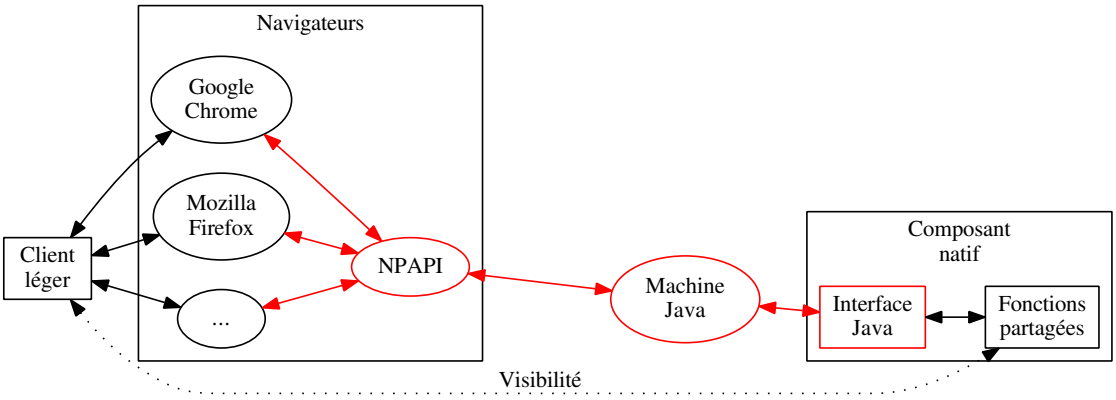

FIGURE 4.8 – Utilisation de Java depuis un client léger

Les éléments indiqués en rouge sont ceux appelés à disparaitre. Les éléments carrés représentent le développement du client léger.

Une solution permettant de conserver le même niveau de fonctionnalité en retirant la dépendance au langage Java est proposée sur la figure 4.9. Elle est constituée d'interfaces spécifiques pour les principaux navigateurs. Ces éléments ont pour unique rôle de permettre l'appel de fonctions depuis une application JavaScript vers du code natif, comme on le faisait avec un plugin Java. Ces interfaces sont elle-même connectée à une interface de développement commune exposant les fonctionnalités souhaitées du code natif. La solution envisagée est basée sur un IDL [10] permettant de généraliser le développement de ces modules d'interfaces.

Cas de l'utilisation d'une interface PKCS#11 : Un cas particulier étudié au cours de cette thèse a été l'utilisation d'un appareil d'authentification accessibles depuis une interface PKCS#11 [11]. En fonction du périphérique utilisé, une bibliothèque logicielle spécifique est utilisée pour fournir les fonctions de sécurité correspondantes. Dans le cas d'un client léger pour une application de sécurité, l'accès à ces fonctionnalités peut lui aussi fonctionner de la façon décrite précédemment. L'écriture d'une extension jouant le rôle d'agent PKCS#11 permettrait

10. Interface Description Language, syntaxe permettant de décrire des interfaces de haut-niveau souvent utilisé pour simplifier les communications inter-processus.

11. La spécification #11 définit une API générique pour l'utilisation de périphériques cryptographiques

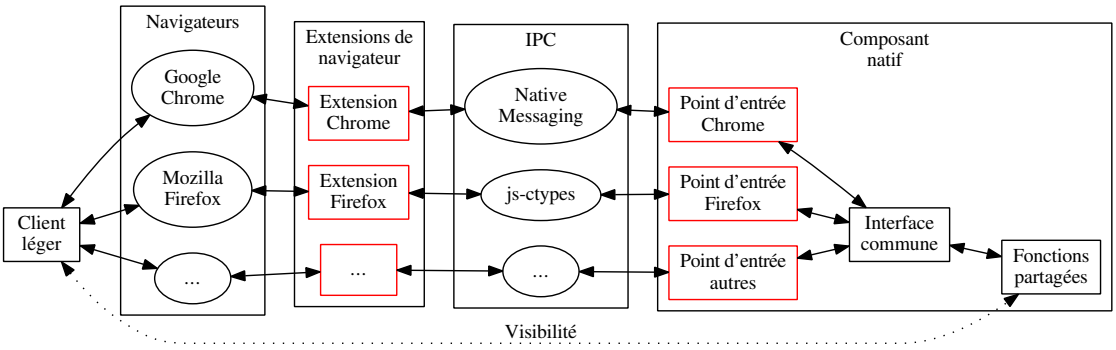

FIGURE 4.9 – Chemin d'appel de code natif depuis un client léger

Les éléments carrés représentent le développement du client léger. Les éléments rouges sont ceux dont la création peut être automatisées de façon générique.

d'apporter ces fonctionnalités dans les différents navigateurs. Cet agent chargerait à la demande les différentes bibliothèques PKCS#11 et exposerait l'interface générique du standard au client léger.

4.3.3.2 Restauration de fonctions initiales

Comme indiqué dans la section 4.2.6, certains apports de sécurité entraînent une altération des fonctionnalités du SGED. Il est possible de restaurer certaines fonctionnalités en utilisant des approches différentes, spécifique au modèle du SGED sécurisé. Les réponses proposées en ce qui concerne les possibilités de traitement automatisées et de journalisation des attributions de droits ne nécessitent pas de modifications particulières ni sur le client ni sur le serveur. En revanche, les problèmes de recherche sur le contenu, de notion d'administrateur de droits et de confidentialité révocable ont un impact sur l'application.

Recherche sur le contenu : En partant du fonctionnement initial dans la solution non sécurisée, la recherche de documents peut se faire à partir de nombreux critères incluant les méta-données et le contenu du corps du document. Cette recherche est faite par le serveur qui ne retourne au client que les éléments pertinents. Dès lors que le corps des documents est rendu inaccessible au serveur, il n'est plus possible d'y effectuer des recherches. Des travaux de recherches existent dans ce domaine et sont notamment mentionné dans la section 4.2.6. Leur mise en œuvre et complexe et dépend de nombreux paramètres selon la méthode utilisée. Certaines approches nécessitent la création d'index spécialisés. L'intégration de fonctionnalités de recherche de ce type implique des modifications importantes à la fois pour le client et le serveur.

Administrateur de droits : La mise en place d'un administrateur central pouvant distribuer des droits d'accès aux utilisateurs n'est pas complexe. Une fois le SGED sécurisé mis en place, il est possible de simplement transmettre les droits vers un utilisateur spécial à chaque création de document. Il n'est donc pas nécessaire de faire des changements particuliers sur le serveur, ni de faire des changements profonds sur le client. Toutefois, l'automatisation du processus d'envoi de droits à destination de ce compte spécial "administrateur" est souhaitable et doit donc être intégrée au processus de création de documents.

Confidentialité révocable : La possibilité d'utiliser un secret révocable décrite dans la section 4.2.6 nécessite pour sa mise en place des changements au SGED. En l'absence de cette possibilité, la création d'un nouveau document par un client est immédiate ; la génération de la clé secrète pour le document se fait à la demande et n'impose pas de délai sensible à l'utilisateur. En revanche, l'utilisation d'un procédé de secret révocable faisant intervenir le serveur du SGED et un serveur tiers pose des problèmes de délais et de possibilités d'erreurs qui doivent être gérées par le client. En lieu et place de la génération immédiate de la clé secrète, il faut maintenant envoyer deux requêtes à deux serveurs différents. Ces envois imposent des délais supplémentaires et des risques d'erreurs (notamment en cas de non disponibilité) pouvant limiter le fonctionnement du client. Du point de vue des deux serveurs, leur tâche sur ce point est limitée à la production d'un élément secret chacun. Cet élément est produit lorsque le client demande une clé pour la création d'un document et est conservé associé à l'identifiant du document.

4.3.4 Synthèse des apports de sécurité

La figure 4.10 résume les changements proposés afin de sécuriser le fonctionnement du système. Les deux premières étapes de migration y sont indiquées.

Les modifications à apporter au niveau du serveur sont minimales. Elles concernent le protocole d'authentification, la validation des changements de méta-données et la gestion des divers annuaires de clés publiques. Au contraire, les modifications sont plus nombreuses côté client. En effet, l'essentiel des changements de sécurité sont appliqués par le client, avant transmission de l'information. Ces modifications touchent le protocole d'authentification ainsi que la méthode de saisie des informations d'authentification, la gestion des droits utilisateurs et le chiffrement des données (documents et messages).

Ces changements et ajouts s'appuient pour fonctionner sur des éléments de base du SGED. Le protocole d'authentification complet tire parti des mécanismes de vérifications déjà présents. Le transfert des documents et des messages n'est pas modifié vis-à-vis du serveur. La gestion des droits, par l'intermédiaire du trousseau

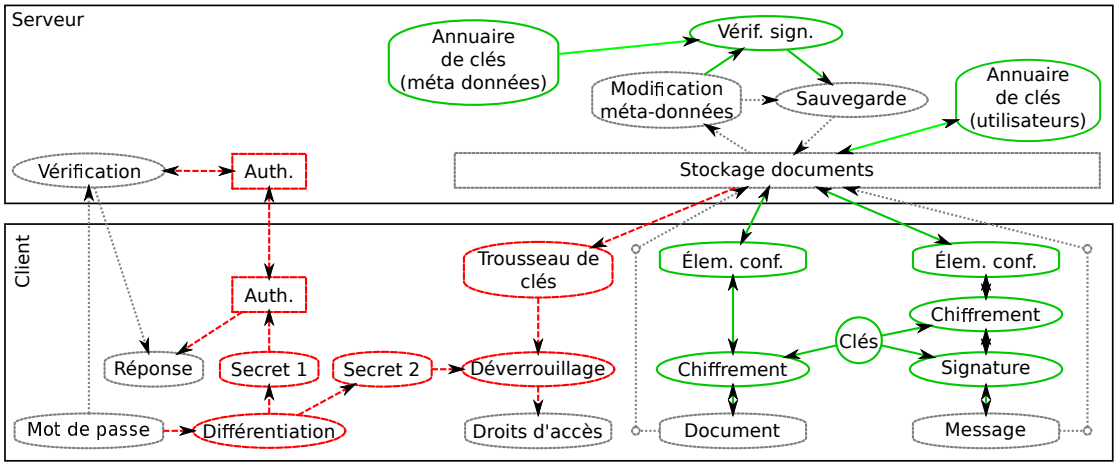

FIGURE 4.10 – Résumé des apports de sécurité à l'application du SGED

Les chemins indiqués en gris (pointillés) représentent le fonctionnement non sécurisé. Les éléments rouges et verts (resp. tiretés et lignes pleines) représentent la première et la deuxième étape de la migration. "Auth." représente le protocole d'authentification de la section 2.4

de clés sécurisé, s'appuie sur les capacités du SGED à transmettre des documents pour leur récupération et sur la capacité à échanger des messages pour la transmission de droits à d'autres utilisateurs. Enfin, la modification des méta-données fait l'objet d'une validation contrôlée de façon indépendante par le serveur, n'imposant pas de changements dans le fonctionnement global des échanges impliqués.

L'ensemble des outils cryptographiques nécessaires doit être disponible sur tous les supports et pas uniquement sur le serveur. En particulier, les différents supports matériels permettant de faire fonctionner l'application cliente doivent pouvoir disposer de fonctions de chiffrement, gestion de clés et signature numérique. Dans le cas du serveur et d'un client "lourd", cette restriction n'est pas un problème majeur, car on a une grande liberté sur les outils utilisables. En particulier, la plate-forme de sécurité logicielle Arcana décrite au chapitre 3 répond complètement aux besoins exprimés ici. En ce qui concerne les clients dit "légers", le problème est évoqué dans la section 4.3.3.1 avec une proposition de solution.

4.4 Travaux futurs

Les solutions proposées dans ce chapitre permettent la migration d'un SGED générique initialement non sécurisé vers un système ayant des propriétés d'intégrité, d'authenticité et de confidentialité appliquées aux documents gérés ainsi qu'aux communications non sensibles. Il reste plusieurs points sur lesquels des améliorations sont possibles.

Tout d'abord, des pistes sont proposées pour permettre la recherche sur les données chiffrées. Cela n'est pas applicable directement ; un travail important est nécessaire pour trouver des méthodes adaptées aux contraintes du SGED et implémenter ces méthodes.

Ensuite, la manipulation des documents côté client se fait toujours dans un cadre "non sécurisé". S'appuyer sur la bonne foi des utilisateurs n'est pas suffisant : qu'elle soit volontaire ou non, une fuite de données est alors possible à ce moment-là. L'intégration du SGED aux applications métiers permettant la manipulation des documents est possible de plusieurs façons. Parmi les pistes possibles, l'utilisation de moyens de protection intégrés aux applications (comme l'intégration, pour les applications en ayant le support, de mécanismes de cryptographie intégrés) est envisageable, mais n'est pas une solution pratique sur le long terme notamment à cause des risques de changements de ces moyens. Une autre piste est le remplacement à l'exécution des fonctions d'accès aux fichiers des applications utilisées. Les applications n'auraient alors besoin d'aucune modification ni fonction spécifique pour accéder aux documents chiffrés. Le comportement d'une application ainsi "surchargée" serait alors normal en tout point, à l'exception des opérations de lecture et d'écriture de documents sécurisés qui mettraient en œuvre le chiffrement du SGED de façon transparente.

Enfin, l'intégration flexible de moyens d'authentification est supporté par la mise en place du protocole d'authentification distant du chapitre 2. Il reste toutefois un travail à faire pour que toutes les modalités d'authentification envisagées permettent, d'une façon ou d'une autre, l'utilisation du mécanisme de "secret secondaire" permettant le déverrouillage du trousseau de clés sécurisé. Si l'utilisation d'un mot de passe ou d'un mécanisme de signature déterministe est immédiat, d'autres modalités peuvent poser des problèmes d'incompatibilité avec cette approche ; dans ces cas-là l'utilisation simultanée d'une modalité d'authentification externe et d'un mot de passe est une solution, mais il est important de chercher des solutions moins contraignantes pour l'utilisateur.

Enfin, le modèle choisi ici pour servir de base à l'application de fonctions de sécurité est un modèle générique ; même s'il est basé sur une application existante, sa transposition à d'autres outils peut nécessiter des adaptations particulières. C'est pour cette raison que les fonctions évoquées sont élémentaires, afin de permettre une adaptation rapide vers des modèles plus complexes. Cependant, il est possible que des fonctions plus avancées existent et soient impactées par les changements causés lors de l'application des solutions de sécurité. Cela inclue des fonctions de génération automatique de documents à partir d'autres documents, des mécanismes d'extraction de données depuis le corps des documents, etc. Cette analyse doit être faite au cas par cas en fonction des applications ciblées.

CONCLUSION

Les travaux présentés dans les différents chapitres de cette thèse partagent un but commun : l'application d'outils cryptographiques comme réponse à des problèmes concrets. Plutôt que de concevoir des solutions génériques en se basant sur des propriétés de sécurité théoriques, nous avons pris le parti d'analyser les conditions d'utilisations des systèmes communicants comme point de départ de nos réflexions. Cette approche a permis de tenir compte de contraintes opérationnelles pratiques lors de la conception des protocoles et mécanismes de sécurité ; contrairement aux approches classiques consistant à appliquer une solution existante à un problème pratique et nécessitant l'adaptation du problème à la solution. Pour cela, nous avons choisi comme support de travail une application particulière en la forme d'un SGED [1].

Ces systèmes sont intéressants d'un point de vue étude de sécurité, car ils regroupent de nombreuses problématiques liées aux systèmes communicants. C'est pour cette raison que nous nous sommes concentrés sur la sécurisation des communications et les problèmes d'authentifications. Ces travaux sont ensuite appliqués aux fonctions des SGED afin de sécuriser l'utilisation et les données grâce à des mécanismes cryptographiques plutôt que par des contrôles d'accès non garantis. Afin de pouvoir fournir le support logiciel nécessaire à l'application des différents outils de sécurités proposés, une plate-forme logicielle de sécurité nommée Arcana a été développée.

La création d'un protocole de communication sécurisée spécifique a été motivé par un manque dans les solutions existantes comme TLS [2] et SSH [3]. Bien que sécurisés, ces derniers présentent une propriété d'interopérabilité importante visant à permettre l'établissement de connexions sécurisées entre des applications provenant de différents fournisseurs. Pour permettre cette interopérabilité une étape de négociation complexe a pour conséquence d'augmenter la surface d'attaque de ces derniers, en plus d'affaiblir la sécurité en restant compatible avec d'anciennes

1. Système de Gestion Électronique de documents, permettant la création, manipulation, et distribution de documents de nature variée

2. Transport Layer Security, protocole de communication sécurisée remplaçant SSL

3. Secure Shell, protocole de communication sécurisée permettant notamment l'ouverture d'une invite de commande sur un système distant

versions vulnérables. Afin de proposer une alternative à ces protocoles de communication sécurisée génériques, une solution spécifique a été conçue nommée SVC [4], basé sur l'observation suivante : il est possible de catégoriser les communications dans deux grandes catégories. D'une part se trouvent les communications ayant un besoin d'interopérabilité ; elles concernent les échanges entre différentes applications provenant de différents développeurs. Dans ce cas, l'interopérabilité est un élément clé du fonctionnement et les protocoles classiques s'appliquent. D'autre part existe une classe de communications pour lesquelles l'interopérabilité n'est pas nécessaire, comme dans le cas d'une application distribuée : les différentes instances proviennent toutes d'une même source et n'ont donc aucun risque d'incompatibilité.

C'est sur la base de ce constat que le protocole SVC a été conçu de zéro en tenant compte à la fois des contraintes de sécurités apportées par les protocoles existants et des contraintes opérationnelles qui s'appliquent à cette classe de communications. Ainsi, des propriétés supplémentaires comme la compatibilité avec des moyens d'authentifications variés et la confidentialité des informations d'identités ont été ajoutées. Le résultat de ce travail est la mise au point d'un protocole flexible adapté à cette classe de communications ayant un risque de vulnérabilité et une consommation de ressources inférieurs aux solutions alternatives.

Le travail sur les protocoles d'authentifications présentés s'appuie sur un besoin lié à la sécurisation des systèmes. Les solutions courantes se concentrent sur la fourniture d'une réponse binaire à une requête d'authentification : réussite ou échec. Afin de concevoir un système sécurisé à base de cryptographie, il est nécessaire d'une part d'obtenir des éléments cryptographiques sûrs liés à la session utilisateur et d'autre part de faire participer les informations d'authentification aux différents traitements cryptographiques mis en œuvre. C'est pour cela que nous avons conçu un protocole d'authentification distant spécifique. Ce dernier permet l'utilisation de modalités d'authentifications variées et fournit au terme de son exécution des éléments cryptographiques permettant l'application d'algorithmes de chiffrement et de signatures numériques tout au long de la session utilisateur.

Au-delà de la conception d'un protocole d'authentification compatible avec des applications cryptographiques ultérieures, un mécanisme de transferts de clés est proposé. Ce dernier, bien qu'indépendant du protocole d'authentification distant, pose des contraintes sur l'utilisation des moyens d'authentifications de l'utilisateur, notamment sur la séparation entre les informations connues du système distant et les informations fournies par l'utilisateur lors de son authentification. Une fois en place, ce mécanisme permet le transfert d'un trousseau de clés sécurisé permettant d'appliquer diverses méthodes cryptographiques dans un contexte de mobilité sans

4. Secure Virtual Connector, connecteur virtuel sécurisé, protocole de communication sécurisée dédiées aux communications internes des applications

imposer d'action particulière à l'utilisateur.

En trame de fond de ces travaux a eu lieu le développement d'une plate-forme logicielle de sécurité nommée Arcana. Construite autour d'une bibliothèque logicielle, elle permet la mise en œuvre de tous les algorithmes cryptographiques nécessaires pour supporter les solutions proposées. En plus d'être complète en termes d'algorithmes, elle est également conçue pour être portable sur différentes architectures afin de permettre son utilisation dans toutes les circonstances imposées par les cas d'usages considérés, en particulier pour des usages mobiles. Parmi les points clés de cette plate-forme se trouve l'utilisation d'algorithmes basés sur des courbes elliptiques pour les opérations de chiffrement asymétriques permettant d'apporter une sécurité équivalente aux systèmes classiques pour une taille de clé inférieure.

Ces développements ont vu, en plus de l'inclusion de nouveaux algorithmes et de nouvelles primitives cryptographiques, l'apport d'optimisations à plusieurs niveaux pour améliorer les performances dans de nombreux cas d'usage et sur différentes architectures. Les optimisations essentielles concernent la gestion de la mémoire dynamique utilisant des mécanismes spécifiques aux applications cryptographiques et l'utilisation de composants matériels dédiés à l'implémentation de certains algorithmes. Le résultat est une bibliothèque logicielle complète ayant une grande accessibilité autant en termes de disponibilité que de facilités d'utilisation. D'une part, la portabilité sur différentes architectures et différents systèmes d'exploitation permet une utilisation sur l'essentiel des systèmes actuels et d'autre part la conception des API [5] permet une utilisation aisée à la fois des primitives cryptographiques et des protocoles de plus haut niveau.

L'ensemble de ces travaux se voie appliqués à un cas d'usage pratique en la forme d'un SGED. Ce type d'application présentant des besoins importants en termes de sécurité l'application de mécanismes cryptographiques permet d'apporter une réponse fiable et incontournable contrairement aux systèmes classiques de contrôle d'accès. Les méthodes et outils proposés ont pour but d'intégrer la sécurité des données dans leur forme plutôt que dans les outils permettant de les manipuler. Afin de permettre cet apport de sécurité cryptographique au fonctionnement d'un SGED, un ensemble de solutions spécifiques a été proposé permettant une migration depuis un système non sécurisé. Ces solutions tirent profit des protocoles développés durant cette thèse ainsi que de réflexions et mécanismes spécifiques aux SGED.

Bien que les développements proposés pour les apports de sécurité au sein d'un SGED soient basés sur la description d'un système générique, ce dernier à lui-même été conçu pour représenter l'architecture et les fonctionnalités d'un système

5. Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

existant. À la fin de l'application des solutions techniques proposées, l'ensemble des fonctionnalités centrale du SGED ont un fonctionnement sécurisé à partir d'outils cryptographiques. Le résultat de ce travail en particulier est donc la mise en place d'un système assurant la confidentialité et l'intégrité des documents et la protection de l'identité des utilisateurs. Cette sécurité est supportée par des méthodes et outils ayant un impact minimal sur l'expérience utilisateur. Ce dernier aspect est essentiel dans la conception de systèmes sécurisés, car un système à l'utilisation malaisée incite les utilisateurs à simplement ne pas l'utiliser, ce qui rend toutes autres mesures de protection inefficace.

Ainsi, l'ensemble des travaux présentés dans cette thèse s'articulent autour des besoins pratiques et théoriques liés à la création de systèmes cryptographiques utilisables. La prise en compte de contraintes pratiques en plus des contraintes théoriques habituelles permet la conception de systèmes à la fois sécurisés et utilisables, car un système sécurisé peu pratique peut se voir complètement ignoré par ses utilisateurs en faveur de solutions peu ou pas sécurisées. Au terme d'un travail d'observation lié essentiellement aux SGED, nous avons conçu des protocoles essentiels servant de base au déploiement de solutions spécifiques afin de protéger l'ensemble des éléments sensibles des systèmes considérés. Enfin, toutes les propositions d'outils de sécurité sont basées sur des concepts cryptographiques et sur une gestion intelligente des clés associées. L'utilisation d'outils cryptographiques à tous les niveaux permet d'apporter une sécurité fiable en éliminant le risque qu'une simple erreur de manipulation, malveillante ou non, puisse avoir un effet négatif sur la protection du système. La gestion des clés est un élément essentiel ; l'ensemble des moyens de sécurités mis en œuvre s'appuyant sur des procédés cryptographiques, la disponibilité des clés associées est primordiale. C'est pour cela que des mécanismes de création, transfert et protections de trousseaux de clés sécurisés ont été conçus et proposés, à la fois de manière générique et plus spécifique à l'utilisation avec les SGED afin d'assurer leur intégration avec l'interface utilisateur du système et permettre une manipulation transparente non rédhibitoire pour l'utilisateur.

Les résultats des différents travaux produits dans ce document permettent d'atteindre le résultat souhaité : la sécurisation efficace par moyens cryptographiques des composants d'un SGED. Il reste en périphérie de ces propositions des travaux ultérieures à accomplir afin de renforcer l'utilisation des solutions proposées. En particulier dans le cas des SGED le problème de l'utilisation des documents en dehors du système sécurisé. D'autres évolutions liés aux développements d'outils cryptographiques pour la conception de clients légers afin d'y intégrer les solutions proposées.

BIBLIOGRAPHIE

[1] D. Eastlake 3rd and T. Hansen. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF).* Number 6234 in Request for Comments. IETF, May 2011. Published : RFC 6234 (Informational).

[2] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFCs 5247, 7057.

[3] Martin R Albrecht, Kenneth G Paterson, and G Watson. Plaintext recovery attacks against SSH. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 16–26. IEEE, 2009.

[4] ANSSI. Référentiel général de sécurité v2.0. `http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf`.

[5] K Atighehchi, A Bonnecaze, and G Risterucci. New models for efficient authenticated dictionaries. *Computers & Security*, 2015.

[6] R. Barnes, M. Thomson, A. Pironti, and A. Langley. Deprecating Secure Sockets Layer Version 3.0. RFC 7568 (Proposed Standard), June 2015.

[7] Steven Michael Bellovin and William R Cheswick. Privacy-enhanced searches using encrypted bloom filters. 2007.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. *Submission to NIST (Round 2)*, 2009.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.

[10] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption : improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM, 2006.

[11] CVE-1999-1085 SSH insertion attack, March 2002.

[12] Valgrind™ Developers. Valgrind home. `http://valgrind.org`. Accessed : 2015-10-01.

[13] Sunjay Dhama. A practical approach to distributed wpa/wpa2 cracking with cuda. 2014.

[14] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568.

[15] Zakir Durumeric, James Kasten, David Adrian, J Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, et al. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[16] Dennis Branstad Elaine Barker, Miles Smid. A profile for u.s. federal cryptographic key management systems. Technical Report SP 800-152, NIST, October 2015.

[17] Adriana Enache, Kévin Atighehchi, Traian Muntean, and Gabriel Risterucci. An efficient parallel algorithm for skein hash functions. *IACR Cryptology ePrint Archive*, 2010 :432, 2010.

[18] Todd Fast, Timothy Wall, Liang Chen, and JNA contributors. Java native access. `https://github.com/java-native-access/jna`. Accessed : 2015-10-07.

[19] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family. *Submission to NIST (round 3)*, 7(7.5) :3, 2010.

[20] OpenSSL Software Foundation. Openssl website. `https://openssl.org/`. Accessed : 2015-09-29.

[21] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*, pages 31–45. Springer, 2004.

[22] Mourad Gouicem. Comparison of seven sha-3 candidates software implementations on smart cards. *IACR Cryptology ePrint Archive*, 2010 :531, 2010.

[23] Torbjörn Granlund and the GMP development team. *GNU MP : The GNU Multiple Precision Arithmetic Library*, 5.1.3 edition, 2013. `http://gmplib.org/`.

[24] Transport Layer Security Working Group and others. The SSL Protocol, Version 3.0. *Netscape Communications*, 1996.

[25] Shay Gueron. Advanced encryption standard (aes) instructions set. *Intel, http ://softwarecommunity. intel. com/articles/eng/3788. htm, accessed*, 25, 2008.

[26] Shay Gueron and Michael E Kounavis. Intel® carry-less multiplication instruction and its usage for computing the gcm mode. *White Paper*, 2010.

[27] Siddharth Gujrathi. Heartbleed bug : An Openssl heartbeat vulnerability. *International Journal of Computer Science and Engine ter Science and Engineering*, 2(5) :61–64, 2014.

[28] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289 (INTERNET STANDARD), February 1998.

[29] Neil Haller. The s/key one-time password system. Technical report, 1995.

[30] D. Harkins and G. Zorn. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. RFC 5931 (Informational), August 2010.

[31] H. Haverinen and J. Salowey. Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM). RFC 4186 (Informational), January 2006.

[32] ISO. Programming languages – C. ISO ISO/IEC 9899 :1999, International Organization for Standardization, Geneva, Switzerland, 1999.

[33] Hamish Ivey-Law and Robert Rolland. Constructing a database of cryptographically strong elliptic curves. *Proceeding of SAR-SSI*, 2010, 2010.

[34] Anil K Jain, Patrick Flynn, and Arun A Ross. *Handbook of biometrics*. Springer Science & Business Media, 2007.

[35] B. Kaliski. PKCS #5 : Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[36] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010.

[37] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11) :770–772, 1981.

[38] Kashif Latif, Muhammad Tariq, Arshad Aziz, and Athar Mahboob. Efficient software implementation of secure hash algorithm (sha-3) candidate-skein. *International Journal of Academic Research*, 3(6), 2011.

[39] Men Long. Implementing skein hash function on xilinx virtex-5 fpga platform. 2009.

[40] Konstantinos Markantonakis et al. *Smart cards, tokens, security and applications*. Springer Science & Business Media, 2007.

[41] D. McGrew, K. Igoe, and M. Salter. *Fundamental Elliptic Curve Cryptography Algorithms*. Number 6090 in Request for Comments. IETF, February 2011. Published : RFC 6090 (Informational).

[42] D. McGrew and J. Viega. The Galois/Counter mode of operation (GCM). *Submission to NIST.*, 2004.

[43] Microsoft Corporation. Microsoft Security Advisory 2718704, June 2012.

[44] Morris Dworkin. Recommendation for block Cipher Modes of Operation : Galois/Counter Mode (GCM) and GMAC. Technical Report SP 800-38D, NIST, November 2007.

[45] Morris Dworkin. SHA-3 standard : Permutation-based hash and extendable-output functions. Technical Report FIPS PUB 202, NIST, August 2014.

[46] Trustworthy Internet Movement. SSL Pulse. `https://www.trustworthyinternet.org/ssl-pulse/`. Accessed : 2015-09-11.

[47] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. HOTP : An HMAC-Based One-Time Password Algorithm. RFC 4226 (Informational), December 2005.

[48] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP : Time-Based One-Time Password Algorithm. RFC 6238 (Informational), May 2011.

[49] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE Bites : Exploiting The SSL 3.0 Fallback*. 2014.

[50] B Clifford Neuman and Theodore Ts' O. Kerberos : An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9) :33–38, 1994.

[51] Legion of the Bouncy Castle Inc. bouncycastle.org. `https://www.bouncycastle.org`. Accessed : 2015-09-30.

[52] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015.

[53] Longyi Qi. Oclint website. `http://oclint.org`. Accessed : 2015-10-01.

[54] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004. Obsoleted by RFC 5751.

[55] E. Rescorla. *Diffie-Hellman Key Agreement Method*. Number 2631 in Request for Comments. IETF, June 1999. Published : RFC 2631 (Proposed Standard).

[56] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFCs 5785, 7230.

[57] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010.

[58] Gabriel Risterucci, Traian Muntean, and Leon Mugwaneza. A new secure virtual connector approach for communication within large distributed systems. In *Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on*, pages 185–193. IEEE, 2015.

[59] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard), June 2013.

[60] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on SSL a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. *Internet : https ://www. isecpartners. com/media/106031/ssl_ attacks_ survey. pdf [June, 2014]*, 2013.

[61] Justin Schuh. Saying goodbye to our old friend npapi. `https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html`. Accessed : 2015-10-13.

[62] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457 (Informational), February 2015.

[63] Wang Shiuh-Jeng and Chang Jin-Fu. Smart card based secure password authentication scheme. *Computers & Security*, 15(3) :231–237, 1996.

[64] D. Simon, B. Aboba, and R. Hurst. The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard), March 2008.

[65] Benjamin Smedberg. Npapi plugins in firefox. `https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/`. Accessed : 2015-10-13.

[66] Roberto Tyley. Spongy castle by rtyley. `https://rtyley.github.io/spongycastle`. Accessed : 2015-09-30.

[67] Julien Vehent. SSL/TLS analysis of the internet's top 1,000,000 websites. `https://jve.linuxwall.info/blog/index.php?post/TLS_Survey`. Accessed : 2015-09-11.

[68] Jesse Walker, Farhana Sheikh, Sanu K Mathew, and Ram Krishnamurthy. A skein-512 hardware implementation. *Aug-2010.[Online]. Available : http ://csrc. nist. gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/WALKER_ skein-intel-hwd. pdf*, 2010.

[69] David M Webster and Marcin Lukowiak. Versatile fpga architecture for skein hashing algorithm. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 268–273. IEEE, 2011.

[70] David A. Wheeler. How to prevent the next heartbleed. `http://http://www.dwheeler.com/essays/heartbleed.html`, 2015. Accessed : 2015-10-01.

[71] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), January 2006.

[72] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254 (Proposed Standard), January 2006.

[73] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[74] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Updated by RFC 6668.

Annexes

A RAPPELS DES OUTILS CRYPTOGRAPHIQUES COURANTS

Tout au long de ce document, certaines notions de cryptographie sont utilisées ; cette annexe sert de résumé rapide. Les outils cryptographiques permettent d'apporter un certain nombre de propriétés de sécurité en fonction de leur type et de leur utilisation. Les propriétés principales sont la confidentialité, l'intégrité et l'authenticité. La confidentialité empêche la divulgation d'informations à des tiers non autorisés. L'intégrité assure que les données ne sont pas modifiées de façon non autorisée. L'authenticité apporte des garanties relatives à l'émetteur d'un message et à l'intégrité.

Algorithmes cryptographiques

Les algorithmes de "bas niveau" se classent en trois grandes catégories : les chiffrements symétriques, les chiffrements asymétriques et les fonctions de hachage.

Les fonctions de chiffrement symétrique incluent des algorithmes comme AES [1]. On utilise principalement des algorithmes de chiffrement par blocs ; ces derniers traitent des blocs de données de taille fixés. L'application de ces algorithmes de chiffrement se fait selon différents modes d'utilisations, comme CTR [2], CBC [3] ou GCM [4]. Ces modes d'utilisations permettent l'application d'un algorithme de chiffrement par blocs sur un grand volume de données. Les deux premiers (CTR et CBC) se chargent uniquement du chiffrement, alors que le mode GCM apporte en plus un mécanisme d'authentification. Il existe aussi des algorithmes de chiffrement à flot ; ces derniers permettent de traiter des données de taille arbitraire,

1. Advanced Encryption Standard, algorithme de chiffrement par bloc le plus utilisé de nos jours

2. Compteur, cet acronyme est utilisé pour identifier le mode compteur, un mode de chiffrement par blocs

3. Cipher Block Chaining, enchaînement de blocs chiffrés : mode d'utilisation consistant à faire intervenir pour chaque bloc chiffré l'état du bloc précédent, afin de construire une chaîne

4. Galois/Counter-Mode, mode de chiffrement par blocs intégrant un mécanisme d'authentification des données

octet par octet. Ces derniers sont généralement moins efficaces que les chiffrements par blocs. Les algorithmes de chiffrement symétriques sont adaptés au traitement rapide de grands volumes de données et sont protégés par des clés secrètes.

Les fonctions de chiffrement asymétriques regroupent des algorithmes comme RSA [5], DSA [6] et ECDSA [7]. Ils ont deux rôles : la production de signatures numériques et des opérations de chiffrement d'informations. La différence avec les algorithmes de chiffrements symétriques est que les rôles ne sont pas interchangeables dans le cas de la cryptographie asymétrique, alors que la cryptographie symétrique ne fait pas de différence entre possibilité de chiffrement et de déchiffrement. Les signatures numériques permettent de garantir qu'une donnée particulière est connue du signataire et est produite à partir d'une clé privée. Une telle signature peut ensuite être vérifiée par n'importe qui à partir de la clé publique associée. Cette paire de clés privée/publique est l'élément sur lequel repose la sécurité du système. Les possibilités de chiffrement asymétrique utilisent la clé publique pour le chiffrement et la clé privée pour le déchiffrement, permettant l'envoi d'une donnée confidentielle à une entité avec la seule connaissance de sa clé publique. Les algorithmes asymétriques sont généralement basés sur des problèmes mathématiques complexes et ont des performances moindres par rapport aux algorithmes symétriques. Ils ne sont donc pas adaptés au traitement de grands volumes de données.

Les fonctions de hachage enfin permettent de produire des hachés (aussi appelés image ou condensât) de taille fixe à partir d'une donnée en entrée de taille quelconque. Ces fonctions en cryptographie sont des fonctions à sens unique : il est facile d'obtenir un haché à partir d'une donnée, mais extrêmement difficile d'obtenir une donnée correspondant à un haché donné. Elles sont déterministes, ce qui signifie que plusieurs applications consécutives d'une même fonction de hachage à une même donnée produit le même haché. Leurs utilisations sont multiples ; parmi les usages courants nous pouvons citer l'utilisation en conjonction avec des signatures numériques et la production de clés secrètes. Les signatures numériques sont généralement appliquées au haché d'une donnée plutôt qu'à la donnée elle-même ; les propriétés des fonctions de hachage cryptographiques font que l'application de la signature au haché est équivalente à la signature de la donnée initiale. Les clés secrètes ayant une taille fixée, l'utilisation de fonctions de hachage permet de partir d'un matériel source de taille quelconque (comme un mot de passe) de produire un élément de taille fixée, compatible avec une clé secrète.

5. Rivest Shamir Adleman, algorithme de chiffrement a clé publique parmis les plus utilisés au monde

6. Digital Signature Algorithm, standard de signature numériques

7. Elliptic-Curve Digital Signature Algorithm, signatures numériques basées sur courbes elliptiques

Clés

Les algorithmes de chiffrements et de signatures numériques s'appuient sur deux éléments principaux pour assurer leur sécurité. D'une part, la complexité calculatoire nécessaire pour inverser les opérations en l'absence de connaissance des éléments secrets et d'autre part le secret des clés. À la base on distingue trois types de clés : les clés secrètes et les paires de clés privées/publiques.

Les clés secrètes sont utilisées en cryptographie symétrique. Elles sont de taille fixe et dépendent des algorithmes utilisés. La forme de ces clés secrètes n'est pas imposée ; il s'agit uniquement d'un bloc de données. Ces clés sont nécessaires à la fois pour les opérations de chiffrement et de déchiffrement. Cela signifie qu'elles doivent être communiquées aux différentes entités devant pouvoir réaliser ces opérations.

Les paires de clés privées/publiques sont utilisées pour les opérations asymétriques. Leur utilisation peut être variée ; de manière générale, la clé publique n'est jamais une donnée sensible et doit être connue de tous pour être utile. La clé privée associée ne doit jamais être diffusée à grande échelle. Idéalement, une clé privée n'est connue que d'un nombre très limité d'entité, voir une seule. Cependant, dans certains cas d'usage, cette clé privée peut être diffusée de façon limitée. C'est ici le rôle asymétrique qui est essentiel, la connaissance de la clé privée servant à prouver l'appartenance à un groupe.

Autres outils

Il existe d'autres outils en cryptographie qu'il est nécessaire d'utiliser pour faire fonctionner des systèmes sécurisés. Parmi ces outils, l'un des plus importantes est le PRNG [8]. La génération d'aléa est à la base de la sécurité des systèmes cryptographiques ; c'est à partir de sources aléatoires que sont générés les différentes clés. Ces sources aléatoires doivent donc présenter un vrai caractère imprévisible afin d'assurer que les clés générées ne peuvent pas être devinées à partir d'une corruption du PRNG.

8. Pseudo-Random Number Generator, générateur pseudo-aléatoire

B CONVENTIONS DE CODAGE SÉCURISÉES

Les conventions de codage appliquées au développement de la bibliothèque logicielle Arcana-Cryptobox couvrent plusieurs aspects : cosmétiques et nommage (pour la cohérence des développements), organisation du code, restriction de types de données,... Nous allons présenter ici les éléments de ces conventions de codage qui sont pertinents au développement d'outils dans un contexte d'application de sécurité ; les éléments cosmétiques sont laissés de côté. Les objectifs principaux ici sont de limiter les risques de confusion souvent source d'erreur, d'améliorer la remontée d'erreurs et de faciliter la portabilité de l'API [1] vers d'autres environnements de développement.

Limitation des types de données : Les fonctions exposées par l'API publique sont limitées à un ensemble réduit de types de données. Cette restriction apporte plusieurs avantages : elle réduit la complexité d'appel et permet une meilleure compatibilité avec des interfaces dans d'autres langages. Les types autorisés sont les suivants :

- bool : booléen
- int : pour les données numériques de petite taille comme le nombre de participants dans un échange
- size_t : pour indiquer la taille d'un bloc de données
- void : un bloc de données opaque, principalement des données à chiffrer
- char : fonctionnellement identique au type précédent, mais utilisé pour distinguer les chaînes de caractères humainement lisibles des blocs de données
- ctx_t : encore une fois, un pointeur opaque. L'API publique a besoin de conserver un ensemble de données pour fonctionner entre différents appels de fonctions (des contextes). Ce type existe pour différencier ces pointeurs des blocs de données.
- pointeurs de fonctions : limitées à un nombre réduit de cas, l'utilisation

1. Application Programming Interface, Interface de programmation exposant les fonctions d'un système ou d'une bibliothèque logicielle

de pointeurs de fonctions est nécessaire pour certains usages ; les types de fonctions utilisées présentent les mêmes limitations de types.

La distinction entre les quatre types de pointeurs permet de clarifier la lecture du code et améliore l'automatisation du système de portabilité vers d'autres langages.

Nommage des paramètres : Les paramètres des fonctions de l'API publique sont nommées afin de permettre une forme d'auto-documentation. Au-delà de la lisibilité, une convention est mise en place pour permettre une meilleure traduction automatique de l'API vers d'autres langages de programmation. Les paramètres de type void représentant un bloc de données sont toujours suivis d'un second paramètre de type size_t suffixé par "_size". En plus de permettre une portabilité plus aisée, cette convention réduit les risques induits par l'utilisation de taille de blocs implicites.

Découpage du code : L'essentiel des fonctions publiques s'articule autour d'un système de "contextes". Un contexte contient toutes les informations nécessaires pour permettre l'enchaînement d'appels de fonctions afin de réaliser une action. Un exemple est l'application d'une application de chiffrement ; elle nécessite plusieurs appels consécutifs pour initialiser, utiliser et finaliser l'opération de chiffrement. Ces différents appels sont capables de fonctionner ensemble, car une donnée commune, le contexte, est passé à chaque étape.

L'utilisation de contexte suit le schéma suivant : création, initialisation, utilisation et finalisation. La création permet d'allouer la mémoire nécessaire à l'opération demandée. L'initialisation permet de définir les paramètres de l'opération. L'utilisation est l'application de l'algorithme demandé aux données et éventuellement la récupération des résultats. La finalisation consiste à libérer la mémoire allouée lors de la création.

Ce schéma de fonctionnement est commun à l'ensemble des modules publics. Il a pour but, comme le reste des conventions décrits ici, de simplifier les appels et de permettre une traduction automatique vers d'autres environnements de développement. Un exemple direct est la création d'une interface pour un langage objet ; dans ce cas les étapes de création/finalisation peuvent être inclue au concept d'objet du langage.

Gestion d'erreur : Le dernier point des conventions de codage qui concerne l'aspect de sécurité est la gestion d'erreur. Il y a deux modèles appliqués au fil du code : celui des fonctions publiques et des fonctions privées. Les fonctions privées ne peuvent être appelées que depuis le code de la bibliothèque ; elles ne sont

pas exposées par l'interface publique. Ces fonctions ne doivent pas avoir de comportement imprévus ; tous leurs paramètres ont une plage de valeurs acceptables et ces limites sont contrôlées par des assertions. Les appels aux fonctions internes n'étant possible que depuis les autres fonctions de la bibliothèque, ils sont supposés contrôlés en amont.

Le modèle appliqué aux fonctions publiques est différent. Toutes les fonctions de l'API publiques renvoient un entier représentant un code d'erreur. Ces fonctions ont elles aussi des restrictions sur l'amplitude des paramètres. Les vérifications sont cette fois faites de façon plus libre ; en cas d'erreur de paramètre, la fonction retourne un code spécifique permettant d'identifier le paramètre erroné. L'objectif ici est qu'une fois les paramètres d'appels publics validés, l'exécution du code soit totalement contrôlée. La bibliothèque ne doit pas arrêter brutalement l'application appelante.

En dehors des paramètres erronés, les autres cas d'erreurs sont peu nombreux et sont tous représentés par une valeur générique, renvoyée par la fonction en erreur. L'application appelante doit vérifier le code de retour de chaque appel de fonction et interrompre le traitement en cas de problème. La logique création-finalisation décrite précédemment reste applicable : dès qu'un contexte d'utilisation provoque une erreur, ce contexte doit être finalisé afin de libérer la mémoire allouée.