Processus sécurisés de dématérialisation de cartes sans contact

Mohamed Amine Bouazzouni

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Réseaux, Télécommunications, Systèmes et Architecture

Présentée et soutenue par :

M. MOHAMED AMINE BOUAZZOUNI

le mercredi 8 novembre 2017

Titre :

Processus sécurisés de dématérialisation de cartes sans contact

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. FABRICE PEYRARD

M. EMMANUEL CONCHON

Rapporteurs :

M. BENJAMIN NGUYEN, INSA Centre Val de Loire

Mme SAMIA SAAD-BOUZEFRANE, CNAM PARIS

Membre(s) du jury :

M. MARIE PIERRE GLEIZES, UNIVERSITE TOULOUSE 3, Président

M. PIERRE PARADINAS, CNAM PARIS, Membre

Remerciements

Je tiens d'abord à adresser mes plus sincères remerciements et ma gratitude éternelle à mes parents sans qui rien de tout cela n'aurait été possible. Durant toute ma vie et mon cursus scolaire, ils ont été présents pour m'encourager et faire en sorte que je dispose de toutes les chances pour réussir dans ma vie. Je tiens ensuite à adresser mes remerciements à la femme qui partage ma vie. Elle a su, durant cette dernière année faire preuve de courage pour pouvoir me supporter lors des moments difficiles. Elle a su trouver les mots pour m'encourager et me remonter le morale quand c'était nécessaire. Je remercie aussi tous les membres de ma famille qui ont été derrière moi avec leurs encouragements.

Je tiens aussi à remercier mon directeur de thèse Fabrice Peyrard et mon co-directeur de thèse Emmanuel Conchon qui m'ont accompagné depuis mon stage de M2R. Je tiens à leurs adresser ma reconnaissance du temps qu'ils m'ont accordé pour m'aider dans mes travaux et de la liberté qu'ils m'ont laissé dans la conduite de la thèse. Je les remercie sincèrement de leur investissement personnel et professionnel au service de la réussite de cette thèse.

Je remercie Marie-Pierre Gleizes de m'avoir fait l'honneur de présider le jury. J'adresse mes sincères remerciements à Samia Saad-Bouzefrane et Benjamin Nguyen pour leur lecture minutieuse de mon manuscrit et pour avoir accepté de rapporter cette thèse. Je remercie également Pierre Paradinas et Abdelmalek Benzekri de m'avoir fait l'honneur de participer au jury. Je suis reconnaissant à l'ensemble des membres du jury pour les remarques pertinentes et les suggestions intéressantes soulevées par leurs questions qui ont contribué à améliorer la qualité de ce travail.

Je tiens enfin à exprimer ma profonde gratitude à mes amis, qui ont été présents à mes côtés durant cette thèse et qui, par leur écoute et leurs conseils m'ont été d'une grande aide dans cette épreuve. Cette liste est très longue et ne se veut pas être exhaustive : Ahmed A., Younes, Bilal, Aziz, Abdelghafour, Ahmad, Bilel, Farouk, Mohammed, Samer, Ismail, Mahdi W., Abderrahim et d'autres que j'ai sûrement oublié. Je tiens aussi à remercier mes collègues doctorants de l'équipe IRT : Cédric, Éric, Jean-Gabriel, Oana, Émilie, Dorin et Yohan.

Mohamed

Résumé

Au fil des années, la technologie sans contact NFC s'est imposée dans notre quotidien au travers des différents services proposés. Les cas d'utilisation sont nombreux allant des cartes de fidélité, des cartes de transport, des cartes de paiement sans contact jusqu'aux cartes de contrôle d'accès. Cependant, les premières générations des cartes NFC ont une sécurité minimale reposant sur l'hypothèse de leur non-clonabilité. De multiples vulnérabilités ont été découvertes et leur exploitation a permis des copies frauduleuses. Afin de remédier à ces vulnérabilités, une nouvelle génération de cartes à la sécurité augmentée a vu le jour. Ces cartes permettent une authentification avec un lecteur basée sur des algorithmes de chiffrements symétriques tels qu'AES, DES, et 3DES. Elles sont plus robustes que la première génération mais ont subi des également une attaque en reverse-engineering.

Pour garantir et améliorer le niveau de sécurité du système de contrôle d'accès, nous proposons dans le cadre de l'opération neOCampus, la dématérialisation sécurisée de la carte sans contact sur un smartphone muni de la technologie NFC. Cette dématérialisation nous permet d'exploiter la puissance de calcul et la capacité de stockage du smartphone afin de déployer des algorithmes d'authentification plus robustes. Cependant, l'OS du smartphone ne peut être considéré comme un environnement de confiance. Afin de répondre à la problématique du stockage et du traitement sécurisés sur un smartphone, plusieurs solutions ont été proposées : les Secure Elements (SE), les Trusted Platform Module (TPM), les Trusted Execution Environment (TEE) et la virtualisation.

Afin de stocker et de traiter de manière sécurisée les données d'authentification, le TEE apparait comme la solution idéale avec le meilleur compromis sécurité/performances. Cependant, de nombreux smartphones n'embarquent pas encore de TEE. Pour remédier à cette contrainte, nous proposons une architecture basée sur l'utilisation de TEEs déportés sur le Cloud. Le smartphone peut le contacter via une liaison Wi-Fi ou 4G. Pour se faire, un protocole d'authentification basé sur IBAKE est proposé.

En plus de ce scénario nominal, deux autres scenarii complémentaires ont été proposés permettant d'accompagner le développement et la démocratisation des TEE non seulement dans le monde des smartphones mais aussi sur des dispositifs peu onéreux comme le Raspberry Pi 3. Ces architectures déploient le même algorithme d'authentification que le scénario nominal. Nous proposons aussi une architecture hors ligne

permettant à un utilisateur de s'authentifier à l'aide d'un jeton de connexion en cas d'absence de réseaux sans fil. Cette solution permet de relâcher la contrainte sur la connectivité du smartphone à son Cloud.

Nous procédons à une évaluation de l'architecture de dématérialisation et de l'algorithme d'authentification en terme de performances et de sécurité. Les opérations cryptographiques du protocole d'authentification sont les plus coûteuses. Nous avons alors procédé à leur évaluation en nous intéressant en particulier aux opérations de chiffrement IBE et à la génération de challenges ECC. Nos implémentations ont été évaluées pour l'infrastructure Cloud et l'environnement mobile.

Nous avons ensuite procédé à une validation du protocole d'authentification sur les trois architectures sélectionnées à l'aide de l'outil Scyther. Nous avons montré, que pour les trois scenarii, la clé de session négociée via le protocole d'authentification restait secrète durant tout le protocole. Cette caractéristique nous garantit que les données d'authentification chiffrées avec cette clé resteront secrètes et que la phase d'identification de la personne est protégée tout en préservant l'ergonomie du système existant.

Abstract

Over the years, the Near Field Communication technology has emerged in our daily lives through a variety of services. There are several use cases for contactless cards : loyalty cards, metro and bus cards, payment cards and access control cards. However, the first version of these cards has a low security level that is based on the assumption that the cards can not be cloned. To address this issue, a new version of NFC cards has been developed. It allows an authentication with the NFC reader through symmetric encryption algorithms such as AES, DES or 3DES. These cards are more robust that the previous ones. However, these cards have also undergone a reverse-engineering attack.

We propose, in the context of the neOCampus project, to replace the contactless cards with a smartphone equipped with the NFC capabilities. This process, called dematerialization, allows us to take advantage of the computational power and the storage capabilities of the smartphone to deploy more complex and robust authentication algorithms. However, the OS of the smartphone can not be considered as a trusted environment for the storage and the processing of sensitive data. To address these issues, several solutions were proposed : Secure Elements (SE), Trusted Platform Module (TPM), Trusted Execution Environment (TEE) and Virtualization.

In order to store and process securely authentication data, the TEE seems to be the best trade-off between security and performances. Nevertheless, many smartphones do not embeed TEE and it is necessary to negotiate agreements with the TEE manufacturers in order to deploy a secure application on it. In order to figure out these issues, we propose to set up an architecture with a TEE in the Cloud. The smartphone has a secure Cloud that can be reached through a Wi-Fi or 4G connection. The reader has also its own secure Cloud reachable with an Ethernet link. An authentication protocol based on IBAKE is also proposed.

In addition to this scenario, two other scenarios were proposed to follow the development and democratization of the TEE on the smartphones and on some inexpensive devices such as Raspberry Pi 3. These alternative architectures deploy the same authentication protocol as the main scenario. We propose an offline architecture allowing a user to authenticate using a connection token. This solution relaxes the connectivity constraint between the smartphone and its secure Cloud.

We perform an evaluation of our architecture and of the authentication algorithm in terms of performances and security. The cryptographical operations of the au-

thentication protocol are the most consuming operations in term of performance. We have chosen to target these operations especially the encryption with the IBE and the ECC challenges generation. Our implementations have been evaluated for a Cloud infrastructure and a mobile-like environment.

We also perform a formal verification of the authentication protocol through the three considered architectures with Scyther. We showed that, for the three scenarios, that the session key negotiated through the authentication protocol remains secret during the overall execution of the protocol. These characteristic guarantee that the authentication data encrypted with this key will remain secret and that this step of the algorithm will be secure while preserving the ergonomy of the existing system.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1.1 Contexte et défis

Au fil des années, la technologie de communication sans contact à faible portée NFC s'est imposée dans notre vie quotidienne à travers les différents services déployés. Des cartes de fidélité de magasins aux cartes de transport en commun en passant par des cartes d'accès aux bâtiments, le spectre d'utilisation de la technologie NFC est large. Elle possède l'avantage de ne nécessiter aucune configuration préalable offrant une rapidité et une facilité de déploiement très appréciées. Ces dernières années, le NFC s'est même invité sur les smartphones ouvrant de plus en plus de possibilités pour les développeurs. Cependant, et plus particulièrement sur les premières versions des cartes NFC, la sécurité de ces cartes ne reposait que sur l'hypothèse qu'elles étaient non-clonables grâce notamment à un verrouillage des données empêchant un attaquant de les lire et de les réécrire sur une autre carte. Néanmoins, plusieurs vulnérabilités [1] [2] ont été détectées sur ces cartes et très vite, des attaques exploitant ces vulnérabilités ont vu le jour permettant la duplication sans limite de ces cartes. De plus, certains dispositifs *low cost* embarquant des primitives des attaques possibles sont disponibles à la vente [3]. Au moment où ces cartes sont de plus en plus utilisées dans les systèmes de contrôle d'accès aux bâtiments et ressources sensibles mais aussi pour le paiement avec l'introduction par de nombreuses banques de cartes de paiement sans contact NFC ou via des applications mobiles propriétaires telles qu'ApplePay et OrangeCash, le facteur sécurité est devenu primordial avant le déploiement de toute solution basée sur NFC.

Afin de pouvoir augmenter la sécurité des systèmes de contrôle d'accès et de paiement notamment, de nouvelles cartes dotées de capacités cryptographiques accrues ont été développées. Ces cartes, appelées DESFIRE, déploient des algorithmes d'authentification avec les lecteurs NFC basés sur des algorithmes de chiffrement symétriques standardisés tels que AES et 3DES. Ces cartes offrent un niveau de sécurité accru comparé aux cartes NFC classiques. Cependant, de récentes études [1] pointent la possibilité d'une attaque *Side Channel* permettant de récupérer la clé 3DES utilisées lors du processus d'authentification. Selon les auteurs, cette attaque nécessite peu de ressources et des dispositifs implémentant cette attaque sont disponibles à la vente pour un millier d'euros environ.

L'utilisation de ces technologies étant exclue pour le déploiement de systèmes de contrôle d'accès fiables, nous proposons dans le cadre de l'opération neOCampus [4] de concevoir une nouvelle solution de contrôle d'accès basée sur la dématérialisation des cartes traditionnelles vulnérables et de les remplacer par un smartphone embarquant une application sécurisée de contrôle d'accès.

L'opération de recherche neOCampus, initiée en juin 2013 par l'université Toulouse III - Paul Sabatier, réunit les enseignants-chercheurs et chercheurs de 10 laboratoires : CESBIO, CIRIMAT, ECOLAB, IRIT, LA, LAAS, LAPLACE, LCC, LERASS, LMDC. Ces laboratoires ont pour objectif de croiser leurs compétences pour améliorer le confort au quotidien de la communauté universitaire tout en diminuant l'empreinte écologique des bâtiments et les coûts de fonctionnement (fluide, eau, électricité...).

NeOCampus a pour objectif de concevoir les produits et services associés aux systèmes cyber-physiques ambiants. La plate-forme associée à neOCampus consiste en de nombreux dispositifs logiciels et matériels interconnectés pour le campus numérique de demain, durable et intelligent, alliant matériels pédagogiques innovants, capteurs, systèmes de communication, de stockage, de localisation, de simulation et des matériaux innovants, au sein de bâtiments universitaires du campus, pour améliorer la qualité de vie des usagers et réduire les consommations de fluides.

Un campus étant un système complexe s'apparentant à une petite ville, il possède de multiples dynamiques qui induisent l'impossibilité de prévoir toutes les conséquences d'un changement, même minime. Ainsi, de gros équipements lourds, statiques et onéreux sont antinomiques avec ces considérations. La démarche de neOCampus se veut évolutive et adaptative car un tel système complexe doit être instrumenté et régulé par de multiples dispositifs disséminés dans l'espace et le temps.

Au sein de l'université de Toulouse chaque étudiant, enseignant, personnel administratif se voit délivrer une carte multi-services MUT qui sert de carte d'étudiant et de carte professionnelle. Il s'agit d'une carte multi-services utilisée pour le contrôle d'accès aux bâtiments sécurisés et aux laboratoires de recherche mais aussi pour le paiement (anciennement Monéo [5] et Izly [6] actuellement).

1.2 Les objectifs visés

La dématérialisation d'une carte sans contact définit l'opération visant à embarquer la carte NFC sur un smartphone dôté de cette même technologie tout en garantissant l'accès au mêmes services avec un niveau de sécurité accru. En effet, la majorité des smartphones modernes sont

dotés de NFC et leur capacité de stockage tout comme leur puissance de calcul en font les candidats idéaux pour le remplacement des cartes traditionnelles. Ce processus de dématérialisation doit répondre aux exigences suivantes :

— Remplacer la carte physique et éliminer le support plastique (ce qui peut être une source d'économie).
— Offrir les mêmes services à l'usager de manière intuitive et simple.
— Garantir un meilleur niveau de sécurité du contrôle d'accès.
— Respecter l'esprit de la technologie NFC en terme de rapidité et facilité de déploiement.

La Figure 1.1 résume le processus de dématérialisation dans le contexte du contrôle d'accès sécurisé.

FIGURE 1.1 – Dématérialisation de la carte MUT

1.3 Les problématiques de la dématérialisation

Le processus de dématérialisation d'une carte physique sur un smartphone se heurte au fait que le système d'exploitation du terminal ne peut être considéré comme un environnement de confiance. En effet, plusieurs études [7] [8] ont permis de démontrer qu'un nombre important de malwares sévissent sur Android et s'attaquent plus particulièrement aux données sensibles stockées sur le smartphone parmi lesquelles les données bancaires et les données d'authentification à des services protégés. Afin de s'assurer de la confidentialité de ces données, une des solutions proposée est le chiffrement. Cette solution est cependant vulnérable à deux types d'attaques : attaque sur le stockage de la clé de chiffrement et attaque sur les entrées/sorties au cas où la clé est fournie par l'utilisateur via le clavier virtuel. De plus, une application avec des privilèges élevés (root sur le smartphone) pourra faire de l'introspection mémoire et récupérer la clé pendant l'exécution de l'opération de chiffrement/déchiffrement.

Afin de répondre à la problématique de stockage et de traitement sécurisé sur smartphone, plusieurs solutions ont été développées. Les Secure Elements (SE) représentent la première solution déployée. Un SE est une carte à puce exécutant des applications JavaCard et offrant un niveau de sécurité comparable aux cartes bancaires que nous utilisons. Il existe trois types de

SE pour les smartphones : le SE intégré, le SE sur la carte SIM et le SE sur la carte mémoire type MicroSD. L'avantage du SE intégré réside dans le fait qu'il est directement câblé à la carte mère et au contrôleur NFC ce qui réduit considérablement la surface d'attaque. Le SE sur la carte SIM présente l'avantage de régler le problème de compatibilité matérielle car tous les smartphones ne sont pas dotés de SE intégré. Le SE sur la carte MicroSD a pour avantage la portabilité du SE et la possibilité de l'embarquer sur différents équipements. Cependant, cet avantage devient un inconvénient en cas de vol de cette carte SD. Les trois types de SE partagent un même inconvénient majeur : ils sont souvent propriétaires et nécessitent des accords avec les fabricants et/ou l'opérateur mobile pour le déploiement d'applications JavaCard. Cette solution est donc peu accessible et peu utilisée par les développeurs d'applications mobiles. De plus, la capacité de traitement et de stockage limitée de ces cartes à puces ne permettent pas le déploiement d'algorithme d'authentification assez robuste.

Les Trusted Platform Module (TPM) représentent une autre solution aux problèmes de stockage et de traitement sécurisés sur un smartphone. Dans un premier temps, cette solution n'était disponible que sur les ordinateurs portables afin d'authentifier l'utilisateur de ces terminaux mobiles. Le TPM consiste en une puce dotée de capacités cryptographiques lui permettant d'effectuer des opérations de chiffrement/déchiffrement, signature et hashage sur des données fournies en entrée. Les TPM ne sont pas conçus pour le stockage et le traitement de larges quantités de données ce qui représente un inconvénient majeur à leur intégration dans des solutions de contrôle d'accès sur smartphones.

Afin de disposer d'un environnement de stockage et de traitement sécurisé affichant des performances élevées, les fabricants de processeurs pour les smartphones, notamment ARM, ont conçu les Trusted Execution Environment (TEE). Un TEE est un processeur ou une zone du processeur chargée d'exécuter les opérations sensibles telles que le chiffrement, le déchiffrement et la signature. L'avantage des TEE par rapport aux solutions sus-citées est sa grande capacité de stockage et de traitement. L'environnement du smartphone se retrouve fragmenté en deux parties : un Rich OS, représentant l'OS standard et un Secure OS représentant le système d'exploitation sécurisé. Cependant, il présente l'inconvénient de nécessiter l'accord du fabriquant et/ou fournisseur du Secure OS lors du déploiement de l'application sécurisée. Pour contourner cette contrainte, certains TEE et Secure OS open source voient le jour. Parmi les plus notables, nous pouvons citer OP-TEE et OPEN-TEE.

Dans l'optique de développer des environnements sécurisés non sujets aux contraintes qu'imposent les constructeurs, plusieurs solutions de virtualisation ont vu le jour. La virtualisation consiste à exécuter deux systèmes d'exploitation sur un même terminal en provoquant le minimum d'impact sur les performances par rapport à un smartphone pourvu d'un seul OS. Des solutions de ce type existe déjà dans l'environnement des ordinateurs parmi lesquelles nous pouvons citer les conteneurs, les sandboxes et Docker. Ces solutions permettent une isolation stricte entre le système hôte et le système invité.

Outre le choix de l'architecture matérielle optimale pour la dématérialisation, il est aussi né-

cessaire de mener une réflexion sur le protocole d'authentification à déployer. La quasi majorité de ces protocoles sont basés sur le concept d'infrastructure de clé publique. Cette autorité de confiance génère les clés publiques/privées et émet les certificats garantissant l'authenticité des clés publiques utilisées. L'inconvénient de cette solution réside dans le manque de flexibilité de déploiement surtout dans les petites structures. Afin de remédier à cette contrainte sans réduire le niveau de sécurité, il est possible d'utiliser une solution plus flexible appelée Identity Based Encryption (IBE). Le concept des IBE consiste à utiliser l'identité d'une personne ou d'un périphérique comme clé publique. Un Private Key Generator (PKG) est utilisé sur chaque terminal afin de générer sa propre clé privée d'où la nécessité de sécuriser ce processus. Cette solution présente l'avantage d'éliminer l'utilisation des certificats, néanmoins, la gestion de l'identité est primordiale pour s'assurer de l'unicité des clés générées.

1.4 Contributions

En prenant en compte l'ensemble des contraintes énoncées et la nécessité de déployer des systèmes de contrôle d'accès fiables, nous proposons la mise en place d'une architecture de dématérialisation alliant sécurité et flexibilité. Nous proposons, implémentons, évaluons et vérifions un protocole d'authentification basé sur l'identité permettant une double authentification entre le smartphone de l'utilisateur et le lecteur du système de contrôle d'accès.

L'utilisation d'un Cloud sécurisé peut être une solution permettant de déporter un TEE sur un serveur accessible par les utilisateurs. Cette solution permet de remédier aux problèmes de compatibilité et d'accords avec les constructeurs et/ou opérateurs télécoms. Nous proposons un protocole d'authentification basé sur l'identité qui est déployé entre deux Clouds permettant d'authentifier le lecteur et le smartphone et donc de déterminer si un utilisateur est autorisé ou pas à accéder au bâtiment sécurisé.

Deux autres scenarii dérivés de ce scénario principal sont présentés notamment en considérant, le positionnement du TEE dans l'infrastructure globale. Nous décrivons aussi un mécanisme d'accès hors connexion basé sur l'utilisation l'utilisation de jetons de connexion. Une évaluation de performances et une vérification formelle du protocole d'authentification en terme de sécurité à l'aide de l'outil Scyther [9] sont présentées.

1.5 Organisation du manuscrit

Cette thèse est structurée en 6 chapitres comme suit :

— **Chapitre 1** : ce chapitre introductif permet d'établir le contexte de la thèse et d'énoncer le problème posé. Il permet de présenter les contraintes à respecter dans l'élaboration de la solution et d'énoncer les contributions de ce travail.

— **Chapitre 2** : ce chapitre traite principalement de la technologie de communication à faible portée NFC. Nous commençons par présenter les caractéristiques de cette tech-

nologie ainsi que ces modes de fonctionnement. Nous présentons les différentes cartes sans contact utilisées dans les systèmes de contrôle d'accès actuels en pointant leurs vulnérabilités.

— **Chapitre 3** : ce chapitre est consacré à l'étude des environnements de stockage et d'exécution sécurisés notamment dans l'environnement mobile. Ce chapitre est divisé en deux parties principales : la solution logicielle représentée par la virtualisation et les solutions matérielles telles que les SE, les TEE et les TPM. Une analyse comparative suivant des critères de performances et de sécurité est discutée afin d'identifier les meilleures solutions par rapport au contrôle d'accès.

— **Chapitre 4** : ce chapitre est dédié à la présentation de la solution proposée. Nous entamons cette partie avec la définition de la dématérialisation en identifiant les avantages et les inconvénients de cette solution. Ensuite, nous présentons l'architecture de dématérialisation retenue en présentant en détail tous les éléments qui la composent. Nous présentons également les différents scenarii alternatifs (online, semi-online et tokens d'authentification) considérés et identifions les défis à relever pour une solution viable. Enfin, nous décrivons le protocole d'authentification basé sur IBAKE que nous proposons de déployer au sein de notre architecture.

— **Chapitre 5** : ce chapitre analyse les solutions retenues en terme de performances et de sécurité. En effet, une étude expérimentale sur deux implémentations (matérielle et virtualisée) nous permet de faire une évaluation des performances des différentes solutions et de statuer sur leur viabilité dans un environnement réel. De plus, nous procédons à une vérification formelle du protocole d'authentification en terme de sécurité pour s'assurer de sa robustesse avec l'outil Scyther.

— **Conclusion** : cette partie conclut la thèse. Nous dressons le bilan et la synthèse de nos travaux avant d'ouvrir sur les perspectives.

2 État de l'art Near Field Communication

Introduction

Les cartes sans contact sont le moyen le plus utilisé pour faire du contrôle d'accès aux immeubles sensibles. La technologie NFC est un maillon essentiel de ces systèmes de contrôle d'accès. En effet, elle permet de s'assurer que l'utilisateur se trouve à moins de 10 centimètres de la porte protégée. L'utilisateur possède une carte NFC qu'il présente devant un lecteur qui vérifie si le possesseur de la carte est habilité à accéder au bâtiment. Plusieurs types de cartes NFC peuvent être utilisées dans les systèmes de contrôle d'accès, allant de la MIFARE Classic [10] à la DESFire [11] en passant par les UltralightC [12]. Ces cartes diffèrent entre elles en terme de capacité de stockage et de capacités cryptographiques. Nous présentons dans ce chapitre, la technologie NFC ainsi que ses modes de fonctionnement et les principaux standards la régissant. Nous présentons ensuite les différents types de cartes sans contact en mettant en exergue les différentes vulnérabilités identifiées les rendant peu recommandables pour un usage dans des infrastructures sensibles.

2.1 Historique et description

La technologie NFC a été développée conjointement par Philips et Sony à la fin de l'année 2002 pour les communications sans contact. NFC est un protocole de communication à distance courte [13], *half duplex* permettant une communication simple entre deux terminaux séparés de quelques centimètres utilisant la fréquence radio de *13.56MHz*. La portée maximale théorique se situe entre 4 et 10 centimètres et le débit maximal peut atteindre 424Kbps. Le Tableau 2.1 permet de faire un comparatif entre les principales technologies sans fil.

Les périphériques qui participent à une communication NFC peuvent être classés en deux catégories [14] [15] : un périphérique produisant un champ électromagnétique et gérant la communication et l'échange de données, appelé lecteur et un périphérique utilisant le champ électromagnétique produit pour répondre à ses requêtes, dit tag ou carte.

Le protocole NFC proposent deux modes de communications : mode actif et mode passif. Dans le mode actif, l'initiateur de la communication génère un champ électromagnétique capté par l'antenne du second périphérique. Ce dernier se met automatiquement en mode cible et traite

Tableau 2.1 – Comparaison des différentes technologies de communication sans fil

Paramètres	**Bluetooth 2.1**	**ZigBee (802.15.4)**	**NFC**
Distance	10-100 m	10-100 m	4-10 cm
Débit	0.8-2.1 Mb/s	0.02-0.2 Mb/s	0.02-0.4 Mb/s
Cout de déploiement	faible	faible	faible
Consommation	élevée	moyenne	faible
Spectre	2.4GHz	2.4GHz	13.56MHz
Topologie réseau	Piconets, scatternet	Étoile, Arbre, ...	un à un
Terminaux/réseau	8	2-65000	2
Utilisation	Centré données	Centré données	Centré humain
Personnalisation	Moyenne	Faible	Forte
Flexibilité	Forte	Forte	Forte
Temps d'appareillage	Approx. 6 sec	Approx. 0.5 sec	Approx. 0.1 sec

le signal reçu afin d'en extraire les informations transmises par l'initiateur. Pour répondre, la cible génère le même champ électromagnétique que l'initiateur, seule la direction de la transmission change. La transmission est en 'half duplex', un seul périphérique à la fois peut émettre. Dans le mode passif, seul l'initiateur génère un champ électromagnétique et les périphériques cibles utilisent ce champ pour émettre (répondre). Dès que la cible entre dans la zone de portée de l'initiateur, il génère un champ électromagnétique qui lui permet d'envoyer des données vers la cible. Après la réception des données, le champ électromagnétique est émis de façon non modulée afin de permettre à la cible de l'utiliser pour répondre. C'est ce qui se passe lorsqu'un tag entre dans la zone de portée radio d'un lecteur.

2.2 Modes de fonctionnement de NFC

Il existe trois modes de fonctionnement pour le protocole NFC [16] : le mode Reader/Writer, le mode Peer-to-Peer et le mode Card-Emulation. Nous allons dans ce qui suit, détailler les spécificités de chacun des trois modes.

2.2.1 Mode Reader/Writer

La Figure 2.1 représente le mode de communication Reader/Writer. Dans ce mode de fonctionnement, un smartphone muni de NFC initie la communication sans fil et peut lire et/ou modifier des données présentes sur le tag NFC. Ce mode est très utilisé pour le partage d'URL et de mots de passe de connexion WiFi par exemple. En effet, l'administrateur initialise un tag NFC avec la bonne URL et la met à disposition des utilisateurs qui 'tapent' [1] avec leur téléphone sur le tag ce qui leur permet de récupérer l'URL voulue.

Les deux périphériques peuvent aussi s'échanger des messages NDEF (NFC Data Exchange

1. Taper un tag NFC veut dire approcher le téléphone de ce dernier pour qu'il puisse communiquer avec le tag et récupérer les données présentes dessus

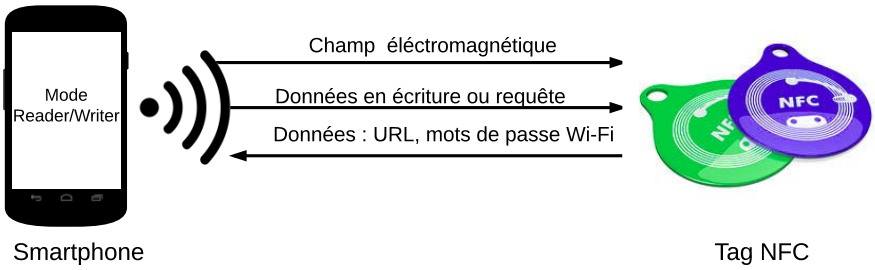

FIGURE 2.1 – Mode Reader/Writer

Format) [17] [18]. Les messages NDEF sont des messages au format binaire qui encapsulent une ou plusieurs données d'applications dans un seul message. Le format NDEF est devenu un format standard pour le stockage de données formatées sur les tags NFC et pour le transport de données entre deux périphériques actifs NFC.

Un message NDEF contient un ou plusieurs enregistrements NDEF (records). Les enregistrements peuvent être chaînés les uns aux autres afin de supporter une large taille de données. Chaque enregistrement NDEF possède les champs suivants : la donnée, la taille de la donnée, le type de la donnée et un identifiant optionnel. La taille d'un enregistrement est variable comme illustré sur la Figure 2.2. Nous avons utilisé ce format de données dans le prototype de dématérialisation 'OCAC' (neOCampus Access Control) que nous avons implémenté dans le cadre du projet neOCampus.

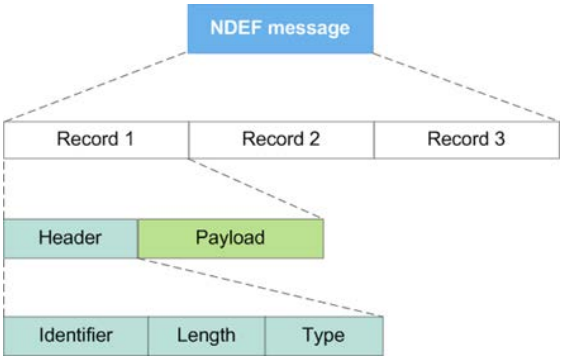

FIGURE 2.2 – Format d'un enregistrement NDEF

2.2.2 Mode peer-to-peer

La Figure 2.3 représente le mode de communication Peer-to-Peer [19]. Dans le mode peer-to-peer, deux périphériques NFC établissent une connexion bidirectionnelle afin d'échanger des informations. Les communications radios de ce mode sont standardisées dans l'ISO/IEC 18092 [20] permettant le modèle *requête/réponse* entre deux périphériques. Dans ce mode, les

smartphones NFC peuvent s'échanger toute sorte de données telles que des cartes de visite, des photos et des URL.

La communication en mode Peer-to-Peer est plus lente que dans les modes classiques lecteur / Card-Emulation, à cause de la gestion d'un protocole plus lourd nécessaire à la répartition des rôles entre les deux périphériques NFC. D'un point de vue usages : ce mode peut servir à initier des passerelles (appairage) avec d'autres technologies permettant des transferts de données à des débits numériques supérieurs que ceux du NFC (Bluetooth, Wi-Fi ou Wi-Fi Direct).

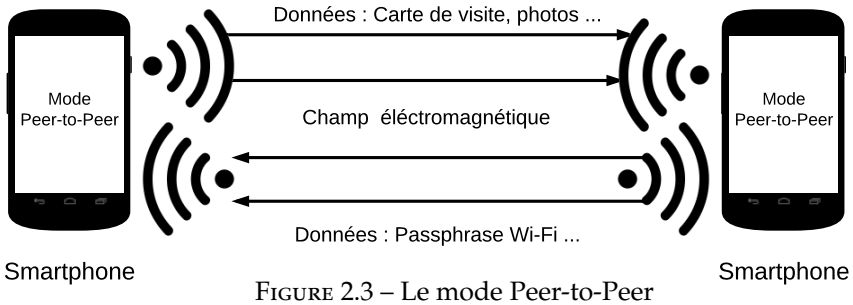

FIGURE 2.3 – Le mode Peer-to-Peer

2.2.3 Mode Card Emulation

La Figure 2.4 permet d'illustrer le mode Card-Emulation [21]. Dans ce mode, les périphériques NFC, essentiellement des smartphones, jouent le rôle d'un tag NFC classique. Lorsqu'un périphérique NFC est dans ce mode, le lecteur peut y accéder de la même manière qu'il accède à une carte tradionnelle. Alors qu'un tag NFC tire son énergie du champ électromagnétique du périphérique actif, le smartphone en mode Card-Emulation peut puiser dans sa propre source d'énergie (batterie). En contre partie, le smartphone est capable d'utiliser les périphériques d'entrées/sorties du téléphone comme l'écran ou le clavier virtuel pour étoffer les cas d'usages possibles. Dans le contexte de cette thèse, c'est ce mode qui nous intéresse dans le processus de dématérialisation de la carte sans contact traditionnelle.

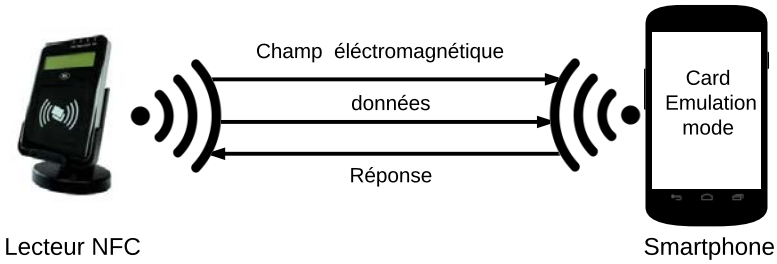

FIGURE 2.4 – Le mode Card-Emulation

2.3 Les cas d'usage du NFC

La technologie NFC est utilisée dans divers domaines allant du divertissement aux services en passant par les réseaux sociaux [22]. Nous présentons dans ce qui suit nombres de cas d'usages de la technologie NFC : le partage de liens, les réseaux sociaux, le partage de données et le contrôle d'accès.

2.3.1 Partage de liens

C'est le cas d'usage historique le plus simple. Une URL est encodée dans un tag NFC et mise à la disposition de nombreux utilisateurs. L'utilisateur utilise son smartphone NFC pour lire le contenu du tag et récupérer alors la page web souhaitée. Par exemple, ce cas d'usage est très fréquemment utilisé lors de conférence ou de séminaire afin de partager des liens utiles avec un grand nombre d'utilisateurs. Il existe aussi des solutions, telles que smartPoster, permettant de transférer des URL et de lancer des tâches sur le smartphone. Ces solutions sont en majorité basées sur l'utilisation du standard NDEF. Ce cas d'usage a été utilisé dans le prototype 'OCAC' notamment pour partager le lien vers l'application de contrôle d'accès à télécharger.

2.3.2 Les réseaux sociaux

Avec ce cas d'usage, nul besoin de chercher la personne que vous voulez ajouter à votre réseau social, ni de choisir le réseau social adéquat. En effet, le tag contient l'identité de la personne sur les réseaux sociaux ainsi que le réseau social sur lequel il est présent. En passant le smartphone sur le tag, le téléphone lit le réseau social indiqué et nous amène directement sur la page de la personne souhaité. Ce cas d'usage peut être vu comme une carte de visite de la personne lisible par toutes les personnes possédant un smartphone NFC.

2.3.3 Partage de données

Ce cas d'usage est relativement nouveau dans l'écosystème NFC. De plus en plus de smartphone sont munis de la technologie NFC qui fournit un cadre flexible et rapide de connexion entre deux terminaux. En effet, aucune configuration n'est requise pour que deux smartphones s'échangent des données en mode Peer-to-Peer. Plusieurs applications, parmi lesquelles Android BEAM permettent le partage de photos, cartes de visites et plusieurs types de fichiers entre deux smartphone communiquant via NFC.

2.3.4 La billetterie et contrôle d'accès

C'est probablement le cas d'usage le plus connu de la technologie NFC. En effet, de plus en plus de fournisseurs de services, principalement les société de transport en commun telles que la RATP à Paris et Tisséo à Toulouse, utilisent les cartes sans contact pour permettre aux utilisateurs

légitimes d'accéder au service désiré. Dans la majorité des cas, un protocole d'authentification est déployé entre la carte et le lecteur afin de déterminer si le possesseur de la carte a les droit d'accéder au service.

Cette liste n'est pas exhaustive mais de ces cas d'usages généraux nous pouvons dériver un ensemble quasi infini d'usages. Nous citons à titre d'exemple : partage de données de connexion Wi-Fi, partage d'adresses mail, partage de données de connexion Bluetooth, accès sans contact aux données médicales de malades etc.

Dans le cadre de ce travail, nous traitons la problématique du contrôle d'accès à travers les cartes sans contact NFC. Dans la section 2.5, nous ferons un focus sur les principales cartes sans contact utilisées dans différents systèmes de contrôles d'accès.

2.4 Les standards NFC

Il existe plusieurs normes régissant le fonctionnement de NFC. Parmi les normes essentielles au bon fonctionnement du protocole nous en avons identifié trois : La norme ISO/IEC 14443, ISO18092 [20] ou ECMA 340 et NFCIP-2 ou ECMA 352 [23].

2.4.1 La norme ISO 14443

Le standard ISO/IEC 14443 [24] est un standard international décrivant les paramètres des tags NFC comme spécifiés dans la norme ISO/IEC 7810 [25] et décrivant l'utilisation de ces cartes pour une compatibilité au niveau international. Ce standard décrit notamment le processus de sélection de la carte entrant dans le champ électromagnétique d'un dispositif générateur (lecteur NFC), le format des données et des échanges, le processus de requête/réponse pour la sélection des services sur la carte ainsi que le mécanisme d'anti-collision. Après l'exécution des phases de connexion décrites dans la norme ISO/IEC 14443, les protocoles de haut niveau utilisant NFC peuvent être initialisés. Ce standard se divise en quatre parties :

1. Partie 1 : Caractéristiques physiques,
2. Partie 2 : Partie radio et signal,
3. Partie 3 : Phase d'initialisation et d'anti-collision,
4. Partie 4 : Protocole de transmission.

La première partie concerne les caractéristiques physiques des cartes sans contact. Outre les définitions des périphériques Proximity Inductive Coupling Card, (PICC : carte), et un Proximity Coupling Device, (PCD : lecteur), cette partie donne des informations sur le champ électromagnétique utilisé lors de la communication et la manière d'alimenter les cartes.

2.4.1.1 Partie 2 : Radio et signal

Cette partie du standard spécifie les interactions radio bas niveau entre un PICC (carte), et un PCD (lecteur). Le PICC est couplé au PCD via un champ électromagnétique généré par ce

dernier.

Pour ces périphériques PICC et PCD il existe deux types de transmissions : Type A et Type B. Une carte ne peut supporter qu'un seul type. Notons aussi que chaque type possède ses propres procédures et de fait sont mutuellement incompatibles.

— **Type A** : Une modulation d'amplitude ASK 100 avec un codage de Miller modifié pour la modulation est utilisée pour transmettre les données du PCD au PICC. Le débit par défaut est de 106kbits/s dans les deux directions.
— **Type B** : Une modulation ASK 10 avec un simple codage NRZ est utilisée. Le débit par défaut est le même que pour le type A.

2.4.1.2 Partie 3 : Initialisation et anticollision

Quand un PICC rentre dans la zone de portée radio du PCD, un canal de communication doit être établi afin que les deux périphériques puissent communiquer. Cependant, plusieurs PICC peuvent être présents dans la zone de portée radio ce qui peut provoquer des collisions au niveau radio. Afin de remédier à ce problème, un système de sélection du tag (ou de la carte) par l'anti-collision [26] est mis en place afin de pouvoir sélectionner le tag voulu parmi tous ceux présents. Le mécanisme d'anti-collision permet d'assurer un déterminisme de choix du tag sélectionné pour garantir une communication unique et fiable.

Le mécanisme d'anti-collision diffère entre les cartes de Type A ou B. Dans chacun des types, chaque carte qui entre dans la zone de portée d'un lecteur s'active en utilisant le champ électromagnétique du PCD. Elle dispose à ce moment là d'assez d'énergie de la part du lecteur pour se mettre en mode "idle". Le lecteur peut communiquer avec les autres cartes. En effet, celles qui sont en mode "idle" ne doivent pas réagir aux communications du lecteur avec les autres cartes ce qui permet d'éviter les collisions des trames entre les différentes cartes.

Au moment où le lecteur envoie une requête permettant de sélectionner une carte, le mécanisme d'anti-collision spécifique au type de la carte est activé. Après la fin de ce mécanisme, la carte se met dans l'état "Actif" et la communication entre le lecteur et la carte peut désormais commencer.

2.4.1.3 Partie 4 : Protocole de transmission

Après l'établissement du canal de communication entre la carte et le lecteur, ce dernier est capable d'envoyer des commandes à la carte afin de pouvoir récupérer certaines informations. Chaque type de carte (A ou B) possède ses propres commandes. Nous allons dans ce qui suit, donner un aperçu sur l'activation du protocole pour le type A. Pour le type B, le même processus est employé sauf que la modulation lors de la transmission est différente.

Activation du protocole pour le PICC type A

La sélection de la carte dans la phase d'anti-collision doit être acquittée par l'envoi de la commande SAK de la part du PICC vers le PCD. Ce message contient des informations relatives à la compatibilité de la carte avec le standard ISO/IEC 14443. Dans le cas contraire, la carte indique qu'elle possède son propre protocole propriétaire.

Dans le cas où la carte est compatible avec le standard ISO/IEC 14443, le lecteur envoie un RATS (Request Answer To Select) à la carte. Cette commande contient deux valeurs très importantes : Card IDentifier (CID) et Frame Size Device Integer (FSDI). Le CID est un identifiant unique pour chaque carte dans le mode actif. Il est permet d'avoir une adresse logique pour le PICC sélectionné. Le FSDI encode la taille maximale du bloc que peut recevoir le PCD de la part du PICC. Cette valeur est inférieure ou égale à 256 octets.

2.4.2 NFCIP-1 (ISO 18092) ou ECMA 340

Ce standard [27] spécifie, en particulier, les schémas de modulation préconisés, le codage des données, le débit de transfert et le format des trames ainsi que les paramètres d'initialisation et les conditions requises pour le contrôle de collision durant cette phase d'initialisation. De plus, ce standard décrit un protocole de transport définissant les méthodes d'activation et de transfert des données via la communication NFC.

Le NFCIP-1 s'appuie sur les deux modes passif et actif. Le mode actif contrairement au passif permet à deux périphériques ayant leurs propres sources d'énergie de communiquer. L'un des deux périphériques est l'initiateur le second est la cible. Les débits offerts par ce mode sont : 106 kb/s, 212 kb/s et 424 kb/s.

2.4.3 NFCIP-2 ou ECMA 352

Le Near Field Communication Interface Protocol 2 (NFCIP-2) est défini par la norme ECMA 352 [23]. Ce standard définit une méthode pour choisir un des trois modes de communication définis dans ECMA-340 ou NFCIP-1, ISO/IEC 14443 (Cartes MIFARE) et ISO/IEC 15693 [28] (Tags RFID par exemple). NFCIP-2 peut dès lors être considéré comme une passerelle entre les interfaces standard déjà existantes.

Un autre objectif du protocole consiste à ne pas perturber des communications en 13.56MHz déjà en cours. En effet, cette fonctionnalité est garantie par l'utilisation d'un protocole CSMA (Carrier Sense Multiple Access). Lorsqu'une communication est déjà détectée, NFCIP-2 n'activera pas le champ électromagnétique du périphérique sur lequel il est implémenté.

Après avoir présenté la technologie NFC, ses modes de fonctionnement et les standards régissant cette technologie, nous passons maintenant à la présentation des différentes cartes NFC.

2.5 Les types de tags NFC

Il existe quatre types de tags définis par le NFC Forum [29]. Chaque type de tag est basé sur un standard dédié. La principale différence entre ces types réside dans la taille de la mémoire ainsi que les débits offerts.

2.5.1 Les tags de type 1

Ils sont basés sur le standard ISO/IEC 14443 type A. Par défaut, les tags sont en mode lecture/écriture mais généralement les constructeurs mettent les cartes en mode lecture seule. La taille de la mémoire disponible varie entre 96 et 2048 octets. Le débit de communication est de 106 kb/s. A cause de la faible mémoire disponible, ce type de tags est seulement utilisé dans le stockage d'informations de tailles très réduites telles que les URLs.

2.5.2 Les tags de type 2

Tout comme les tags de type 1, les tags de type 2 sont basés sur le standard ISO/IEC 14443 type A. Ils partagent les mêmes caractéristiques en terme de contrôle d'accès aux données. Cependant, ce type de tags offre une capacité de stockage minimale inférieure à celles des types 1. En effet, la capacité varie entre 48 et 2048 octets. Le débit de communication culmine à 106 kb/s.

2.5.3 Les tags de type 3

Les tags de type 3 sont basés sur le standard JIS X 6319-4 célèbre sous le nom de FeliCa [30] au Japon. Le débit de communication atteint presque le double des deux précédents types à 206 kb/s. La capacité mémoire est variable et peut au maximum atteindre 1 MB par service sachant que la carte peut héberger plusieurs services. Ce type de tags est conçu pour des applications complexes telles que l'accès à des services de transports et certains paiements. De par leurs complexité, ces tags sont plus chers que les précédents.

2.5.4 Les tags de type 4

Ces tags présentent une compatibilité totale avec le standard ISO/IEC 14443 (Partie 4 inclue). L'interface NFC est compatible Type A et B en même temps. Par défaut, les tags sont en mode lecture/écriture mais généralement les constructeurs mettent les cartes en mode lecture seule. La capacité mémoire culmine à 32 ko par service et le débit varie entre 106 kb/s et 424 kb/s.

2.6 Les cartes sans contact

De nos jours, nombres de systèmes de contrôle d'accès sont basés sur les cartes sans contact MIFARE. Il en existe plusieurs types et évolutions. Dans cette section nous avons choisi de présenter les principales cartes disponibles sur le marché : la MIFARE Classic [10], la MIFARE Ultralight [31] et la MIFARE DESFIRE [11].

2.6.1 La carte MIFARE Classic

En 1995, NXP a introduit les cartes sans contact basées sur NFC appelées MIFARE. Ces cartes ont été développées pour cibler des cas d'usages spécifiques allant des transports en commun au le contrôle d'accès et la vente de ticket d'accès. Il existe deux variantes de la carte MIFARE : la 1K et la 4K [1]. La taille de la clé de chiffrement utilisée est de 48 bits quelle que soit la carte. Nous décrivons dans cette section la structure logique, l'accès à la mémoire et les communications de la carte 1K.

2.6.1.1 Structure logique de la carte

La carte MIFARE classique est principalement une carte contenant de la mémoire avec ajout de certaines caractéristiques supplémentaires. Cette mémoire est de type EEPROM avec un découpage en blocs de 16 octets qui sont regroupés pour former des secteurs. La mémoire dans la carte 1K est divisée en 16 secteurs de 4 blocs chacun au contraire de la 4K où les 32 premiers secteurs sont constitués de 4 blocs tandis que les 8 secteurs restants sont divisés en 16 blocs. Le dernier bloc de chaque secteur contient les clés définissant les conditions d'accès aux quatre blocs présents sur le même secteur. La Figure 2.5 montre la structure logique la carte MIFARE Classic 1K.

Le bloc 0 du secteur 0 est appelé *bloc du fabriquant* et contient des données constructeurs. Les 4 premiers octets représentent un identifiant unique de la carte appelé *UID* [2]. L'octet suivant est appelé *BCC (Bit Count Check)* est utilisé pour vérifier l'intégrité de l'*UID*. Il est calculé en effectuant un XOR entre les 4 octets de l'identifiant. Les octets restants accueillent des données constructeurs. Ce bloc est approvisionné en données à la fabrication et est verrouillé ce qui assure que les données y figurant ne seront jamais modifiées.

Le lecteur doit être authentifié pour un secteur pour y accéder et effectuer des opérations mémoires sur ce secteur. Le bloc gérant l'authentification est le dernier de chaque secteur. Ce bloc héberge notamment :

— La clé secrète A sur 6 octets (48 bits)
— Les conditions d'accès aux blocs de ce secteur sur 4 octets
— La clé secrète B (qui est optionnelle) sur 6 octets également.

1. Cela correspond à la taille mémoire pour le stockage des données utilisateurs 1024 bits ou 4096 bits
2. Unique ID

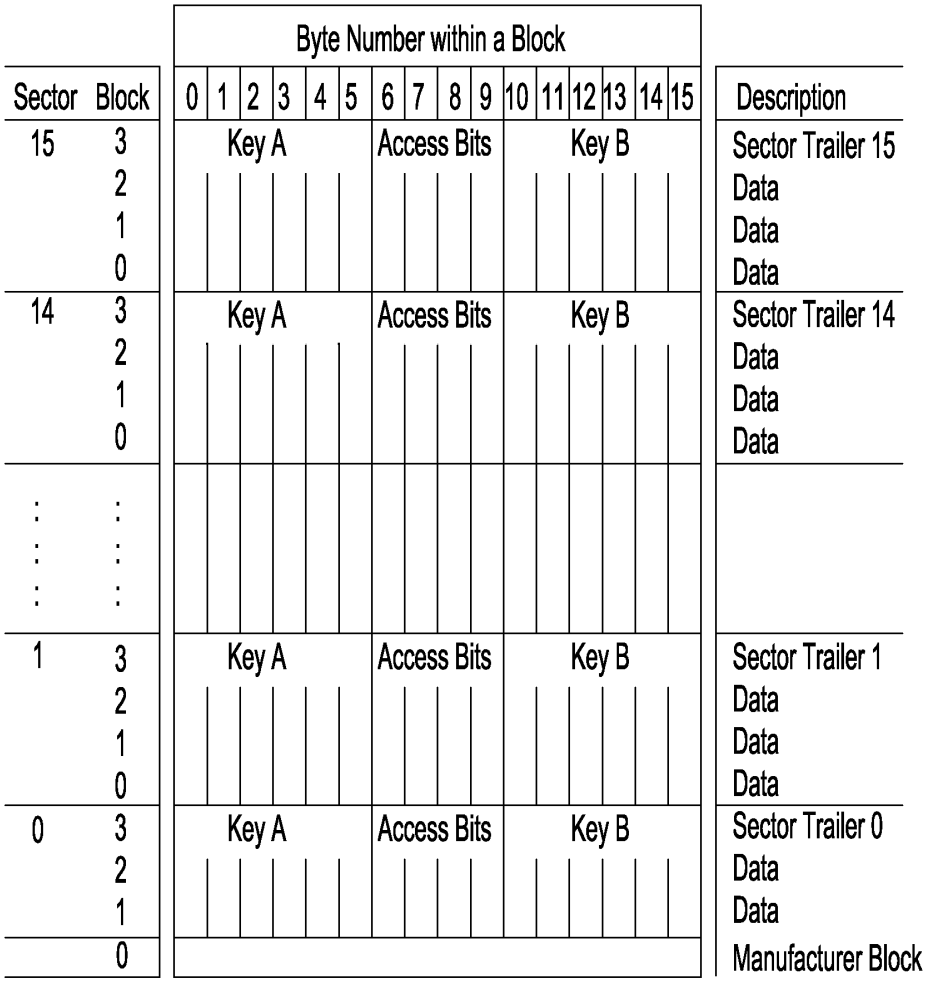

FIGURE 2.5 – Structure logique d'une carte MIFARE 1K

Il est impossible de lire la clé secrète A et la clé secrète B n'est lisible qu'au cas où la clé secrète A est connue. Les 'Access bits' définissent les conditions d'accès à la mémoire, le type de données du bloc et si le bloc est en lecture seule ou en lecture/écriture. Si la clé secrète B n'est pas utilisée, il est possible d'utiliser les 6 octets nécessaires au stockage de cette clé pour stocker des données.

Pour la carte MIFARE 1K, le second et le troisième blocs du secteur 0, et les trois premiers blocs des secteurs de 1 à 15 sont utilisés pour le stockage de données. Il est aussi possible de stocker des valeurs numériques qui sont notamment utilisées pour les porte-monnaies électroniques. Il est possible de configurer la politique d'accès à ces zones mémoires via les bits d'accès comme blocs à accès en lecture seule ou en lecture écriture. Toute modification du contenu d'un bloc implique une authentification préalable comme expliqué en section 2.6.1.4.

2.6.1.2 Les accès à la mémoire d'une carte MIFARE Classic

Les spécifications fonctionnelles pour les cartes MIFARE Classic 1K et 4K définissent six opérations pouvant être utilisées pour accéder et manipuler les données des blocs. Afin de pouvoir définir les conditions d'exécution de chacune des six opérations sus-citées, chaque bloc de données possède trois bits (bits dans les octets 6 à 9 du dernier bloc de chaque secteur) dont la combinaison de valeurs (8 au total) définit les autorisations que possède telle ou telle opération. Le dernier bloc du secteur possède lui aussi ses trois bits de contrôle d'accès, cependant, comme il n'est pas utilisé pour le stockage de données, les deux seules opérations disponibles sont la lecture et l'écriture de la mémoire du bloc.

Les opérations mémoires disponibles pour chaque bloc de données sont résumées sur le Tableau 2.2.

Tableau 2.2 – Les opérations mémoires sur une carte MIFARE Classic

Opération	**Description**	**Validité sur type de bloc**
Lecture	Lit un seul bloc mémoire	Blocs de données, Blocs de valeurs, Bloc de fin de secteur
Écriture	Écrit un seul bloc mémoire	Blocs de données, Blocs de valeurs, Bloc de fin de secteur.
Incrémentation	Incrémente une valeur et la stocke sur le registre interne de données	Blocs de valeurs
Décrémentation	Décrémente une valeur et la stocke sur le registre interne de données	Blocs de valeurs
Transfert	Transfert le contenu d'un registre interne vers un bloc	Blocs de valeurs
Restauration	Lire le contenu d'un bloc dans le registre interne de données	Blocs de valeurs

2.6.1.3 Les conditions d'accès à la mémoire

Comme défini précédemment, chaque bloc d'un secteur possède trois bits qui par leur combinaison de valeurs définissent les conditions d'accès à ce bloc. Afin de garantir l'intégrité de ces bits, ils sont stockés une fois de manière inversée et une autre fois de manière normale (non-inversée). Ces bits gèrent les droits d'accès à la mémoire en utilisant les clés secrètes A et B. Le format des bits d'accès est vérifié à chaque accès mémoire et si les règles d'accès définies sont violées, le secteur dans son intégralité est verrouillé de manière irréversible.

Après avoir exposé la structure mémoire de la carte nous allons présenter dans la section suivante l'algorithme d'authentification en trois passes pour les cartes MIFARE classic en pointant les vulnérabilités et les attaques possibles.

2.6.1.4 Authentification en trois passes pour les cartes MIFARE 1K

Dans le contexte des cartes MIFARE Classic 1K, le processus d'authentification déployé est de type *Challenge/Response* en trois passes.

Afin que le lecteur puisse accéder à la mémoire de la carte, il doit se soumettre à un processus d'authentifications en trois passes présenté sur la Figure 2.6. Dans un premier temps, le lecteur spécifie le secteur auquel il veut accéder en utilisant la clé A ou la clé B. La carte commence par lire la clé secrète ainsi que les conditions d'accès du dernier secteur et finit la première passe par l'envoi d'un challenge vers le lecteur. Lors de la deuxième passe, ce dernier calcule la réponse à ce challenge et l'envoie à la carte accompagnée d'un challenge pour cette dernière. Pour la troisième et dernière passe, la carte vérifie que la réponse du lecteur est bien correcte et calcule elle même la réponse au challenge de la carte puis le retourne au lecteur.

Après la vérification par le lecteur de la réponse de la carte, si l'authentification se passe correctement, le lecteur peut accéder aux zones mémoires qu'il a spécifiées. Si le processus d'authentification se passe mal, l'accès aux blocs sélectionnés sera refusé.

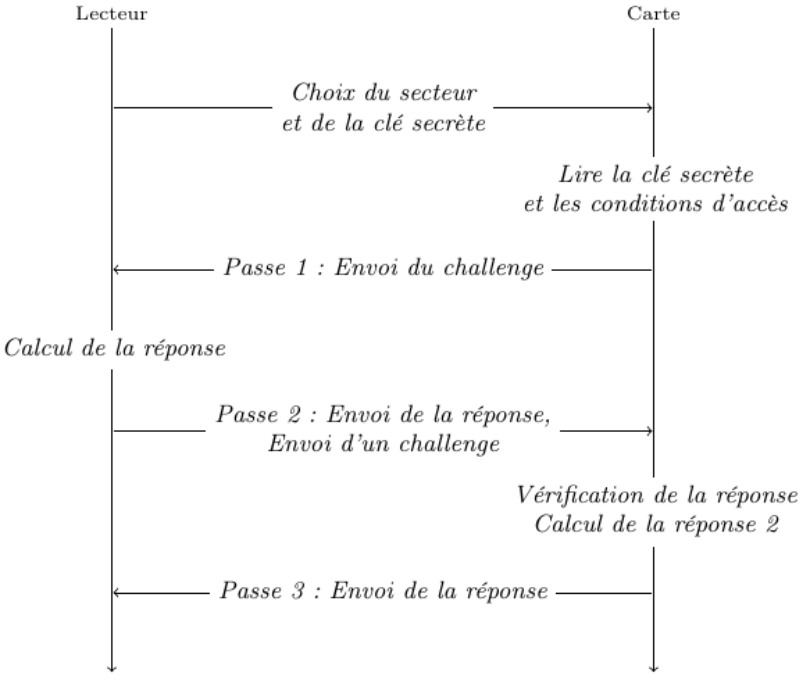

FIGURE 2.6 – Authentification en trois passes des cartes MIFARE Classic

2.6.2 Les cartes MIFARE Ultralight C

La carte MIFARE Ultralight C est aussi une carte à puce conforme à la spécification ISO/IEC 14443 A conçue pour supporter un nombre limité de services. La mémoire de cette carte est divisée en 48 pages mémoires accessible à l'utilisateur en lecteur et/ou écriture. La carte MIFARE Ultralight C possède des caractéristiques supplémentaires telles qu'un compteur sur 16 bits et l'authentification basée sur l'algorithme de chiffrement 3DES [12].

2.6.3 La carte MIFARE DESFIRE EV1

La MIFARE DESFire EV1 est une carte sans contact basée sur l'*open global standards* pour son interface et ses capacités cryptographiques. Comme son nom l'indique, cette carte est dotée de la capacité d'exécuter des algorithmes cryptographiques plus sûrs que ceux présents sur les cartes de génération précédentes. Parmi ces algorithmes, elle est capable d'exécuter du DES, 2DES, 3DES et de l'AES pour sécuriser le transfert de données via une communication authentifiée.

La carte DESFire EV1 possède un système de fichiers flexible pouvant contenir jusqu'à 28 applications simultanément et 32 fichiers par applications : des fichiers de données standards, des fichiers de sauvegarde, etc. La taille de chaque fichier est déterminée à sa création. Chacun des fichiers présents se voit associer quatorze clés et possède ses propres paramètres y compris le mode de communications (en clair ou chiffré) et des règles d'accès spécifiques.

Pour la partie sécurité, la carte DESFIRE se voit attribuer un numéro de série unique sur sept octets, une *master key* et les quatorze clés par application. Elle supporte l'authentification basée sur les algorithmes DES, 2K3DES, 3K3DES et AES ainsi que les algorithmes d'authentification spécifiés au niveau applicatif. La carte assure aussi la sécurité du système de fichier.

2.6.3.1 Système de fichiers de la carte MIFARE DESFire EV1

La carte MIFARE DESFire EV1 offre un système de fichiers flexible permettant d'accueillir 28 applications. Chaque application est identifiée par un identifiant d'application *AID* [1] sur 3 octets. Cette solution permet à un lecteur de sélectionner l'application adéquate pour le cas d'usage souhaité. Les droits d'accès aux fichiers des différentes applications diffèrent car plusieurs fichiers sont soumis à une authentification préalable à leur utilisation. Ces droits d'accès, codés sur 4 bits, sont liés à une des quatorze clés possibles, assignée au fichier à la création.

La carte DESFire EV1 supporte plusieurs modes de communication : le mode texte clair, le mode combiné (clair et chiffré) et le mode chiffré entièrement. Par défaut, tous les fichiers sont en mode communication en clair. Si l'accès à un fichier ou une opération sur un fichier requiert une authentification préalable, ces opérations sont verrouillées en attendant cette authentification et annulées si aucune authentification n'est faite. Le Tableau 2.3 présente les opérations possibles sur le système de fichiers des cartes DESFire EV1.

1. Application IDentifier

TABLEAU 2.3 – Types de fichiers et opérations sur une carte DESFire EV1

Type du fichier	Description	Opérations possibles
Fichier de données standard	Contient les données utilisateur non formattées	Lecture des données Écriture des données
Fichier de sauvegarde	Contient des données utilisateur non formattées en intégrant des mécanisme de sauvegarde	Lecture des données Écriture des données Valider les changements Annuler les changements
Fichier de valeurs	Stockage et manipulation d'un entier de 32 bits	Lecture des données Écriture des données Récupérer la valeur Créditer, Débiter, Valider les changements Annuler les changements
Fichier d'enregistrement linéraire	Stockage de données utilisateur formattées et structurées (programme de fidelité par exemple)	Lecture des enregistrements Écriture des enregistrements Vider les enregistrements Valider les enregistrements Annuler les changements
Fichier d'enregistrement cyclique	Stockage de données utilisateur formattées et structurées et automatiquement réécrire les précédents enregistrements	Lecture des enregistrements Écriture des enregistrements Vider les enregistrements Valider les enregistrements Annuler les changements

2.6.3.2 Sécurité des cartes DESFire EV1

Les cartes MIFARE DESFire EV1 possèdent plusieurs propriétés de sécurité assurant une fiabilité d'utilisation. Chaque carte se voit assigner, à sa fabrication, un UID unique sur 7 octets qui ne peut être modifié. Cette propriété permet de garantir l'unicité de la carte et et son non-clonage par un attaquant pour éviter qu'elle soit utilisée de manière frauduleuse. De plus, chaque carte possède la faculté de vérifier l'intégrité des données et de détecter tout changement sur ces dernières en utilisant des codes CRC16 et CRC32.

La confidentialité des données est assurée par l'utilisation d'algorithmes de chiffrement : DES, 2K3DES, 3K3DES et AES. Afin de pouvoir vérifier l'identité de la carte et du lecteur, le fabriquant de la carte prévoit un algorithme d'authentification de type *challenge/response* utilisant la capacité cryptographique de la carte d'exécuter les algorithmes de chiffrement type DES, 3DES et AES. Un processus d'authentification réussi permet de négocier un secret partagé, appelé clé secrète, utilisé dans la suite de la communication pour chiffrer les paquets qui transitent entre le lecteur et la carte.

2.6.3.3 Algorithme d'authentification en trois étapes

Le processus d'authentification en trois étapes implémenté par les cartes DESFire EV1 est détaillé par la Figure 2.7.

— **Sélectionner la clé AES à utiliser** : Le lecteur envoie à la carte un numéro correspondant au rang de la clé à utiliser. Par exemple, $0xFE$ correspond à la quatorzième clé.

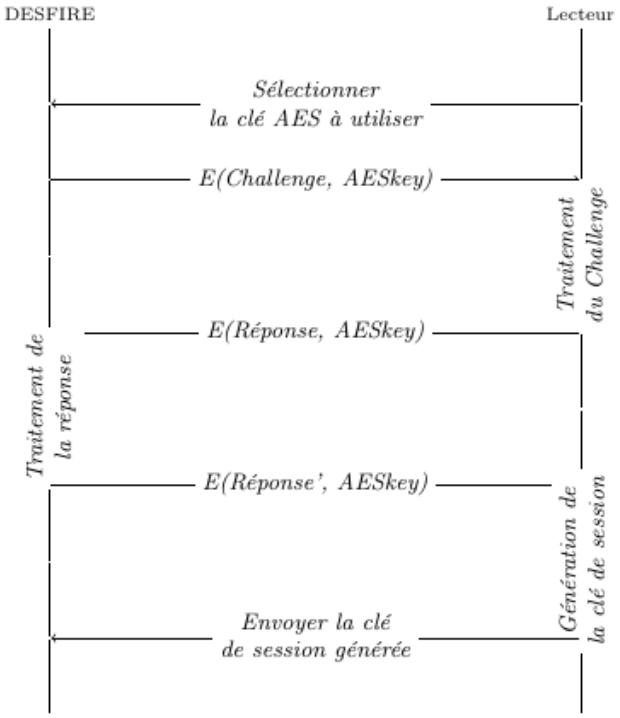

FIGURE 2.7 – Processus d'authentification des cartes DESFire EV1

— **Envoyer un challenge chiffré** : La carte génère de manière aléatoire un ensemble de 16 octets qui sera chiffré par la clé sélectionnée lors de l'étape précédente.
— **Traitement du challenge** : Le challenge est déchiffré par le lecteur avec la clé symétrique utilisée pour le chiffrement (clé commune entre la carte et le lecteur). La valeur obtenue est décalée d'un octet vers la gauche de manière circulaire (rotation). Le résultat est nommé $c1$. Le lecteur génère à son tour une série de 16 octets de manière aléatoire qui sera concaténée à la valeur c1 avant d'être chiffrée et renvoyer à la carte.
— **Traitement de la réponse R2** : Après déchiffrement du message reçu, la carte extrait les deux séquences de 16 octets et applique la rotation inverse à la valeur précédemment envoyée. Si la valeur ne correspond pas, le processus d'authentification est avorté. Sinon, la carte applique la même rotation à la deuxième valeur de 16 octets et la renvoie au lecteur.
— **Génération de la clé de session** : Le lecteur déchiffre le message reçu et applique une rotation inverse à la valeur reçue. Si la valeur ne correspond pas à celle générée précédemment l'authentification échoue. Sinon une clé de session AES 128 est générée avec la Formule 2.1.

$$SessionKey = RND_{card}[0-3] + RND_{Reader}[0-3] + RND_{card}[12-15] + RND_{Reader}[12-15] \quad (2.1)$$

2.6.4 Faiblesses des cartes sans contact

En dépit des mécanismes de sécurité implémentés dans les cartes sans contact présentées ci-dessus, les cartes MIFARE Classic et la carte DESFire EV1 présentent certaines vulnérabilités exploitées par des attaquants pour effectuer des opérations non-autorisées. Ces attaques permettent notamment de cloner les cartes MIFARE Classic 1K et permettent aussi de récupérer la clé de session négociée par la carte DESFire. Nous présentons dans ce qui suit, les principales vulnérabilités identifiées sur la carte MIFARE Classic et sur la carte MIFARE DESFire en exposant certaines attaques répertoriées dans la littérature.

2.6.4.1 Vulnérabilités de la carte MIFARE Classic

De manière générale, il existe quatre principales catégories de vulnérabilités :

— **Faiblesse du générateur pseudo-aléatoire [32]** : C'est la vulnérabilité la plus critique découverte sur la carte MIFARE Classic. Plusieurs attaques ciblent spécifiquement cette vulnérabilité ou la combinent à d'autres vulnérabilités afin de cloner la carte ou compromettre les données de la carte.

— **Faiblesse de l'algorithme cryptographique** : Pour la carte MIFARE Classic, c'est la faiblesse de l'algorithme de chiffrement CRYPTO1 [33]. C'est un algorithme de chiffrement propriétaire orienté flux développé par NXP qui a été cassé à partir de l'année 2008 [2]

— **Faiblesse du protocole de communication [34]** : Cette faiblesse est relative à la manière d'utiliser l'algorithme de chiffrement et non intrinsèquement dûe à ce dernier.

— **Faiblesse de l'implémentation** : Cette faiblesse est le résultat d'une mauvaise implémentation de l'algorithme de chiffrement ou du protocole d'authentification. Ces mauvaises implémentations introduisent des biais qui ne sont pas dus aux algorithmes eux-mêmes.

Vulnérabilité 1 : Faible entropie du PRNG

Comme précédemment mentionné, cette vulnérabilité est la plus critique découverte pour les MIFARE Classic. En effet, la taille de la séquence aléatoire est de 16 bits ce qui induit une entropie de 2^{16} (65536 valeurs). Cette entropie est très faible et selon [32], il suffit de 0.6 seconde pour générer l'ensemble des valeurs possibles. De plus, le générateur se réinitialise à un état connu à chaque fois que le tag est alimenté par le lecteur. Cette vulnérabilité majeure permet à un attaquant de deviner le challenge envoyé par le tag vers le lecteur et de compromettre le processus d'authentification.

Vulnérabilité 2 : Fuite d'informations à partir des bits de parités

Cette faiblesse est celle du protocole de communication. En effet, comme nous l'avons expliqué précédemment dans la section 2.6.1.4, lors de la deuxième passe de l'authentification, le lecteur

envoie la réponse au challenge de la carte tout en lui envoyant un challenge à son tour. A la réception des deux informations, la carte commence par vérifier les bits de parité avant de vérifier la validité des challenges [35]. Si un des bits de parité est incorrect, la carte ne répond pas à la sollicitation du lecteur. Dans le cas contraire, la carte enchaîne par la vérification de l'exactitude du challenge. Dans ce cas, si la vérification échoue, la carte envoie un message d'erreur de quatre bits ayant comme valeur *0x5* qui indique une erreur de transmission.

Cependant, ce qui fait la faiblesse du protocole de communication est que le message d'erreur est transmis de manière chiffrée. Le lecteur non authentifié ne peut, en théorie, déchiffrer le message qu'en connaissant le clair et la clé de chiffrement. Cependant, en connaissant le clair du message chiffré, nous pouvons procéder à une attaque à clair connu [34] afin de remonter à 4 bits du flux de la clé.

Vulnérabilité 3 : Déploiement du système avec les clés par défaut

Il s'agit d'une faiblesse dans l'implémentation. En effet, certains constructeurs implémentent leur système en utilisant certaines clés par défaut afin de faciliter les phases de tests et de démonstration pour les clients. Ces clés ont fini par fuiter et certaines d'entre elles sont répertoriées dans la liste ci-dessous.

- `A0A1A2A3A4A5, AABBCCDDEEFF;`
- `AABBCCDDEEFF, D3F7D3F7D3F7;`
- `4D3A99C351DD, 1A982C7E459A;`
- `FFFFFFFFFFFF, 000000000000.`

Il incombe à celui qui déploie le système de changer les clés afin d'éviter l'utilisation des clés par défaut. Cependant, la plupart des cartes déployées utilisent encore les clés par défaut. Cette faiblesse nous permet de dumper la mémoire de la carte sur le principe d'une attaque par dictionnaire (dictionnaire de clés) et permet de lire ou écrire n'importe quel secteur (sauf le bloc 0 du secteur 0).

2.6.4.2 Faiblesse de la carte MIFARE DESFire

Comme décrit dans les sections précédentes, la carte MIFARE DESFire possède plus de capacités cryptographiques que la MIFARE Classic. En effet, son algorithme d'authentification repose sur l'utilisation d'un chiffrement 3DES présentant plus de fiabilité que l'algorithme d'authentification en trois passes de la MIFARE Classic. Cependant, Oswald et al. dans [1] ont réalisé une attaque side channel sur la consommation électrique de la carte permettant de pouvoir remonter à la clé 3DES négociée entre le lecteur et la carte.

Pour ce faire, leurs expérimentations se basent sur un scénario *BlackBox* c'est à dire avec aucune connaissance préalable sur la carte (hormis la liste des commandes que peut exécuter le lecteur sur la carte et les spécifications). Par conséquent, il est nécessaire d'effectuer un mapping

entre les différentes portions de la trace générée par l'analyse de consommation et les opérations qui y correspondent avant d'effectuer l'attaque proprement dite.

Afin de préparer l'attaque *side channel*, les auteurs ont enregistré des captures de consommation d'énergie par la carte pour le protocole d'authentification en variant la clé qu'utilise la carte et les valeurs des challenges générés de part et d'autres. Ensuite, les auteurs procèdent à une *Correlation Power Analysis (CPA)* pour localiser les points sur les courbes d'énergie correspondants à la clé de la carte, et les deux challenges. Ils arrivent par ce biais à obtenir une corrélation significative permettant d'isoler chacune des opérations et notamment les entrées/sorties du chiffrement 3DES.

Après avoir repéré les opérations à cibler lors de l'analyse, les auteurs passent à la phase de récupération de la clé 3DES utilisée. En comparant plusieurs traces enregistrées, ils prennent conscience que même pour des données constantes, la forme de la trace varie d'une exécution à l'autre. Ceci est dû aux contre-mesures implémentées pour contrarier d'éventuelles analyses *CPA*. Afin de récupérer la clé, les auteurs commencent par établir un modèle de consommation en se basant sur des tests avec des clés connues. Après plusieurs expérimentations, ce modèle est en place. Ils ont ensuite monté une CPA afin de récupérer la clé 3DES en entier en récupérant 6 bits des SBOX sur chaque tour. Les auteurs affirment qu'avec leur attaque, l'extraction de l'ensemble de la clé 3DES pour une MIFARE DESFire peut être effectuée en approximativement sept heures avec leur équipement.

2.7 Conclusion

Nous nous sommes intéressés dans ce chapitre à la technologie NFC permettant de communiquer en mode sans contact. Nous avons d'abord détaillé les modes de fonctionnement de NFC parmi lesquels le mode Card-Emulation permettant d'émuler une carte sans contact sur un smartphone muni de NFC. Nous avons ensuite abordé les normes régissant le fonctionnement de NFC. Les différents types de tags NFC ont été présentés avant de faire un focus sur les cartes les plus utilisées dans le contexte du contrôle d'accès : la carte MIFARE Classic et la carte MIFARE DESFire. Nous avons détaillé leur structure et leur comportement avant de pointer les vulnérabilités de ces dernières les rendant inutilisables dans le contrôle d'accès aux bâtiments sensibles. Nous allons dans la suite du manuscrit, exposer notre démarche de dématérialisation de ces cartes sur des smartphones muni de NFC. Nous exposons dans le chapitre suivant les technologies permettant de faire du stockage et de l'exécution sécurisée sur un smartphone ce qui représente la pierre angulaire du processus de dématérialisation.

3 État de l'art des environnements d'exécution sécurisés

3.1 Introduction

De nos jours, les smartphones sont conçus pour traiter une grande quantité de données augmentant ainsi le spectre des applications qu'ils peuvent accueillir. En effet, les smartphones sont capables d'embarquer des applications commerciales, professionnelles comme celles qui traitent les données personnelles. Dans ce contexte, les smartphones doivent stocker et traiter des informations sensibles et critiques de manière sécurisée. Pour ce faire, la solution la plus adaptée consiste à utiliser des algorithmes de chiffrement pour garantir la confidentialité des données. Cependant, le stockage sécurisé des clés de chiffrement est une problématique avérée. De plus, l'introspection de la mémoire peut mener l'attaquant à la récupération de la clé en cours d'exécution. En effet, une application ayant assez d'autorisations peut intercepter la clé sur la mémoire volatile au cours de l'exécution de l'opération de chiffrement et/ou déchiffrement. Plusieurs travaux de recherches se sont intéressés aux différentes solutions de stockage et de traitement sécurisés des données sur un smartphone. Nous identifions deux principales familles de solutions : des solutions logicielles et des solutions matérielles. Dans ce chapitre, nous présentons ces deux catégories en mettant en exergue les avantages et les inconvénients de chacune d'entre elles. Une comparaison entre ces différentes solutions est fournie afin d'avoir un référentiel pour le choix de la technologie adéquate à chaque contexte d'utilisation. Dans la suite de ce chapitre, nous allons d'abord définir les environnements sécurisés en identifiant leurs caractéristiques avant d'exposer les solutions matérielles et logicielles pour l'exécution et le stockage sécurisé dans le monde du mobile.

3.2 Les environnements sécurisés

Un environnement sécurisé est un environnement qui permet à une application de stocker et de traiter de manière sécurisée les données sensibles telles que les données d'authentification [36]. Afin de pouvoir garantir cette caractéristique, un tel environnement doit posséder les propriétés suivantes :

— **Isolation de l'exécution** : Chaque application doit s'exécuter indépendamment des autres applications. Cette propriété garantit qu'aucune application malicieuse ne puisse accéder aux données sensibles en cours de manipulation par une application sécurisée. Cette propriété interdit aussi l'accès au code de l'application en cours d'exécution (présent en RAM par exemple).
— **Stockage sécurisé** : Cette propriété garantit la confidentialité et l'intégrité de toutes les données, y compris les exécutables, qui représentent l'application à lancer. De plus, il est aussi nécessaire de protéger les données manipulées et générées par l'application et éventuellement un cache/proxy utilisé pour augmenter les performances. Les données les plus sensibles représentent les mots de passes, les clés de chiffrements ainsi que les certificats.
— **Approvisionnement sécurisé** : Cette propriété garantit la possibilité d'envoyer des données à une application spécifique en cours d'exécution dans l'environnement sécurisé tout en s'assurant de la confidentialité et de l'intégrité des données échangées. Cette propriété inclut le fait de pouvoir installer de manière distante des applications sensibles et le transfert de clés de chiffrements et de certificats.

Dans la suite du manuscrit, une application s'exécutant dans un environnement qui satisfait aux propriétés sus-citées sera appelée application sécurisée.

3.3 Solution logicielle

L'évolution qui s'opère dans le monde des smartphones mène ces derniers à devenir de plus en plus performants. En effet, leurs performances tendent vers celles de certains ordinateurs. Cette nouvelle donne a amené chercheurs et industriels à adapter des solutions de sécurité disponibles pour les ordinateurs sur les smartphones. La virtualisation de systèmes d'exploitation s'inscrit pleinement dans cette démarche en fournissant la possibilité d'exécuter, entre autres, un OS de confiance permettant de mettre en application les exigences énoncées dans la section 3.2. Chaque OS possède sa propre pile d'exécution avec toutes les caractéristiques d'un OS ordinaire. Afin d'éviter une fuite de données d'un OS vers un autre, il est nécessaire de mettre en place un mécanisme d'isolation performant. La Figure 3.1 met en exergue les deux catégories de virtualisation. Pour la virtualisation de type 1, les deux systèmes d'exploitation sont physiquement indépendants alors que pour la virtualisation de type 2, l'OS standard du smartphone agit comme un hôte et l'instance supplémentaire peut dès lors être vue comme un invité. Afin de pouvoir réaliser ce partitionnement et de ce fait garantir l'isolation et la sécurité des deux OS, un composant logiciel est nécessaire : l'hyperviseur. Dans le reste de la section, le processus complet de virtualisation et différentes solutions existantes sont présentés.

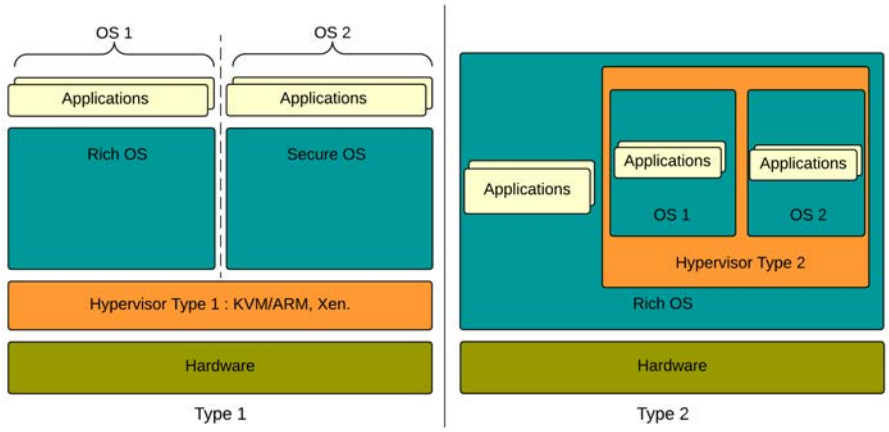

FIGURE 3.1 – Virtualisation types 1 et 2

3.3.1 Critères de conception d'un mécanisme de virtualisation sécurisé

La virtualisation vise à offrir deux environnements complètement séparés s'exécutant sur le même smartphone par une virtualisation au niveau matériel ou au niveau logiciel. Cette virtualisation s'appuie sur la notion d'hyperviseur, aussi appelé gestionnaire de machine virtuelle. Un hyperviseur est un programme qui permet à plusieurs systèmes d'exploitation de partager un seul hôte matériel tout en garantissant l'abstraction par rapport à l'architecture matérielle et en garantissant une isolation entre les deux OS cohabitant sur la plate-forme. Afin que l'hyperviseur puisse remplir son rôle, il est nécessaire de prendre en compte les exigences suivantes :

— **Portabilité** : L'hyperviseur doit pouvoir s'exécuter sur une multitude de plate-formes et d'architectures mobiles. Certaines parties du smartphone sont plus difficiles à virtualiser car elles sont spécifiques à un modèle donné telle que la possibilité d'avoir un processeur ou une RAM dédiés pour la partie GSM. Chaque processeur possède sa propre architecture et plusieurs cœurs. L'emplacement de l'hyperviseur joue donc un rôle très important pour établir la compatibilité avec le plus de matériel possible.

— **Compatibilité** : Le but de l'utilisation d'un hyperviseur est de créer une couche d'abstraction entre le matériel et le système d'exploitation quand la machine virtuelle est en cours d'exécution. L'hyperviseur doit fournir une solution pour le problème de la compatibilité des systèmes. En effet, plusieurs applications sont dépendantes d'un OS ou de librairies spécifiques ainsi que d'un écosystème donné (ApplePay ne fonctionne que sur un appareil Apple).

— **Sécurité** : Afin d'assurer la sécurité d'un tel système de virtualisation, il est nécessaire que l'hyperviseur garantisse une parfaite isolation de l'OS hôte et invité. Aussi, il est indispensable de compter sur la possibilité d'installer de manière sécurisée les applications sur le OS invité et de lui fournir les données nécessaires à son fonctionnement.

— **Faible complexité** : La faible complexité exigée pour l'hyperviseur vise à garantir sa

fiabilité. Cette caractéristique permet la vérification et la maintenabilité de ce composant.

— **Performances** : Une application exécutée sur l'OS invité, doit disposer des même performances qu'une application s'exécutant sur le hôte. De plus, la consommation d'énergie (batterie) ne doit pas être affectée de manière significative par la virtualisation d'OS afin que cette solution soit adoptée par les utilisateurs.

3.3.2 Architecture d'un système virtualisé : Proposition de VMware

Afin de fournir un système de virtualisation puissant avec un minimum de perte de performances, VMware a proposé un hyperviseur de type 2 pour la virtualisation sur smartphone [37]. Cet hyperviseur est installé sur Android et peut exécuter un OS invité parallèlement à l'OS hôte. La Figure 3.2 permet d'illustrer le concept d'hyperviseur type 2. Ce choix d'hyperviseur de type 2 permet de s'affranchir des contraintes de la diversité du matériel et des architectures processeurs. Un utilisateur peut recourir à cette solution sur son propre terminal afin de pouvoir exécuter les deux OS côte à côte. L'avantage de cette solution réside dans le fait qu'elle n'interfère pas dans le cycle normal de développement des applications.

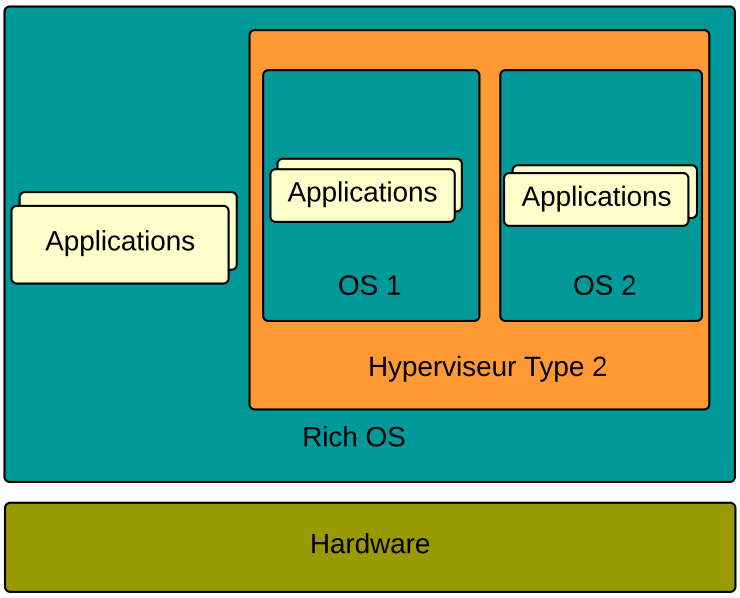

FIGURE 3.2 – Virtualisation type 2

Au niveau matériel, le processeur utilise le paradigme de temps partagé entre l'hôte et l'invité. Ce mécanisme consiste à ce que les deux OS partagent un processeur physique au cours de leurs exécutions respectives. Afin de basculer d'un système à un autre, le processeur s'occupe d'effectuer un changement de contexte en donnant le contrôle à l'un des deux systèmes. Du point de vue de l'OS hôte, l'exécution de l'invité est similaire à l'exécution d'un thread. Barr et al. dans [37] ont identifié des problèmes majeurs relatifs à la virtualisation : la compatibilité avec l'ensemble des instructions du processeur, le partage de la mémoire entre les deux systèmes, la virtualisation de la plate-forme, le problème de stockage des données, d'accès au réseau et le problème du téléphone multiple. Nous allons dans ce qui suit, détailler les challenges à relever pour la mise en place d'un tel système.

3.3.2.1 Partage de la mémoire

L'hôte et l'invité partagent la même mémoire sur le terminal. Sur un smartphone, nous retrouvons classiquement deux types de mémoires : la mémoire volatile (RAM) et la mémoire non-volatile ou persistante (stockage interne ou carte SD). Afin de résoudre le problème de partage de la mémoire, VMware propose de virtualiser deux sous-couches : Memory Management Unit (MMU) et la mémoire physique SDRAM du terminal. La virtualisation de la MMU permet d'avoir une abstraction de l'espace mémoire adressé. Cette opération doit inclure la traduction des adresses processeur ainsi que la protection de la mémoire. Le Virtual Machine Monitor (VMM) maintient une correspondance entre les adresses physiques de l'invité et les adresses physiques de l'hôte. Comme l'invité s'exécute dans l'espace utilisateur et dans l'espace noyau, il est nécessaire de disposer d'un mécanisme de protection contre les accès aux résultats de registres privilégiés. Des copies fantômes sont maintenues comme sauvegarde pour ces registres.

3.3.2.2 Virtualisation de la plate-forme

Conserver toutes les capacités du smartphone pendant la virtualisation implique la virtualisation de sa plate-forme matérielle. En effet, afin de permettre à l'invité de disposer de l'accès réseau et aux différentes applications (GPS, VOIP, etc) il est nécessaire de gérer les accès concurrents de l'hôte et de l'invité aux éléments physiques telles que les antennes et le matériel physique.

3.3.2.3 Stockage des données

La mémoire de l'hôte est utilisée en même temps pour stocker l'hyperviseur et l'image de la machine virtuelle. Cette dernière est bien trop volumineuse pour être stockée en mémoire flash, ce qui implique de la déplacer sur la carte SD. L'hyperviseur est installé en mémoire flash intégrée. Le challenge consiste à garantir un niveau de performances élevé tout en effectuant des opérations de vérification d'intégrité.

3.3.2.4 Accès au réseau

Les smartphones utilisent la pile TCP/IP afin de communiquer sur les réseaux sans fil. Dans le système virtualisé, cette pile est émulée par un framework de haut niveau appelé TCP para-virtualisé. Ce framework intervient au niveau du système de socket et non au niveau de l'interface du terminal. Cette approche est intéressante car elle permet de minimiser la perte en performances et de permettre plus de flexibilité dans le processus de déploiement. Cette architecture se divise en deux modules : le module client et le moteur de décharge. Le premier est exécuté côté invité et intercepte en cours d'exécution toutes les requêtes réseau formulées par cet OS. Le module de décharge est exécuté par l'hôte où il est chargé d'intercepter et de recevoir les requêtes de l'invité. Le trafic de ce dernier est acheminé via un tunnel afin de satisfaire aux exigences de sécurité nécessaires à l'OS invité.

3.3.2.5 Multiples lignes téléphoniques

La caractéristique principale d'un smartphone est de permettre aux utilisateurs de passer des appels téléphoniques. Pour un système virtualisé sécurisé, il est nécessaire de distinguer le téléphone virtualisé du le téléphone natif. La solution la plus triviale consisterait à acquérir un smartphone supportant deux cartes SIM. Ce type de téléphone existe, cependant les terminaux possédant cette capacité sont rares sur le marché. Une alternative plus réaliste consisterait à utiliser une seule carte SIM pour héberger plusieurs lignes. En effet, cette approche consiste à fournir de multiples IMSI (International Mobile Subscriber Identities) à une seule carte SIM permettant au terminal d'apparaître comme un ensemble de téléphones sur le réseau. Par conséquent, chaque smartphone virtualisé peut disposer de ses propres lignes téléphoniques.

3.3.2.6 Sécurité

La mise en œuvre d'une telle virtualisation doit nécessairement tenir compte de certaines contraintes de sécurité. En effet, l'isolation entre l'hôte et l'invité, la protection des données sensibles et les communications de manière sécurisée doivent atteindre un niveau de sécurité élevé. De plus, il est primordial de se prémunir de certaines attaques physiques dans le but d'éviter la divulgation de données sensibles si le téléphone est volé ou perdu.

Une autre menace significative réside dans le téléchargement d'applications malveillantes. Ces dernières peuvent induire des comportements indésirables et introduire de nouvelles vulnérabilités exploitables par un attaquant. Dans un système basé sur les permissions comme Android, l'utilisateur a un rôle prépondérant dans la sécurité du système tout entier. Cependant, la majorité des utilisateurs de ce système ne sont pas conscients du danger inhérent au fait d'accorder certaines permissions sensibles aux applications. Pour ces raisons, la gestion des applications dans l'OS invité (secure OS) doit prendre en considération ces attaques logicielles. En effet, autoriser l'installation d'applications ayant des privilèges élevés peut compromettre les données sensibles présentes dans le Secure OS. Dans le but d'accomplir cela, les bonnes pratiques

suivantes sont fortement recommandées afin d'aboutir à un niveau de sécurité satisfaisant :

— **Restriction sur les applications de l'hôte** : Les applications exécutées sur l'hôte ne doivent pas pouvoir accéder aux données de l'invité y compris le carnet de contacts, les SMS[1] et toutes les données stockées sur ce dernier. De plus, les accès réseaux ainsi que les communications téléphoniques doivent être sécurisées.

— **Protéger le trafic réseau de l'invité** : Une solution raisonnable pour protéger les communications réseaux de l'invité est de les faire passer par un tunnel chiffré. Dans ce cas, il faut sécuriser la procédure de négociation et éviter que l'hôte ne puisse intercepter les données négociées et plus particulièrement les clés de chiffrement.

— **Sécuriser les données sur la carte SD** : Les données stockées sur une carte SD d'un smartphone sont potentiellement accessibles par toutes les applications sur l'hôte et plus particulièrement par d'éventuels attaquants. De plus, les cartes SD peuvent être retirées du terminal ce qui les rend vulnérables au vol et aux attaques physiques. Le chiffrement est une solution permettant de rendre ces données non-lisibles par les applications non-autorisées et les potentiels attaquants. Cependant, la sécurité d'une telle solution repose sur la confidentialité de la clé de chiffrement ce qui se rapporte au problème de stockage de la clé. Plusieurs solutions ont été proposées dans le but de sécuriser ce stockage notamment celle proposée par Coojimans et al. dans [38] présentée dans la section 3.5.2.

— **Protéger le changement de contexte** : Une solution basique consisterait à utiliser des mots de passes pour éviter les changements de contextes non-approuvés. Si l'invité est inactif pendant une certaine période de temps correspondant au temps de la session, il doit être désactivé.

3.3.3 KVM/ARM : Exploiter les extensions de virtualisation d'ARM

Contrairement à la solution présentée lors de la section précédente, KVM/ARM est un hyperviseur de type 1 développé par ARM pour exécuter un OS non-modifié sur une architecture ARM. Depuis la version V7 de l'architecture ARM, la virtualisation est supportée sous forme d'une extension optionnelle [39]. Le processus de virtualisation repose sur ces extensions pour mettre en place un nouveau type d'hyperviseurs. La Figure 3.3 permet de visualiser le concept d'hyperviseur type 1.

ARM introduit un nouveau mode CPU plus privilégié appelé 'HYP mode' invoqué lors des appels systèmes les plus sensibles. Pour cette raison, l'hyperviseur doit, au minimum, être exécuté dans ce mode. Les machines virtuelles s'exécutent soit dans l'espace utilisateur soit dans l'espace noyau. Au moment d'exécuter des opérations sensibles, le mode HYP prend le contrôle jusqu'à la terminaison de cette opération. Une fois terminée, un changement de contexte est nécessaire pour revenir au mode d'origine de l'appel système.

Afin de limiter le coût en terme de performances qu'induit les multiples changements de contextes, le mode HYP ne gère pas tous les appels systèmes et les défauts de pages mémoire.

1. Short Message System

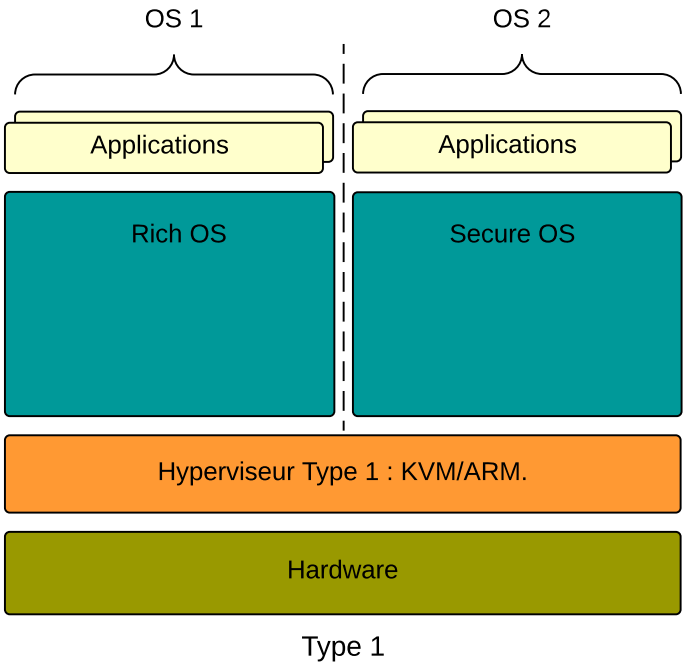

FIGURE 3.3 – Virtualisation type 1

Cette option permet de minimiser l'impact de la virtualisation sur les performances du smartphone.

Afin de faciliter le processus de développement de l'hyperviseur, ARM a décidé de réduire le contrôle des registres dans le mode HYP. Nous exposons dans ce qui suit les aptitudes offertes par l'extension de virtualisation ARM.

— **Virtualisation de la mémoire** : Quand une machine virtuelle est en cours d'exécution, les adresses physiques gérées par la VM sont appelées *adresses physiques intermédiaires* ou *adresses physiques de l'invité*. Il est nécessaire de les traduire en adresses physiques appelées *adresses physiques de l'hôte*. Cette opération de traduction a pour but de permettre à l'invité d'exécuter ses opérations sur le processeur physique. Le mode HYP est responsable de la gestion de ce processus de traduction.

— **Virtualisation des interruptions** : Un contrôleur générique des interruptions a été mis en place afin d'acheminer les interruptions soit de l'hôte soit de l'invité vers le CPU en détectant de quel environnement l'interruption émane.

— **Virtualisation temporelle** : ARM introduit un compteur afin de mesurer le temps réel passé et un *timer* pour chaque processeur. Ce compteur et ces timers sont utilisés pour déclencher une interruption à destination du CPU après une certaine période de temps.

Afin de garantir l'isolation entre le hôte et l'invité, les timers ne peuvent pas être gérés par l'invité.

En se basant sur l'extension de virtualisation ARM, Dall et al. ont proposé un hyperviseur dans [40] que nous exposons dans la section suivante.

3.3.3.1 Virtualisation en mode divisé

L'exécution de l'hyperviseur dans le mode HYP est attrayant mais aussi source de problèmes. En effet, des changements significatifs sont à prévoir dans le noyau Linux afin de l'exécuter dans le mode HYP. Dans un second temps, cette hypothèse risque d'affecter les performances natives du smartphone. Afin de résoudre ce problème, Dall et al. dans [40] ont proposé une virtualisation en mode divisé. L'hyperviseur peut dès lors être exécuté dans différents modes CPU ce qui lui permet de tirer partie des avantages de chaque mode. L'hyperviseur KVM/ARM profite des avantages de l'extension de virtualisation ARM et exploite dans le même temps les services Linux exécutés dans le mode Kernel. Cette approche permet de minimiser les changements dans le noyau Linux ce qui est nécessaire pour une adoption à grande échelle.

L'hyperviseur est divisé en deux composants : le *Lowvisor* et le *Highvisor*. Le *Lowvisor* exploite l'extension de virtualisation ARM présente dans le mode HYP. Il est chargé des opérations suivantes :

— Le *Lowvisor* charge le contexte d'exécution adéquat en configurant le système avec les paramètres appropriés.
— Le *Lowvisor* est responsable du changement de contexte entre l'hôte et l'invité. Cette caractéristique est rendue possible par le fait qu'il est exécuté dans le mode HYP.
— Le *Lowvisor* gère les interruptions et les exceptions systèmes.

Le *Highvisor* s'exécute dans le mode Kernel du noyau Linux. Il peut utiliser les fonctions standards de gestion de ressources du noyau comme par exemple l'ordonnancement des processus, l'allocation mémoire etc.

3.3.3.2 Virtualisation du CPU

Le modèle KVM/ARM procure une interface à la machine virtuelle identique au CPU physique incluant un accès aux registres d'états du processeur physique tout en garantissant que l'hyperviseur garde le contrôle de la partie matérielle. Les programmes qui s'exécutent sur la machine virtuelle doivent avoir le même accès persistant aux registres d'état que les programmes qui s'exécutent sur le CPU. Cette caractéristique est assurée par la sauvegarde et le changement de contexte gérés par le mode HYP.

3.3.3.3 Virtualisation de la mémoire

La virtualisation de la mémoire est effectuée en traduisant les adresses virtuelles utilisées dans l'invité pour tous les accès mémoires demandés par la machine virtuelle. La traduction

des adresses ne peut être faite en dehors du mode HYP. Le *highvisor* gère l'accès à la mémoire spécifiquement allouée à l'invité. Si un processus tente d'accéder à d'autres zones mémoires qui ne lui ont pas été allouées, cela provoquera une erreur et déclenchera une interruption. Ces erreurs sont levées et gérées par l'hyperviseur ce qui garantit l'isolation entre les deux environnements.

3.4 Solutions matérielles

Nous avons, dans la section précédente, exposé la solution logicielle qui consiste à virtualiser une deuxième instance d'un téléphone sécurisé. La sécurité de ce système repose presque entièrement sur la capacité de l'hyperviseur à garantir une bonne isolation entre les deux instances du téléphone. Cette solution a un impact direct sur les performances du système ce qui induit un manque d'adoption par la communauté. De plus, de part son architecture, cette solution est vulnérable aux attaques physiques lorsque l'attaquant peut disposer physiquement du smartphone. Cette attaque est rendue possible par le fait que l'hôte et l'invité partagent la même plate-forme. Afin de pouvoir disposer de solutions sécurisées permettant de répondre à ces problématiques et de n'induire aucun impact sur les performances du système, un autre type de solution a été développé. Il s'agit des solutions matérielles. Ces solutions présentent l'avantage de réduire de manière considérable les intrusions et les attaques et de garantir une meilleure protection contre les attaques physiques. De plus, les coûts de fabrication de plus en plus réduits permettent le déploiement *low cost* à grande échelle de ces composants. Enfin, ces solutions peuvent être certifiées (type critères communs EAL [41]) donnant ainsi une garantie sur leur niveau de sécurité et des attaques contre lesquelles elles peuvent résister. Cependant, l'inconvénient majeur de ces solutions réside dans le fait qu'elles sont souvent propriétaires et nécessitent l'accord du fabriquant pour le déploiement d'application tierces. Nous présentons dans cette section, trois solutions matérielles présentes sur les smartphones : les Secure Elements (SE), les Trusted Platform Module (TPM) et les Trusted Execution Environment (TEE).

3.4.1 Secure Elements (SE)

Les SE sont des composants électroniques ayant les mêmes caractéristiques que les cartes à puces traditionnelles [42]. En effet, les applications exécutées dans les SE sont des applications JavaCard. Ces cartes possèdent des propriétés d'anti-effraction et fournissent un niveau de sécurité très élevé [43] [44]. En effet, plusieurs SE [44] sont certifiés en EAL4+ [1] et au delà. Nous dénombrons trois principaux types de Secure Elements : SE embarqués, UICC SE et Micro SD SE [45]. Ces différents types sont présentés à la Figure 3.4

1. http://www.commoncriteriaportal.org/

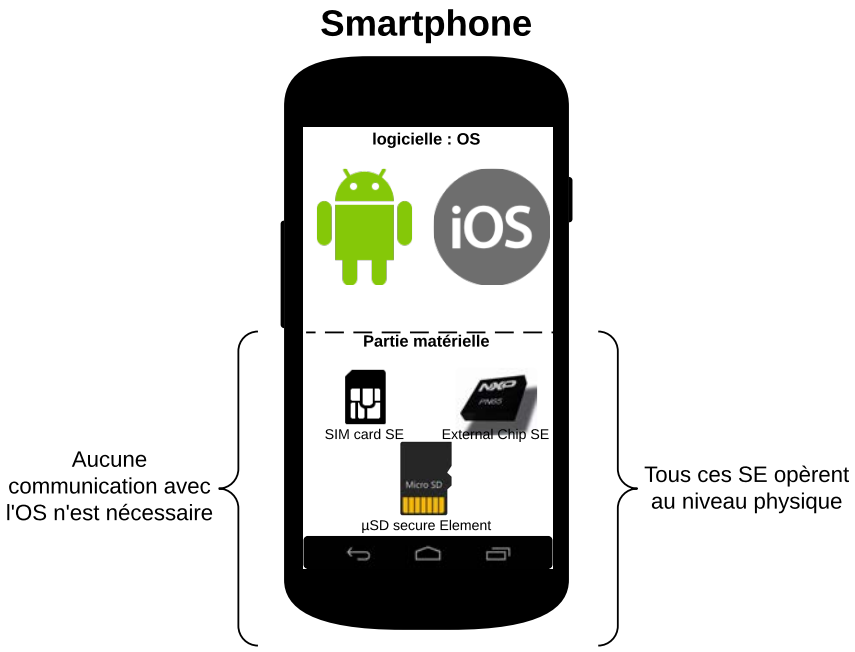

FIGURE 3.4 – Types de Secure Elements

3.4.1.1 SE embarqués

Pour ce type de SE, le composant est directement câblé à la carte mère. Il ne peut être ni enlevé ni transféré sur un autre téléphone. Dans ce scénario, la sécurité de cette solution est équivalente à la sécurité d'une carte à puce traditionnelle. En effet, sur un téléphone Android muni de la technologie NFC, le SE embarqué est seulement relié au contrôleur NFC et ne peut communiquer avec les autres composants du téléphone.

L'inconvénient de cette solution est la difficulté de définir les droits d'accès supplémentaires à la puce permettant à des tiers de déployer leurs applications. Les droits d'accès définis par les fabricants sont souvent très restrictifs et ne permettent pas aux développeurs de déployer leurs applications sur le SE sans accord préalable avec le fabriquant de la puce. Cette restriction rend cette solution difficilement adoptable à grande échelle.

3.4.1.2 UICC SE

Les cartes SIM délivrées par les opérateurs télécoms peuvent contenir un environnement sécurisé pour le stockage de données sensibles et l'exécution d'applications sécurisées. L'avantage de cette solution réside dans le fait que tous les téléphones peuvent utiliser cette solution au contraire des SE embarqués qui ne sont pas présents sur tous les téléphones.

Cependant, l'obligation d'avoir des accords avec les opérateurs télécoms constitue un inconvénient majeur et un frein certain au déploiement d'applications sécurisées sur les cartes SIM. En effet, les opérateurs télécoms doivent signer électroniquement les applications autorisées à être exécutées sur les cartes SIM. Ce processus de signature est lourd et constitue un obstacle pour les développeurs qui limite le déploiement à grande échelle de cette solution.

3.4.1.3 Micro SD SE

Ce type de SE possède le même niveau de sécurité que les UICC SE. Contrairement aux deux premiers SE, ce dernier n'est pas directement câblé à la carte mère du téléphone et ne nécessite pas d'accord avec les opérateurs télécoms. Ces cartes sont détachables et transférables vers d'autres téléphones. Cette propriété constitue en même temps un avantage et un inconvénient majeur en terme de sécurité pour cette solution. Le fait de pouvoir enlever et transférer la carte peut répondre à des besoins de flexibilité. En effet, dans le cas où l'utilisateur change souvent de téléphone, cette propriété lui permet de garder ses données sensibles et ses applications sécurisées sur la carte Micro SD et les transférer facilement d'un téléphone à un autre. Néanmoins, le fait de retirer la carte l'expose à des risques de vol.

3.4.1.4 Analyse comparative

Nativement, ces trois solutions possèdent le même niveau de sécurité. Cependant, la manière de les connecter au smartphone peut introduire des biais pour des attaques. Les SE embarqués sont directement connectés à la carte mère du smartphone et peuvent interagir avec le contrôleur NFC de manière directe au contraire de ses concurrents qui doivent passer par les bus, non sécurisés, de l'appareil et à travers Android. Pour IOS d'Apple, l'application de paiement ApplePay utilise les SE pour stocker les données de la transaction. Cependant, aucune information ne filtre sur la manière dont fonctionne son SE.

Le modèle d'approvisionnement pour les SE est basé sur l'exclusivité d'accès au périphérique par le fabricant. En effet, seules les applications signées par les fabricants des SE peuvent être exécutées sur ce dernier. Cette propriété réduit la surface d'attaque à partir du smartphone. D'un autre côté, cette restriction constitue un frein au développement des solutions basées sur les SE car il est difficile d'obtenir les accords nécessaires avec les fabricants pour le déploiement d'applications.

3.4.2 Trusted Platform Module (TPM)

Les TPM ont une architecture basée sur un micro-contrôleur avec des capacité cryptographiques accrues. En 1999, le Trusted Computing Group (TCG) [46] a été chargé, en collaboration avec différents industriels d'élaborer une spécification pour une nouvelle plate-forme dans le but d'assurer l'anonymat et la sécurité dans le monde des ordinateurs portables. Auparavant, leur sécurité reposait sur le chiffrement, l'authentification login/password et des jetons de connexion. Cependant, ces éléments ont été sujets à plusieurs faiblesses : vol de données, accès non-autorisé à l'ordinateur et des accès non-autorisés au réseau.

— **Vol de données** : A cause de la mobilité des ordinateurs portables, la probabilité liée au vol du terminal est plus élevée que celle d'un ordinateur fixe. Pour éviter la fuite de données sensibles présentes sur le terminal, la solution adoptée est de chiffrer les données sur l'ordinateur. Cette solution induit une faiblesse majeure : les clés de chiffrement utilisées dans le processus de chiffrement sont stockées sur le disque dur et sont sensibles à des attaques physiques sur la mémoire telles que l'introspection de la RAM et du disque dur. Pour résoudre ce problème, le TPM propose un mécanisme de stockage sécurisé des clés de chiffrements.

— **Accès non-autorisé au terminal** : Si n'importe qui peut accéder au terminal, il faut limiter l'impact des attaques perpétrées par des utilisateurs malveillants. Pour éviter l'accès de personnes malveillantes, plusieurs solutions ont été développées comprenant l'authentification à l'aide du couple login/password et de l'authentification biométrique [47]. Cependant, ces solutions présentent aussi des faiblesses et sont vulnérables à plusieurs attaques. Pour le premier type d'authentification, des attaques par dictionnaire et par force brute peuvent mettre à mal ce genre de système d'authentification [48]. Pour l'authentification biométrique, ces données peuvent être falsifiées pour tromper le système d'authentification [49].

— **Accès non-autorisé au réseau** : Un ordinateur portable volé avec les paramètres d'authentification peut avoir accès au réseau interne de la société et voler des données sensibles. Plusieurs solutions ont été déployées dans le but de garantir la sécurité du réseau d'accès. Parmi ces solutions, nous pouvons citer le Windows Network Logon [50]. Ces solutions présentent cependant plusieurs faiblesses [51] permettant à un attaquant de contourner ces mécanismes de sécurité. En outre, les certificats utilisés dans le processus d'authentification peuvent être falsifiés. Afin de remédier à ces problèmes, le TPM propose de mettre en place une méthode d'authentification basée sur les PKI [1] pour l'authentification de la plate-forme et d'effectuer une protection matérielle des données d'authentification.

Le TPM est une implémentation matérielle des spécifications du groupe TCG dans le but de fournir des solutions pour les problèmes sus-cités.

1. Infrastructure à clés publiques (Public Key Infrastructure)

3.4.2.1 Architecture des TPM et caractéristiques

Le TPM est un micro-contrôleur sûr avec des capacités cryptographiques supplémentaires [52]. Il est capable d'accomplir des opérations cryptographiques très complexes allant du chiffrement symétrique au chiffrement asymétrique type RSA [53]. L'avantage de l'utilisation d'un TPM est qu'un développeur n'a pas à connaître l'implémentation des algorithmes. En effet, celles-ci sont inclues sous fome binaire dans le TPM et uniquement accessibles au travers d'une interface de programmation (API). Les fonctionnalités cryptographiques qu'un TPM possède sont les suivantes :

— **Accélérateur RSA** : Un module interne effectue les opérations de chiffrements/déchiffrements RSA avec une taille maximale de clé de 2048 bits. Un autre module intégré permet de faire de la signature et de l'*emballage de clé* [1].

— **Calcul de hashs** : Le TPM est capable de calculer les hashs correspondant à des données fournies en entrée. La capacité de calcul et de stockage du TPM n'est pas suffisante pour traiter des quantités importantes de données (clés de chiffrements). L'objectif principal du TPM est de traiter des données sensibles de petite taille telles que les clés de chiffrements et les certificats.

— **Génération pseudo-aléatoire** : Le TPM possède un générateur pseudo-aléatoire matériel qui au contraire des PRNG logiciels présentent une sécurité accrue. Il est notamment utilisé dans les processus de génération de clés et de challenges pour les algorithmes d'authentification.

3.4.2.2 Architecture d'un TPM

La Figure 3.5 présente les principaux éléments de la plate-forme TPM : l'Endorsment Key (EK), l'Attestation Identities Key, les certificats et les Platform Configuration Registers (PCRs).

— **L'Endorsment Key (EK)** : Elle consiste en une paire de clé RSA privée/publique d'une taille de 2048 bits. La clé privée est stockée dans le TPM et n'est jamais utilisée à l'extérieur. Cette caractéristique permet de protéger la clé face à un attaquant externe. Cette clé est unique pour chaque TPM et son processus de génération diffère d'un constructeur à un autre. En effet, une première catégorie de constructeurs utilise les commandes spéciales déjà intégrées au TPM afin de lancer le processus de génération. D'autres constructeurs préfèrent générer la clé en dehors du TPM avant de la stocker à l'intérieur et de sceller définitivement l'accès à la zone mémoire contenant cette clé.

— **L'Attestation Identity keys** : Le but principal de ces clés est d'effectuer une authentification avec un fournisseur de service. Cette authentification diffère d'une authentification utilisateur classique. En effet, au lieu d'authentifier une personne, c'est la plate-forme qui est authentifiée.

— **Les certificats** : Le TPM stocke trois sortes de certificats : Endorsment Certificate, Platform

1. C'est une technique consistant à chiffrer avec une clé symétrique la clé privée d'un chiffrement asymétrique

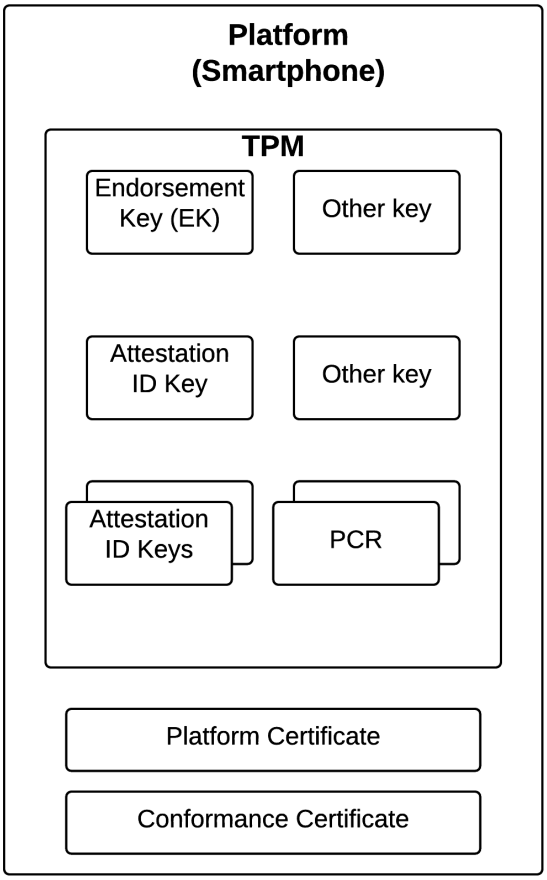

FIGURE 3.5 – Architecture d'un TPM

Certificate et Conformance Certificate.

- *Endorsement Certificate* : L'utilité de ce certificat réside dans le fait d'assurer l'intégrité de l'Endorsement Key (EK) stockée dans le TPM. Ce certificat peut être délivré par la même autorité ayant délivré la clé. La confiance dans le TPM est garantie par l'unicité de l'EK pour chaque composant et les certificats protègent la clé contre l'usurpation de cette clé.
- *Platform Certificate* : Ce certificat est délivré par le constructeur du TPM. L'utilité de ce certificat réside dans le fait de s'assurer que tous les composants de sécurité fournis avec la plate-forme sont authentiques. Ce certificat permet d'établir une confiance dans la plate-forme et de s'assurer que cette dernière n'a pas été détournée.
- *Conformance Certificate* : Ce certificat est produit par un laboratoire d'évaluation tiers ou par le constructeur lui-même. Il permet de garantir que les propriétés de sécurité revendiquées par le constructeur sont authentiques.

— **Platform Configuration Register (PCR)** : Les PCRs sont utilisés afin d'associer une don-

née sensible à un terminal spécifique. Ce processus d'association permet de garantir l'utilisation exclusive de cette donnée par ce terminal spécifique. Il s'agit d'une forme d'authentification du TPM. Certaines informations matérielles et/ou logicielles spécifiques à l'état actuel du terminal sont compilées afin de calculer une valeur particulière stockée dans les registre PCRs. Cette valeur est fortement liée à un terminal unique et est utilisée pour authentifier le TPM au moment où il essaye d'accéder à la donnée associée. Si le processus d'authentification échoue, l'accès à la donnée n'est pas autorisé.

3.4.2.3 Association de données critiques à un TPM

Associer des données critiques à un TPM spécifique est équivalent au fait de rendre ces données non utilisables par d'autres TPM. Après le calcul de la valeur particulière pour discriminer un terminal particulier, les données sensibles à associer sont mélangées avec un certain nombre de PCR. Le résultat de cette opération est chiffré et stocké sur le TPM. Au moment de l'accès à la donnée sensible pour utilisation, cette dernière doit être déchiffrée. Dans un second temps, la plate-forme recalcule la valeur des registres PCR utilisés dans le processus d'association afin de pouvoir extraire la donnée.

Le TPM peut être utilisé comme une PKI fournissant les certificats et les clés de chiffrement nécessaires à l'établissement de communications sécurisées. Dans le contexte des smartphones, les TPM peuvent être très utiles pour assurer la sécurité d'un système avec différentes entités (Fabriquant du smartphone, opérateur mobile, utilisateurs etc).

3.4.3 Trusted Execution Environments (TEE)

Les TEE [54] [55] [56] ont été développés pour résoudre les problèmes de compatibilité et de performances induits par l'utilisation des Secure Elements et des TPM. Dans cette section, nous faisons une description détaillée de la technologie TEE en pointant ses avantages et ses inconvénients avant de finir par une analyse comparative des différentes technologies présentées en amont.

3.4.3.1 Architecture générale

Les TEE sont une combinaison d'une partie matérielle et d'une autre logicielle. La partie matérielle correspond au processeur physique tandis que la partie logicielle consiste en un système d'exploitation sûr visant à exécuter les applications critiques. Dès lors, nous pouvons distinguer deux environnements au sein d'un TEE :

— **Rich Execution Environment (REE)** : Il est aussi appelé Normal World Execution Environment. Il représente le système d'exploitation (OS) standard du téléphone tel qu'Android par exemple. Le terme *Rich* décrit les fonctionnalités étendues de l'OS comme la gestion de l'appareil photo, la téléphonie et les applications courantes. Ces nombreuses fonctionnalités augmentent la taille du code et de facto augmente le risque qu'une vulnérabilité

ne soit découverte.

— **Trusted Execution Environment (TEE)** : Il représente l'OS sécurisé responsable de l'exécution des opérations sensibles telles que le chiffrement ou la signature. Il possède aussi la capacité de sécuriser les entrées/sorties en utilisant un mode sécurisé pour les bus connectant les I/O au processeur.

3.4.3.2 Composants et caractéristiques d'un TEE

Le TEE fournit une zone du processeur avec un OS sécurisé et une isolation totale avec le Rich OS afin d'améliorer la sécurité du traitement et du stockage des données sensibles. L'architecture d'un TEE est décrite sur la Figure 3.6.

L'autre fonctionnalité importante des TEE est le stockage sécurisé. En effet, dans le cas d'opérations cryptographiques, la génération et le stockage de la clé de chiffrement doivent être sécurisés afin de ne pas compromettre le processus de chiffrement. D'un point de vue pratique, la clé doit être présente sur le smartphone mais isolée des données utilisateur afin de s'assurer qu'aucune fuite ne soit possible.

Contrairement au SE, le TEE fournit un canal de communication sécurisé entre le processeur et les périphériques d'entrée/sortie, tout particulièrement pour la saisie et l'affichage sur écran. De ce fait, le TEE utilise un canal sécurisé pour les opérations entrées/sorties alors que le SE s'appuie sur les opérations de base implémentées sur l'OS standard du smartphone. L'affichage sécurisé est aussi une propriété importante car l'utilisateur doit être sûr que ce qu'il voit sur l'écran provient bien de l'application sécurisée et non pas d'un malware. Par exemple, un malware peut afficher une boite de dialogue pour récupérer un mot de passe de l'utilisateur en se faisant passer pour une application légitime.

Pour illustrer la différence entre le TEE et le SE prenons le cas où le système sollicite l'utilisateur pour un code PIN. Le SE est dépendant des fonctionnalités standards fournies par Android. Plusieurs attaques [57] [58] permettent à un attaquant d'intercepter et d'enregistrer le mot de passe saisi. Pour sa part, le TEE utilise un mécanisme sécurisé d'entrée/sortie basé sur des bus sécurisés rendant les attaques sus-citées impossibles.

Autre caractéristique importante du TEE : le Secure boot. Il consiste à vérifier l'intégrité du Secure OS afin de détecter tout éventuel changement. Si le processus de vérification échoue, le processus de boot suivant est interrompu :

— Lecture du contenu de la ROM (qui est verrouillée à la fabrication).
— Vérification de la signature et l'intégrité du secure OS.
— Chargement du Secure OS qui prend le contrôle.

Dans le monde des TEE, deux systèmes hétérogènes cohabitent, par conséquent, il est nécessaire de définir des règles d'isolation entre les deux systèmes pour éviter les fuites de données. Ainsi, il est nécessaire de mettre en place un troisième mode nommé *Monitor Mode*. Il est chargé de faire la sauvegarde et le changement de contexte entre le Rich OS et le Secure OS. Il est chargé d'assurer une étanchéité entre l'exécution du Rich OS et celle du Secure OS.

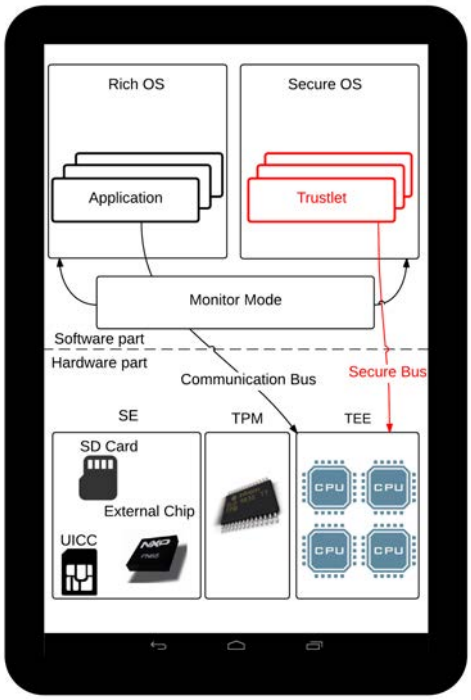

FIGURE 3.6 – Architecture d'un TEE

3.4.3.3 Le Secure OS

Le Secure OS est un système d'exploitation avec un ensemble limité d'instructions. Cette contrainte est nécessaire pour réduire la surface d'attaque et en limitant les vulnérabilités potentielles. Cet OS a également la capacité d'ordonnancer différentes applications selon des règles définies. Les applications exécutées par un Secure OS sont appelées *Trustlets*. Elles sont chargées d'effectuer les opérations cryptographiques comme la génération de clés, le chiffrement et le déchiffrement par exemple. Elles sont signées par le fabricant du processeur ou le fournisseur du Secure OS et cette signature est vérifiée avant chaque exécution de la Trustlet. Certaines API, dont le code est vérifié et analysé, peuvent être rajoutées pour étendre le spectre des opérations possibles à condition que ces dernières n'ajoutent pas de brèches de sécurité.

Jusqu'à maintenant, la majorité des Secure OS développés pour les TEE sont propriétaires ce qui rend le composant inaccessible pour des applications tierces. En effet, pour qu'un développeur puisse déployer son application sur la partie TEE, cette dernière devra être signée par le fabricant du processeur ou le fournisseur du Secure OS. Ces sociétés sont réticentes à l'idée d'intégrer des applications tierces dans leur écosystème ce qui rend l'adoption de la technologie TEE par les développeurs très limitée. Cette contrainte est également similaire pour les SE et les TPM. Cependant, pour ce qui est des TEE, des signes d'ouverture sont constatés de par le développement de TEE open-source dont nous présentons entre autres OP-TEE et OPEN-TEE

en section 3.4.3.4.

3.4.3.4 Panorama des Secure OS

Au moment de l'écriture de ce manuscrit, plusieurs implémentations des Secure OS pour les TEE sont disponibles. Dans ce qui suit, nous allons décrire les Secure OS les plus répandus en pointant leurs principales caractéristiques.

— **Sierra TEE** : Sierra TEE [59] est un Secure OS pouvant fonctionner avec Android, Linux ou BSD comme Rich OS, permet d'effectuer des opérations de gestion de clés, Data Right Management (DRM [1]) et sécurisation des opérations d'entrée/sortie. Par exemple, Sierra TEE assure la confidentialité de la saisie des mots de passe et des informations sensibles sur Android. Selon le fournisseur de l'OS, le système d'exploitation Sierra TEE propose un processus de gestion de l'intégrité du Rich OS et effectue de multiples analyses parmi lesquelles celles vérifiant le système de fichier et le système d'exploitation Android ainsi que la table d'interruptions. Ces tests visent à garantir la sécurité de l'exécution d'Android.

— **Genode** : Genode [60] est un Secure OS avec une complexité très faible. Son code source tient en approximativement 10000 lignes ce qui permet une vérification du code afin de détecter d'éventuelles failles inhérentes à l'écriture de l'OS. Genode gère l'exécution du driver de l'écran tactile ainsi que les périphériques d'entrées/sorties. Cette caractéristique permet une interaction sécurisée entre l'utilisateur et le TEE même au travers du matériel (écran, clavier virtuel, etc.). Le fournisseur de l'OS revendique le fait qu'aucune information sensible ne fuite durant l'exécution de la Trustlet. Ce Secure OS peut notamment fonctionner avec un Linux comme Rich OS.

— **Trusted Language Runtime (TLR)** : Ce système d'exploitation sécurisé [61] basé sur une plate-forme .Net repose sur un système de multiples *trustboxes*. Une *trustbox* est un environnement d'exécution protégeant la confidentialité et l'intégrité du code et des données. Chaque *trustbox* est comme un conteneur isolé des autres et peut faire l'objet d'une vérification distante à l'aide d'une paire de clés (privée/publique) initialisée sur chaque *trustbox*. Pour le moment, TLR n'est pas capable de communiquer de manière sécurisée avec les utilisateurs car les opérations d'entrée/sortie sont contrôlées par le Rich OS. Ceci implique qu'un attaquant a la possibilité d'intercepter les interactions de TLR avec l'utilisateur ce qui peut donner lieu à des fuites de données importantes.

— **OP-TEE** : OP-TEE [62] est un Secure OS développé conjointement par STMicroelectronics et Linaro [63] qui est compatible avec les standards TEE émis par GlobalPlatform [54]. OP-TEE peut être utilisé comme Secure OS sur un matériel spécialisé type ARM Juno ou comme TEE virtualisé à l'aide de l'hyperviseur FVP [2] décrit en section 5.2.1.1. Il fournit aux développeurs une bibliothèque cryptographique certifiée afin de pouvoir mettre en

1. Data Right Management
2. Fixed Virtual Platform

œuvre leurs Trustlets. Il est aussi doté d'un mode moniteur permettant le changement de contexte sécurisé entre les deux environnements. Il possède également la capacité de gérer le multi-coeurs avec la contrainte qu'un seul coeur à la fois peut être en exécution sécurisée. OP-TEE ne cesse de se développer, et se trouve depuis peu disponible sur des plate-formes embarquées légères et très peu chères telles que sur le Raspberry Pi 3, Alwinner A80 ou la HiKey board.

— **TrustKernel T6** : T6 [64] est un Secure OS pour les processeurs ARM avec la fonctionnalité TrustZone. Il peut exécuter de manière simultanée le Secure OS avec les multiples Rich OS disponibles (Android, Linux, etc.). L'équipe TrustKernel fournit le code source et le support nécessaire aux développeurs. Il est également fourni avec de multiples librairies telles que LibC et OpenSSL afin de faciliter le développement d'applications. Comme OP-TEE, T6 est compatible avec les standards de GlobalPlatform.

Au contraire des SE, nous assistons à une émergence de Secure OS open-source. Cela constitue une très bonne opportunité pour les développeurs d'intégrer cette technologie dans leurs processus de développement d'applications sécurisées.

3.4.4 Comparaison entre les différentes solutions matérielles

Afin de pouvoir établir une comparaison entre les solutions matérielles, nous avons sélectionné les critères suivants :

— ***Tamper resitance*** : Cette caractéristique détermine la résistance du composant de sécurité aux attaques physiques. En effet, même si le dispositif embarquant le composant de sécurité tombe aux mains d'un attaquant, il est nécessaire que l'attaquant ne puisse s'y introduire et en extraire des informations sensibles comme les clés de chiffrements par exemple.

— **Entrées/Sorties sécurisées** : Cette caractéristique détermine la capacité de ces composants à interagir de manière sécurisée avec l'utilisateur. La notion d'entrées sécurisées inclut la récupération des mots de passes de l'utilisateurs et toute sorte d'informations sensibles. La notion de sorties sécurisées peut être définie par le fait d'éviter qu'un malware affiche de fausses informations sur l'écran incitant l'utilisateur à effectuer une action permettant à l'attaquant de faire fuiter des informations.

— **Capacité élevée de traitement** : Cette caractéristique dénote la capacité d'un composant à effectuer des opérations complexes dans un temps réduit. Dans le contexte de la sécurité, ça se traduit par le fait de pouvoir générer les clés de chiffrements, exécuter les opérations de chiffrement/déchiffrement, signature et authentification de manière rapide.

— **Capacité élevée de stockage** : Cette caractéristique représente la capacité d'un composant à stocker de large quantité de données de manière sécurisée. Cette capacité peut s'avérer très utile au moment de stocker un annuaire de clés ou de faire, par exemple, la signature d'un large fichier.

— **Dépendance au fabriquant** : Cette caractéristique détermine si un composant est lié à

son constructeur ou pas. Ce critère est très important dans le choix de la technologie à déployer. En effet, si un composant est verrouillé par le constructeur, il est nécessaire d'obtenir l'accord de ce dernier pour déployer toute application sécurisée sur le composant. Cette restriction se caractérise par la non-signature d'applications tierces par le fabriquant ce qui empêche les développeurs d'utiliser ces éléments de sécurité dans les solutions qu'ils conçoivent.

— **Niveau de sécurité prouvé** : Cette caractéristique permet de classer les composants de sécurité par niveau de sécurité. En effet, la majorité des constructeurs de ces composants leur font passer une évaluation de sécurité (critères communs [41] par exemple). Cette évaluation permet de déterminer à quel genre d'attaque peut résister le composant et aussi de hiérarchiser les attaques qu'il peut subir.

Le tableau 3.1 montre les principales différences entre les SE, les TEE et les TPM selon les critères énoncés ci-dessus.

Tableau 3.1 – Comparaison entre les solutions matérielles

Criterias	SE	TEE	TPM
Tamper Resistance	✓		✓
Entrées/sorties sécurisées		✓	
Capacité de traitement élevée		✓	✓
Capacité de stockage élevée		✓	
Dépendance au fabriquant	✓	✓	✓
Niveau de sécurité prouvé	✓	✓	✓

Si nous sortons du monde du mobile, il existe d'autres solutions de stockage et d'exécution sécurisées prometteuse. Intel SGX s'inscrit dans ce panel de solutions. Elle est pour le moment présente sur les dernières générations de processeurs pour ordinateurs et serveurs Intel. Cependant, nous ne pouvons exclure de les voir arriver sur les terminaux mobiles munis de processeurs Intel. Nous décrivons ce TEE dans la section qui suit.

3.4.5 Intel SGX

Intel SGX [65] [66] [67] est une extension de l'ensemble des instructions des processeurs Intel x86 qui permettent à une application d'instancier des conteneurs protégés, appelés *enclave*, contenant le code et les données. La zone mémoire réservée par l'enclave est protégée contre tous les accès logiciels externes y compris les processus privilégiés tels que les systèmes d'exploitation et les hyperviseurs. Pour supporter les enclaves, SGX met à disposition un ensemble d'instructions nouvelles ainsi que des méthodes d'accès mémoires différentes. Aussi, SGX supporte la possibilité de vérifier à distance l'intégrité d'une enclave mais aussi de sceller une

enclave permettant entre autres de stocker les données d'une enclave en mémoire non-volatile pour de futures utilisations.

3.4.5.1 Protection mémoire sur SGX

Quand un processeur accède aux données d'une enclave, le mode du processeur est automatiquement transféré en mode enclave. Ce mode met en œuvre des vérifications additionnelles sur chaque accès mémoire assurant que seul le code résidant dans une enclave soit habilité à accéder à cette zone mémoire. En d'autres termes, tout accès à la zone mémoire d'une enclave à partir d'un code résidant à l'extérieur de cette dernière est prohibé. Les accès mémoires aux zones non-enclavées sont gérés de manière ordinaire à l'aide des pages mémoire [68].

Les données d'une enclave sont stockées dans une zone mémoire réservée appelée Enclave Page Cache (EPC). SGX introduit un ensemble d'instructions et des structures de données pour supporter le mode enclave et les opérations liées à l'EPC. Les instructions pour SGX se divisent en deux parties : des instructions en mode utilisateur et des instructions en mode Kernel. Afin de mettre en place des mécanismes de protection pour cette zone mémoire contre des attaques connues telles que le snooping, le contenu mémoire de l'EPC est chiffré avec le Memory Encryption Engine (MEE). Ce contenu mémoire n'est déchiffré que lorsque le processeur bascule dans le mode enclave et sont re-chiffrés dès que le processeur s'apprête à quitter ce mode.

3.4.5.2 Cycles de vie d'une enclave

Au cours de sa vie, une enclave passe principalement par quatre états :

— **L'enclave n'existe pas encore** : A ce stade, l'enclave n'existe pas encore et doit être créée via l'instruction *ECREATE*. Cette dernière permet d'initialiser la structure de contrôle de l'enclave pour la future utilisation.
— **Enclave non-initialisée** : A ce stade du cycle de vie de l'enclave, cette dernière est créée mais ne dispose pas de mémoire pour pouvoir s'exécuter. Il est possible d'utiliser l'instruction *EADD* pour allouer la mémoire nécessaire au fonctionnement de l'enclave. L'instruction *EINIT* permet d'initialiser l'enclave.
— **Enclave initialisée** : A ce stade du cycle de vie, l'enclave est créée et dispose de la mémoire nécessaire à son exécution. Elle peut désormais invoquer les opérations à exécuter. L'instruction *EREMOVE* permet de détruire une enclave.
— **Enclave en cours d'utilisation** : La commande *EENTER* ou *ERESUME* permettent au processeur de passer en mode enclave afin d'exécuter le code défini. La bascule vers l'état précédent se fait à l'aide de la commande *EEXIT*.

La Figure 3.7 permet de visualiser les différentes phases du cycle de vie d'une enclave.

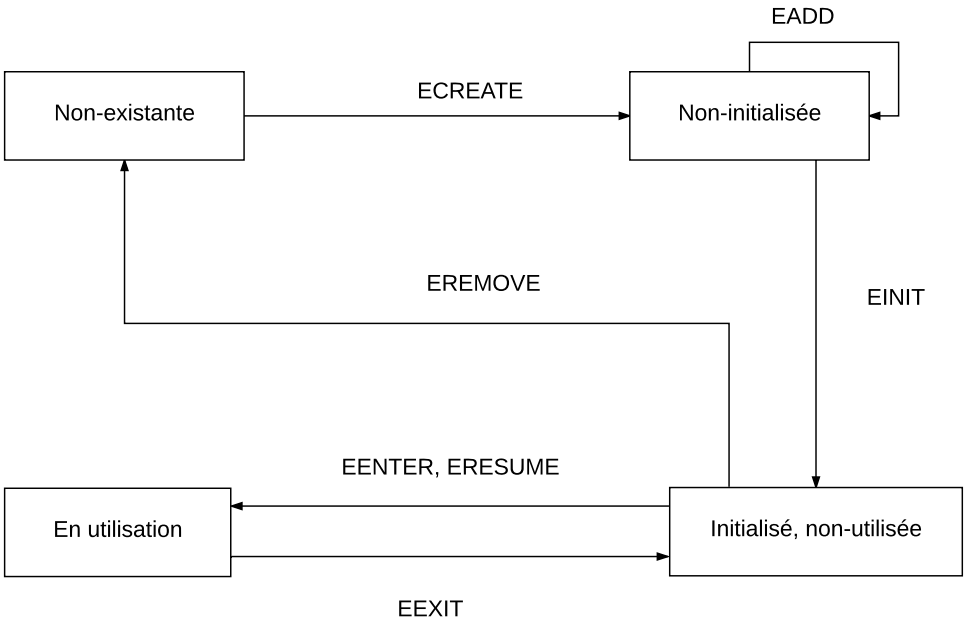

FIGURE 3.7 – Cycle de vie d'une enclave SGX

3.4.5.3 Performances et allocation mémoire supplémentaires

Pour un processeur Intel Skylake, la taille de l'EPC se situe entre 64 et 128Mo. Pour supporter des enclaves nécessitant plus de mémoire, SGX fournit des mécanismes de pagination mémoire permettant à l'EPC d'échanger des pages mémoires avec la mémoire non-fiable RAM. Le système utilise des instructions privilégiées afin de copier une page mémoire dans un buffer chiffré en RAM en dehors de l'EPC. Avant de réutiliser la page mémoire EPC utilisée, le système doit exécuter des opérations d'effacement de la mémoire.

SGX induit un impact sur les performances du processeur lors de l'exécution d'une enclave. Comme les instructions privilégiées ne peuvent être exécutées qu'à l'intérieur d'une enclave, il est nécessaire de quitter le mode enclave afin de traiter les appels systèmes privilégiés. Il est évident que ces changements de contextes récurrents induisent une charge supplémentaire due aux vérifications de sécurité, chiffrement de la zone mémoire et les opérations relatives à la protection des enclaves. Une autre limitation en terme de performances réside dans l'échange de pages mémoires pour les applications nécessitant de la mémoire supplémentaire. Ce processus est lourd pour le système et comporte des opérations très coûteuses en terme de performances.

3.5 Solutions existantes basées sur des environnements sécurisés

Nous présentons dans cette section plusieurs solutions recensées dans la littérature. La première solution abordée, On boad Credentials (ObC), est un TEE open source permettant le développement libre d'applications sécurisées. La seconde, Secure key storage on Android résout le problème de stockage sécurisé de clés de chiffrement sur Android. Nous présentons ensuite la solution de stockage sécurisé de Samsung : le Samsung KNOX. Ces solutions sont basées sur des TEE pour garantir la sécurité des données d'authentification sur deux OS : Windows 8 et Android respectivement. La troisième solution matérielle exposée est un cloud de TPMs illustrant les concepts de la section 3.4.2. Nous présentons ensuite une solution logicielle, Cells, qui est l'implémentation d'un hyperviseur type 2 basée sur les recommandations de VMware exposées en section 3.3.2. Pour finir, nous étudions deux exemples de TEE open source offrant la possibilité de virtualiser la plate-forme.

3.5.1 On board Credentials (ObC)

ObC [69] est un projet développé par l'équipe de recherche du constructeur Nokia visant à simplifier l'utilisation d'un TEE physique avec l'introduction d'un nouveau système d'approvisionnement d'applications sécurisées. Avec ObC, un développeur d'application pourra se contenter d'avoir l'accord de l'utilisateur du smartphone afin d'installer son application sécurisée sur le TEE contrairement au schéma de base qui requiert l'accord du constructeur du TEE ainsi que le fournisseur de l'OS sécurisé. De plus, une API est définie permettant le développement simplifié d'applications des deux côtés : REE et TEE. Pour ce faire, l'architecture ObC est divisée en deux composants : l'interpréteur ObC et l'ordonnanceur ObC.

— **Interpréteur ObC** : C'est une petite machine virtuelle qui procure un environnement d'exécution pour les applications sécurisées qui peut être vue comme un Secure OS léger. Cet interpréteur peut s'exécuter comme un OS sécurisé sur un TEE physique ou comme une application sécurisée tournant sur un Secure OS existant. Dans ce dernier cas, l'interpréteur enrichi les caractéristiques du Secure OS qui l'héberge avec des possibilités supplémentaires d'approvisionnement sécurisé en applications et en améliorant l'isolation entre les applications. En contre-partie de ces améliorations, nous pouvons noter que ces nouvelles caractéristiques rallongent le code du Secure OS et donc augmentent la surface d'attaque.

— **L'ordonnanceur ObC** : Cet ordonnanceur s'exécute sur la partie REE (l'OS de base du smartphone). Il a pour but de gérer le cycle de vie de l'application sécurisée et d'interagir avec l'interpréteur pour effectuer ce travail. Plus précisément, l'ordonnanceur est en charge de fournir à l'interpréteur les applications sécurisées, leurs données et l'état de leur exécution. Il est nécessaire de noter que les applications sécurisées gérées par

l'ordonnanceur sont toujours chiffrées. Enfin, cet ordonnanceur peut être utilisé pour accéder aux capacités cryptographiques offertes par le TEE via l'interpréteur.

ObC peut être vu comme une approche intéressante dans le processus de libération des implémentations de TEE pour les développeurs. Cependant, cette implémentation est spécifique aux smartphones de Nokia ce qui limite son adoption à grande échelle. Une autre limitation est ses performances réduites dues aux communications entre l'interpréteur et l'ordonnanceur. En effet, les multiples changements de contexte entre le REE et le TEE et le chiffrement/déchiffrement des applications sécurisées ont un impact important sur les performances.

3.5.2 Secure key storage on Android

Coojimans et al. dans [38] étudient la problématique du stockage sécurisé des clés de chiffrements sur Android. Les auteurs ont exposé plusieurs scénarios d'attaques et évalué les solutions existantes de stockage de clés de chiffrements sur Android. Dans un premier temps, ils ont introduit une solution purement logicielle basée sur l'utilisation d'une bibliothèque présente depuis l'API 1 d'Android : Bouncy Castle [70]. Dans un second temps, ils ont abordé la solution KeyStore présente sur Android à partir de l'API 18 (Android 4.3). Les auteurs ont testé la solution KeyStore avec deux processeurs dotés de capacités TEE : un processeur Qualcomm et un Texas Instrument MShield [56].

Leurs expérimentations montrent qu'aucune solution ne permet d'avoir une sécurité totale pour le stockage des clés de chiffrement. Néanmoins, les auteurs ont constaté que les solutions basées sur l'utilisation des TEE sont plus sûres et procurent une meilleure protection contre les attaques considérées.

Pour conclure, Coojimans et al. émettent plusieurs recommandations afin d'améliorer le niveau de sécurité du KeyStore. Ces recommandations peuvent être résumées dans les points suivants :

— Chiffrer tous les fichiers du KeyStore avec une clé générée dans le TEE ou fournie par l'utilisateur.

— Inclure l'identité de l'utilisateur de l'application qui a généré la paire de clés dans la vérification de l'intégrité du KeyStore.

Ces recommandations ne nécessitent aucun changement dans les architectures ou les systèmes actuels.

3.5.3 Samsung KNOX

KNOX [71] est une solution propriétaire permettant le stockage et l'exécution sécurisés sur un smartphone Android de la marque Samsung. Cette dernière a introduit KNOX au début de l'année 2013 avec la version 1.0.0. Cette version a été déployée avec Android 4.3 sur ses smartphones Galaxy S3 et Galaxy S4. Plusieurs versions ont depuis vu le jour telles que la 2.3 déployée sur le Galaxy Note 3 et supérieur.

KNOX fournit un environnement sécurisé parallèlement à l'environnement normal du smartphone. Il permet d'exécuter des applications sécurisées ou sensibles dans un environnement sécurisé isolé. Les composants permettant de construire un tel environnement sont : la séquence de Secure Boot et l'utilisation d'un TEE (ARM TrustZone) augmenté d'un SELinux sur la partie Android.

KNOX est un second environnement Android sécurisé au sein du système d'exploitation lui-même. Le sous-système d'exploitation sécurisé est doté de sa propre page d'accueil, de son propre système de démarrage, de ses applications et de ses widgets. KNOX permet d'avoir une isolation entre ses propres applications et une isolation entre ses applications et les applications Android classiques. Au delà de l'isolation, tous les fichiers de KNOX sont chiffrés en utilisant l'algorithme de chiffrement AES (Advanced Encryption Standard) avec une clé de 256 bits.

3.5.4 Cells : Une architecture virtuelle pour le smartphone

Cells, proposée par Andrus et al. dans [72], est une plate-forme virtualisée qui permet d'avoir plusieurs instances virtuelles de smartphones Android sur le même terminal. Cells garantit une isolation entre les smartphones physique et virtuels et entre les terminaux virtuels eux-mêmes. Il procure un niveau de performances très satisfaisant proche du natif.

Cette solution ne permet pas de lancer plusieurs instances d'Android. En effet, Cells exécute plusieurs instances de téléphones virtuels sur une seule instance de l'OS. Cette approche favorise le partage des parties communes en lecture seule du code et des données. De plus, la mémoire utilisée dans cette solution est minimisée et par conséquent l'impact de la solution sur la mémoire physique du smartphone est très faible d'après ses concepteurs.

Afin de fournir une ligne téléphonique spécifique à chaque téléphone virtuel, Cells utilise la voix sur IP (VoIP) pour contourner le problème lié à la présence d'une carte SIM sur la majorité des téléphones. Les appels entrants et sortants utilisent le réseau téléphonique de la carte SIM et sont routés à travers le service de VoIP à chaque instance de téléphone virtuel en utilisant un Caller ID. Cells utilise cette combinaison d'un serveur de VoIP et du réseau téléphonique de la carte SIM afin de permettre aux utilisateurs d'émettre ou de recevoir des appels en utilisant l'opérateur de leur carte SIM. En effet, le service de VoIP agit comme un proxy entre l'identité du communicant et le réseau cellulaire.

Andrus et al. revendiquent l'exécution de cinq téléphones virtuels sur un smartphone physique avec un impact minimal sur les performances natives du smartphone. De plus, les auteurs précisent que cette solution permet aussi le partage de périphériques matériels tels que la caméra, les capteurs et l'écran tactile entre l'hôte et l'invité de manière transparente à l'utilisateur.

Lorsqu'un utilisateur a besoin de créer une instance virtualisée d'un téléphone, il commence d'abord par la créer sur un ordinateur. Ensuite, cette instance est transférée sur le smartphone physique via la la liaison USB. Dès lors, l'OS du smartphone cohabite avec un OS invité sur le même matériel.

Les résultats des expérimentations menées par les auteurs montrent un minimum d'impact

sur les performances lors du déploiement de plusieurs instances de téléphones virtuels sur les deux smartphones considérés : Nexus 1 et Nexus S. Cette solution est prometteuse dans le déploiement de téléphones (professionnels ou personnels) tout en garantissant la sécurité du stockage et du traitement des données.

3.5.5 Une architecture cloud pour des TPM virtuels

Liu et al. dans [73] proposent un cloud de TPM utilisable par un terminal distant. Cette solution consiste à avoir un cluster de TPM physiques accessible via le réseau sur un terminal qui ne possède pas de TPM physique. Chaque service est représenté par un numéro de port virtuel sur le cloud. Un protocole basé sur les requêtes/réponses est déployé entre l'utilisateur et le cloud de TPM.

Un canal de communication sécurisé est également nécessaire afin d'échanger les données sensibles entre l'utilisateur et le cloud. Chaque utilisateur se voit doté d'une paire unique de clés privée/publique. La Figure 3.8 représente le schéma de communication expliquant le mécanisme de requête/réponse sécurisé.

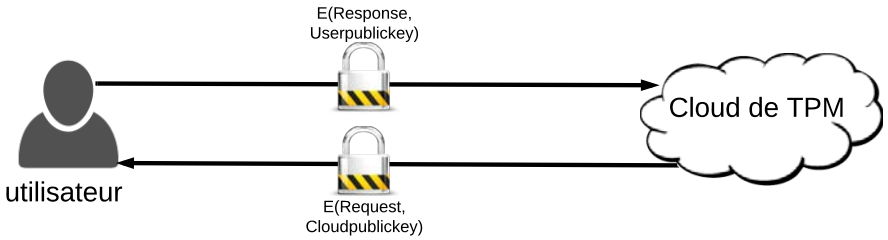

FIGURE 3.8 – Cloud de TPMs

L'utilisateur construit une requête pour le cloud qui représente une concaténation des opérations à exécuter et de la clé publique de l'utilisateur. La requête est chiffrée avec la clé publique du cloud avant d'être envoyée sur le réseau. Dès la réception et le déchiffrement de la requête, le cloud traite les opérations de l'utilisateur et génère les réponses souhaitées. Ces résultats sont compilés dans un paquet réponse et chiffrés en utilisant la clé publique reçue dans la requête avant d'être envoyés à l'utilisateur. Selon les auteurs, cette solution présente des performances telles, qu'elle donne à l'utilisateur l'impression d'utiliser un TPM physique.

Cette solution est similaire à celle proposée par Urien dans [74] pour un cloud de Secure Elements. Cependant, ces solutions ne sont possibles que dans le contexte connecté où le smartphone est capable de communiquer avec un cloud. Néanmoins, cette connexion n'est pas systématique et peut être coûteuse en terme de consommation d'énergie. Une autre limitation pointée par Le Vinh et al. dans [43] est que le smartphone et le cloud doivent être sécurisés. De plus, si nous intégrons le Cloud de TPM dans un protocole d'authentification complexe, les temps de transmissions peuvent avoir un impact sur les performances du système.

3.5.6 OPEN-TEE : Un TEE virtualisé et open source

OPEN-TEE [75], est un TEE entièrement virtualisé respectant les spécifications de GlobalPlatform. Il permet le développement d'applications sécurisées pour les TEE afin de les tester sur une plate-forme purement logicielle sans les contraintes d'accès à un TEE physique. L'objectif des auteurs était d'élaborer un SDK et un framework permettant le développement et le test d'applications sécurisées. Ce SDK doit demander une configuration minimale ainsi que le moins de maintenance possible.

Les auteurs ont choisi de diviser leur framework en un ensemble de composants dont les principaux sont :

— **Base** : C'est un démon qui s'exécute en mode utilisateur sur l'OS. Il est responsable de la configuration et de la préparation de l'environnement.

— **Manager** : Il peut être considéré comme le Trusted OS d'OPEN-TEE. Il est notamment chargé de gérer les connexions entre les applications, de surveiller l'état des différentes applications et de partager la mémoire entre les applications afin de fournir des zones de stockage sécurisées pour chaque application.

Launcher : Ce composant est chargé de la création et du lancement des applications sécurisées.

D'autres composants sont utilisés parmi lesquels : les API GlobalPlatform et les communications IPC (InterProcess Communications) qui sont notamment utilisées pour la transmission des messages entre la Client Application (CA) et la Trusted Application (TA).

Le SDK et le framework OPEN-TEE sont actuellement en test par plusieurs organisations afin de valider les résultats obtenus par l'équipe de développement.

3.5.7 Virtualisation des TEE

Vahidi et al. dans [76] proposent de concevoir et de mettre en place un hyperviseur pour la plate-forme NovaThor U8500 qui opère dans la partie TEE du processeur ARM Cortex-A9. L'objectif principal de cette proposition est de garantir l'isolation entre les trustlets elles-mêmes et entre les trustlets et les applications du Rich OS (Linux ou Android).

Cette solution a pour but de concevoir et d'implémenter un hyperviseur opérant dans le Secure OS. L'hyperviseur se charge de virtualiser la plate-forme sous-jacente de manière à ce que le TEE puisse assurer une isolation entre les Trusted Applications elles même et entre une TA et une Application du Rich OS. Pour leur plate-forme, les auteurs ont choisi d'implémenter un hyperviseur de type 1 avec un code court ayant une faible complexité et procurant des performances proches du natif.

La Figure 3.9 présente l'architecture choisie par les auteurs et identifie les flux de données entre les différents composants. Le flux étiqueté (1) correspond à l'invocation par un client d'une fonction d'API sur le TEE. Le flux étiqueté (2) représente un appel à partir du Secure OS du TEE vers une application sécurisée effectuant un traitement spécifique (chiffrement, hashage,

signature ...). Le flux (3) est la communication entre le Secure OS et l'hyperviseur qui permet l'utilisation des composants matériels si nécessaire.

Prenons l'exemple d'une application sécurisée qui fait du hashage d'un message m. Le flux étiqueté (1) correspondrait pour cette application, à l'invocation de la fonction de hashage présente sur la partie TEE pour exécuter l'action requise. Une fois les données transmises vers le Secure OS, ce dernier invoque la Trusted Application spécifique visée et lui transmet les données à hasher pour notre exemple (flux étiqueté (2)). Au cas où l'application ait besoin de communiquer avec l'utilisateur à travers l'écran tactile, pour y afficher le résultat du hash par exemple, l'hyperviseur est sollicité pour faire de l'abstraction du matériel. Cette étape correspond au flux étiqueté (3).

La sécurité de cette solution repose principalement sur la sécurité de l'hyperviseur. Les auteurs prévoient de vérifier formellement la sécurité de leur hyperviseur afin de prouver son niveau de sécurité.

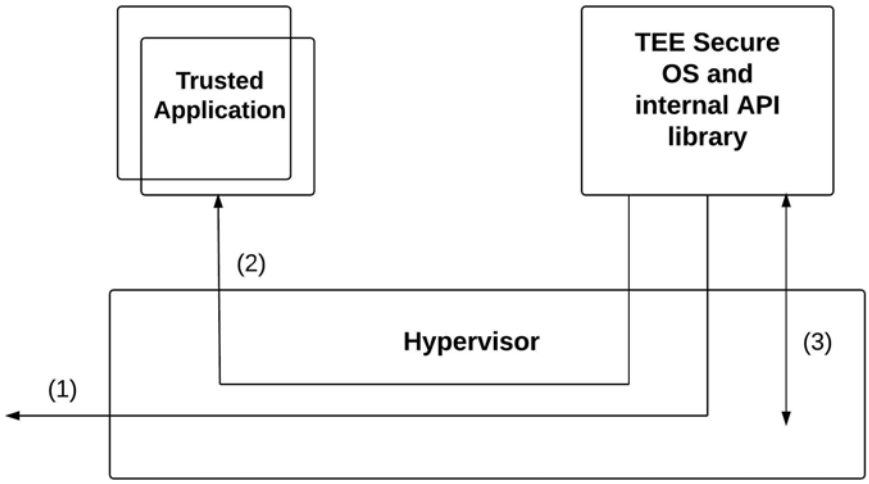

FIGURE 3.9 – Architecture de virtualisation TEE et ses flux de données

3.5.8 Trusty TEE

Trusty [77] TEE est un projet de Google visant à déployer un TEE sur Android. Il comprend un système d'exploitation mobile sécurisé (Trusty OS) et un ensemble de bibliothèques et d'API pour définir les communications entre Applications exécutées sous Trusty OS et Android. Trusty TEE est également basé sur la technologie ARM TrustZone [78]. L'introduction de TrustyTEE dans l'écosystème Android est très récente, et les mises à jour très régulières.

Le déploiement d'applications tierces n'est pas possible actuellement sur la version de Trusty. Google attire l'attention des développeurs aux fait que les Trusted Applications ont accès à tous

les secrets du smartphone et qu'il est nécessaire de mettre en place un mécanisme de signature des applications afin de réguler le déploiement de telles applications sur un smartphone.

3.5.9 Trustonic TEE

Le TEE développé par Trustonic [79] est la première solution de TEE commerciale avec des licences payantes. Le Trustonic TEE est une fusion entre les technologies fournies par ARM, Giesecke et Devrient (G et D) et Gemalto. Trustonic est soutenu de manière significative par ARM TrustZone qui fournit les moyens matériels nécessaires dans le processus d'instanciation du Trustonic. G et D a fourni le système d'exploitation mobile sécurisé Mobicore. Le Trustonic TEE est compatible avec les standards de GlobalPlatform.

3.6 Comparaison entre les environnements sécurisés sur mobile

Notre étude des environnements de stockage et de traitement sécurisés sur les plate-formes mobiles nous a permis de les synthétiser par le Tableau 3.2. Tout d'abord, nous notons que les solutions matérielles ne sont sensibles ni aux attaques physiques ni aux attaques logicielles contrairement à la virtualisation. Nous remarquons aussi que les technologies présentées hormis la virtualisation dépendent des fabricants et donc nécessitent la négociation d'accords préalables au déploiement d'applications sécurisées. Cette contrainte freine le développement de solutions basées sur ces technologies.

Tableau 3.2 – Comparaison entre les environnements sécurisés sur mobile

Solution	Type	Resistance Att. Phy	Resitance Att. Log	impact perf	Capacité trait.	Capacité stock.
Secure Elements	Puce physique	Non	Non	0	Limitée	Limitée
TEE	physique et logique	Oui	Oui	0	Élevée	Élevée
TPM	Puce physique	Non	Non	0	Faible	Faible
Virtualisation	Purement logicielle	Oui	Oui	Minimal	Élevée	Élevée

Dans un second temps, nous observons que l'impact sur les performances constitue un critère intéressant de classement de ces solutions. Les solutions matérielles (SE, TEE, TPM) n'ont aucun impact sur les performances du système car elles disposent d'une capacité de stockage et de traitement indépendante de celle du smartphone. Par contre, la virtualisation repose sur l'exécution de plusieurs instances du smartphone sur le même terminal. Même si l'impact induit par cette solution reste minimal, il n'est pas à écarter lors de l'élaboration de solution temps réel.

Un autre critère important est la capacité de stockage et de traitement. Pour le SE et le TPM, cette capacité est limitée car ces cartes à puces ne sont pas conçues dans l'optique de traiter et de stocker de larges quantités de données. Pour les TEE et la virtualisation, leurs architectures leur permettent de tirer la pleine puissance du smartphone.

La virtualisation est un exemple intéressant d'environnement sécurisé sur mobile avec une grande capacité de stockage et de traitement ainsi qu'un niveau de sécurité particulièrement élevé. De plus, les capacités multi-cœurs des processeurs actuels permettent de limiter l'impact du processus de virtualisation sur les performances. Cependant, au moment de rédiger cette thèse, il nous apparaît clairement que la virtualisation n'est pas assez 'mature' pour être déployée à grande échelle dans des solutions commerciales de stockage et de traitement sécurisés.

3.7 Conclusion

Après avoir introduit les principales caractéristiques d'un environnement de traitement et de stockage sécurisé, nous avons présenté deux types de solutions : les solutions logicielles et les solutions matérielles. La virtualisation permet d'exécuter des instances de téléphones virtuels tout en assurant l'isolation entre ces instances. La pièce majeure permettant une telle propriété est l'hyperviseur. Nous avons discuté les deux types de virtualisation type 1 et type 2 en mettant en exergue les particularité de chacune des solutions. Cependant, cette solution n'a pas été massivement adoptée par la communauté. Ce manque de popularité est dû nécessairement à deux facteurs : l'impact sur les performances et la non-résistances aux attaques physiques.

Nous avons dès lors présenté des solutions matérielles. Ces solutions se basant sur des composants distincts du téléphone en lui même, permettent de fait de n'induire aucun impact sur les performances du smartphone. Ce type de solutions permet aussi de résister, pour la majorité d'entre elles, aux attaques physiques lorsqu'un attaquant parvient à intervenir physiquement sur le téléphone. Nous avons présenté les différents types de Secure Elements et nous avons proposé une analyse comparative afin de pouvoir mettre en évidence les forces et les faiblesse de chaque type. Nous sommes passé ensuite à la présentation des TPM qui sont en réalité des cartes à puces avec des capacités cryptographiques accrues. Le principal inconvénient des TPM est la taille réduite du stockage ne permettant pas de traiter de larges quantité de données. Les Trusted Execution Environment apparaissent comme la solution la plus prometteuse permettant de fournir un environnement de stockage et d'exécution sécurisés avec des performances élevées et un bon niveau de sécurité. Une analyse comparative des solutions matérielle permet de mettre en lumière les différences majeures entre ces composants.

Nous avons présenté différentes solutions de stockage et d'exécution sécurisés dans la littérature basées sur les technologies sus-citées. Nous allons dans la suite de ce manuscrit vous exposer la manière dont nous utilisons ces technologies pour concevoir et mettre en place un système de contrôle d'accès sécurisé sur un smartphone.

4 Dématérialisation sécurisée des cartes sans contact

4.1 Introduction

Afin de mettre en place un système de contrôle d'accès fiable corrigeant les vulnérabilités des cartes NFC actuelles, il est nécessaire de recourir à la dématérialisation de ces cartes sur des supports offrant une puissance de calcul et une capacité de stockage suffisantes pour le déploiement d'algorithmes d'authentification plus fiables. Le smartphone apparaît comme le candidat idéal pour accueillir l'application de contrôle d'accès dématérialisée. Cependant, le stockage de données sensibles sur un smartphone est une problématique en soi. En effet, les applications Android disposant d'autorisations suffisantes et de privilèges élevés sont à même de voler ces données à différents endroits : sur la mémoire de stockage ou sur la mémoire volatile en cours d'exécution. Ces vulnérabilités présentes sur Android ne nous permettent pas de déployer d'applications sécurisées de contrôles d'accès sous la forme d'applications simples. Comme abordé dans le chapitre précédent, il existe plusieurs solutions de stockage et de traitement sécurisés sur les smartphones Android. La solution la plus prometteuse est le TEE. Nous allons dans ce chapitre vous exposer notre architecture de dématérialisation sécurisée en détaillant les différents scenarii retenus.

4.2 Qu'est ce que la dématérialisation de cartes sans contact ?

La dématérialisation de cartes sans contact consiste à transférer les données d'authentification initialement stockées sur une carte sans contact sur un smartphone muni de la technologie NFC, qui va agir, au travers d'une application mobile, comme une carte traditionnelle. De plus, la dématérialisation vise à offrir un niveau de sécurité plus élevé ainsi qu'un panel de fonctionnalités plus élargi que celui offert par les cartes traditionnelles . Par exemple, il serait possible d'héberger sa carte bancaire, sa carte de transport en commun et sa carte d'accès sur le même terminal. Dans le cadre de cette thèse, notre cas d'utilisation est le contrôle d'accès aux bâtiments sensibles de l'Université Paul Sabatier à Toulouse dans le cadre du projet neOCampus. Ce projet vise la mise en place d'un campus durable et intelligent. Dès lors, l'architecture de

dématérialisation de base est représentée par la Figure 4.1.

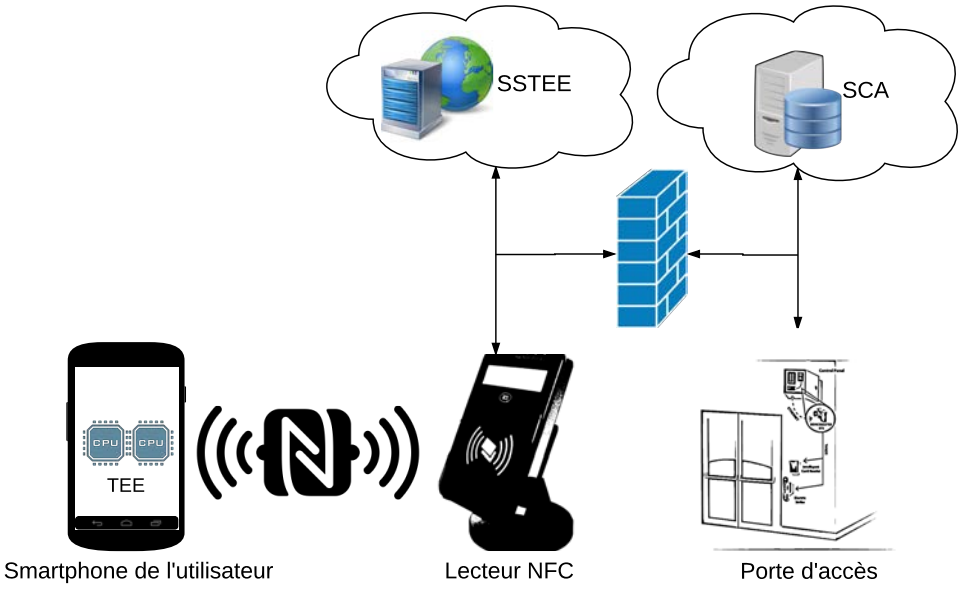

FIGURE 4.1 – Architecture de base d'une dématérialisation de cartes sans contact

Nous constatons qu'avec la dématérialisation, le téléphone agit comme une carte traditionnelle et que l'implémentation de cette dématérialisation ne nécessite, pour le scénario nominal, aucune modification dans l'architecture existante des systèmes de contrôle d'accès. Cependant, il est nécessaire de protéger les données d'authentification et l'exécution de l'application de contrôle d'accès sur le smartphone. Android n'est pas en mesure aujourd'hui de nous fournir cette protection de manière native. Nous avons donc choisi d'utiliser un TEE pour la sécurisation de l'application de contrôle d'accès. Cependant, la majorité des smartphones n'est pas muni de TEE. Cette restriction nous a amené à repenser l'architecture de base présentée en Figure 4.1 et de la faire évoluer pour garder un niveau de sécurité élevé tout en s'affranchissant des accords avec les constructeurs. L'architecture développée est présentée dans la section qui suit.

4.3 Architecture de dématérialisation de cartes sans contact

Afin de mettre en place un système de contrôle d'accès fiable sur le campus de l'université Paul Sabatier nous proposons l'architecture présentée en Figure 4.2. Notre architecture est composée du smartphone d'un utilisateur muni de la technologie NFC et d'un lecteur mural NFC pour chaque porte à sécuriser ainsi que d'un TEE déporté sur un Cloud pour le Smartphone (CSTEE) et le lecteur (Serveur Sécurisé de TEE ou SSTEE). Un Serveur de Contrôle d'Accès (SCA) est intégré au système pour gérer la base de données des personnes habilitées. Il est à noter que ce serveur est isolé et qu'il n'est pas accessible depuis l'extérieur du fait d'une isolation

physique (au travers d'un firewall comme illustré sur la Figure 4.2).

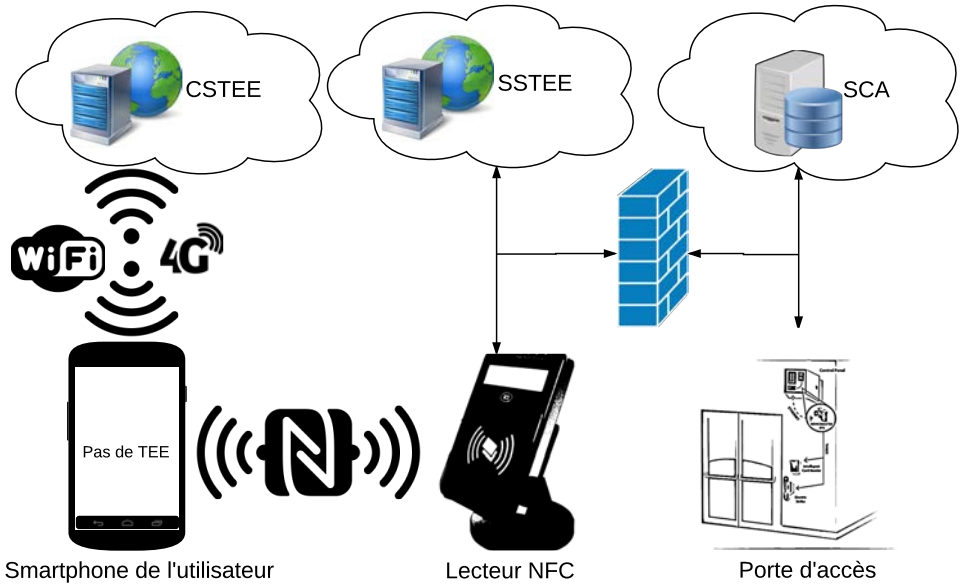

FIGURE 4.2 – Architecture proposée de dématérialisation de cartes sans contact

4.3.1 Éléments de l'architecture

4.3.1.1 Le smartphone

Le smartphone utilisé dans notre architecture doit nécessairement être muni de la technologie NFC pour initier le processus d'authentification en tapant sur le lecteur[1]. Dans ce scénario, nous supposons que le téléphone ne dispose pas d'un TEE intégré ou qu'il n'est pas possible de l'utiliser faute d'accord avec le fabriquant. Le smartphone accède à un TEE déporté sur un Cloud sécurisé CSTEE qu'il doit pouvoir contacter pour dérouler le protocole d'authentification. Pour ce faire, il est nécessaire que le smartphone ait un accès réseau à ce serveur soit via une connexion 4G ou une connexion Wi-Fi. Il est aussi indispensable de disposer d'une double authentification entre le smartphone et le CSTEE. Le smartphone sert de relai aux messages envoyés depuis le lecteur NFC.

4.3.1.2 Lecteur NFC

Il s'agit d'un lecteur NFC mural classique installé pour contrôler l'accès à chaque porte. C'est un dispositif très peu cher qui ne présente aucune capacité de stockage ou de traitement sécurisé. Il est installé dans les infrastructures actuelles de contrôle d'accès basées sur l'utilisation de cartes

1. Taper un lecteur NFC veut dire approcher le téléphone de ce dernier pour qu'il puisse être alimenté par son champ électromagnétique

sans contact. Il est indispensable qu'il dispose d'une connexion directe avec le SSTEE contenant le TEE déporté. Nous supposons que cette connexion Ethernet fait partie de l'infrastructure et nous la considérons comme sûre. Cette caractéristique n'induit aucun changement dans l'infrastructure actuelle qui est déjà câblée de cette manière pour relier le lecteur au serveur de contrôle d'accès. Il est possible de remplacer ce lecteur avec du matériel peu onéreux et performant comme les Rasberry Pi. Il est nécessaire de rajouter au RPi le support de la technologie NFC via des cartes d'extensions comme celle fournie par NXP. Ce changement permet de rajouter des fonctionnalités supplémentaires telles qu'un écran LCD pour saisir un code PIN ou un dispositif de recueil d'informations biométriques.

4.3.1.3 Serveur sécurisé côté smartphone (CSTEE)

Ce serveur consiste en un ou plusieurs TEE, physiques ou virtualisés, accessibles via Internet. C'est la pièce maîtresse de cette architecture car il est responsable de l'exécution des opérations sensibles composant le protocole d'authentification telles que le chiffrement, le déchiffrement, le hashage et la signature côté smartphone. L'utilisateur peut faire valoir son droit de disposer de son propre serveur privé contenant ses données d'authentification et peut être aussi ses données biométriques.

4.3.1.4 Serveur sécurisé côté lecteur (SSTEE)

Ce serveur possède les mêmes fonctionnalités que le serveur côté client. Il exécute l'algorithme d'authentification de manière sécurisée afin de permettre l'authentification du smartphone. Il est intégré à l'architecture de contrôle d'accès actuelle. En effet, son déploiement ne nécessite pas de changement physique sur l'architecture de contrôle d'accès présente sur le campus de l'université.

4.3.1.5 Serveur de contrôle d'accès (SCA)

Ce serveur est en charge de vérifier dans sa base de données, si la personne authentifiée au travers du protocole est réellement autorisée à accéder à la ressource. Le SCA fait partie des systèmes de contrôle d'accès actuels déployés dans l'université. Nous supposons que ce serveur n'est pas accessible depuis l'extérieur et est physiquement câblé au lecteur NFC et au serveur sécurisé.

4.3.2 Hypothèses et identification des menaces

Dans l'architecture proposée, toutes les parties impliquées sont supposées bienveillantes et honnêtes et ne s'écarteront pas du protocole proposé. De plus, les smartphones des utilisateurs, les lecteurs et les serveurs (CSTEE et SSTEE) n'échangent que des messages valides. Les serveurs (CSTEE et SSTEE) sont supposés sécurisés et aucune information ne peut filtrer d'un utilisateur à l'autre. La sécurité des serveurs dans le Cloud est en dehors du champ de cette thèse. Nous

supposons aussi que les communications entre le lecteur et le serveur sécurisé sont sûres et qu'un attaquant ne peut accéder au réseau filaire déployé.

Les smartphones sont supposés dotés de la technologie NFC en mode émulation de cartes et peuvent agir comme une carte sans contact traditionnelle. Aucune autre hypothèse n'est faite sur les liens de communication.

Nous pouvons identifier plusieurs menaces sur l'architecture. Tout d'abord, l'attaque classique consiste à écouter les communications afin d'en extraire les données d'authentification des utilisateurs. Afin d'éviter cette attaque, chaque communication doit être chiffrée. De plus, il est nécessaire d'introduire des mécanismes pour lutter contre le rejeu de transactions. L'utilisation de numéros de séquences pour chaque paquet est une solution à ce problème. Dans un second temps, nous nous intéressons aux menaces visant le smartphone de l'utilisateur. En effet, ces terminaux sont vulnérables aux malwares, trojans qui peuvent intercepter les informations sensibles transitant au travers du terminal. Les smartphone *rootés* [1] ouvrent la porte à des attaques plus graves sur les données présentes sur le téléphone et les données qui y sont traitées. Par conséquent, aucune opération sensible ni données critiques ne doivent être exécutées ou stockées sur le terminal. **Nous ne considérons pas les smartphones comme des périphériques de confiance**.

Dans l'architecture proposée, toutes les opérations cryptographiques sont réalisées sur le TEE déporté sur un des deux serveurs CSTEE ou SSTEE. Le smartphone doit être authentifié au CSTEE avant de pouvoir s'authentifier à la plate-forme de contrôle d'accès. Cependant, comme toutes les opérations cryptographiques réalisées sur le smartphone peuvent être compromises, une connexion sécurisée type TLS [80] n'est pas suffisante. En effet, un attaquant ayant compromis le smartphone peut intercepter la négociation de la clé de session et peut dès lors déchiffrer toutes les communications entre le smartphone et le CSTEE. Plusieurs solutions ont été proposées pour remédier à cette problématique. Par exemple, Mulliner et al. dans [81] proposent d'augmenter le TLS existant avec un *One Time Password (OTP)* généré sur un serveur sécurisé et transmis à l'utilisateur via un autre canal (SMS par exemple). Cet OTP n'est valable que quelques minutes et doit permettre à l'utilisateur de s'authentifier en ne laissant pas le temps à un attaquant d'intercepter l'OTP et de le rejouer. Une autre solution consisterait à utiliser le Secure Element (SE) de la carte SIM pour y héberger une clé partagée avec le cloud qui ne sera utilisée que pour la première authentification [82].

Dans la suite de ce manuscrit, nous nous concentrerons sur la partie authentification de l'utilisateur au serveur de contrôle d'accès en prenant comme hypothèse qu'il est déjà connecté à son CSTEE. Cette hypothèse est réaliste car cette authentification avec le Cloud est effectuée avant le contrôle d'accès et n'est plus nécessaire tant que l'utilisateur possède toujours sa connectivité avec ce Cloud. Dans les sections suivantes, nous présentons les deux parties majeures de notre architecture de dématérialisation : le chiffrement IBE et le protocole d'authentification IBAKE [83].

1. rooter son téléphone signifie devenir l'administrateur du système et avoir tous les droits sur son téléphone Android

4.4 Identity Based Encryption (IBE)

Le chiffrement basé sur l'identité ou Identity Based Encryption (IBE) a été conçu pour résoudre certaines problématiques induites par l'utilisation du chiffrement asymétrique et l'échange de la clé publique. La solution classique consiste à utiliser une infrastructure de clés publiques ou Public Key Infrastructure (PKI) délivrant des certificats de confiance d'une source unique, connue sous le nom d'Autorité de Certification (AC) afin de lier l'identité d'une personne à une clé publique. Cependant, le processus d'enregistrement à une PKI est très fastidieux et nécessite de fournir des preuves de son identité avant tout enregistrement. Cette solution repose sur la confiance mutuelle accordée à l'Autorité de Certification par deux utilisateurs ou plus. Cependant, certaines problématiques inhérentes à l'utilisation de PKI ont vu le jour. La révocation [1] de clés fait partie des problèmes majeurs de l'utilisation de PKI notamment dans des environnements embarqués limités en terme de connectivité.

Afin de résoudre les problématiques des solutions basées sur les certificats, le mécanisme d'IBE utilise directement l'identité d'un utilisateur (adresse email par exemple) ou d'un périphérique comme clé publique de l'algorithme de chiffrement, signature etc. Shamir en 1984 dans [84] a été le premier à introduire cette notion. En effet, il a eu l'idée d'utiliser son adresse réseau comme clé publique et faisant générer sa clé privée par un tiers de confiance.

Dans le début des années 2000, Boneh et Franklin dans [85] ont présenté le premier schéma pratique et satisfaisant en terme de sécurité des IBE. Leur schéma est basé sur du pairing sur des courbes elliptiques avec des performances similaires à l'algorithme d'ELGammal [86]. Ils fournissent également une preuve de sécurité dans le modèle d'oracle aléatoire permettant leur utilisation dans des systèmes distribués sur Internet. Nous allons dans ce qui suit expliquer le concept des IBE.

4.4.1 IBE vs PKI traditionnelle

Le Tableau 4.1 permet de référencer les principales différences entre un système basé PKI et un système basé IBE.

Une infrastructure de clé publique (PKI) permet la génération de paires de clés et leur distribution à des clients enregistrés. Son principal rôle est de permettre aux utilisateurs et aux ordinateurs d'échanger des informations de manière sécurisée sur un réseau qui n'est pas sûr, par exemple Internet. Cette sécurité est dûe à la confiance accordée à l'Autorité de Certification et la stricte vérification des identités des détenteurs de clés publiques. Le mécanisme d'IBE permet d'utiliser l'identité de la personne comme partie publique de l'algorithme et de distribuer les clés privées à leurs possesseurs. Aucun mécanisme d'enregistrement n'est prévu.

La PKI peut générer la paire de clé de l'utilisateur. De fait, il est nécessaire pour une PKI de prévoir de grandes bases de données pour stocker ces clés. Pour ce qui est des IBE, ils utilisent

1. La révocation de clés ou d'un certificat consiste à annuler le certificat avant sa date d'expiration et de faire opposition de tout usage du certificat ou des clés associées

Tableau 4.1 – Comparaison entre une PKI et un IBE

Caractéristiques	PKI	IBE
Génération de la clé publique	Utilise des informations de l'utilisateur	Utilise l'identité de l'utilisateur
Génération de la clé privée	Générée par l'Autorité de Certification ou par l'utilisateur	Par le PKG
Distribution de la clé privée	Canal de communication sécurisé	Canal de communication sécurisé
Distribution de la clé publique	Annuaire public ou par certificat utilisateur	L'identité est la clé publique
Key Escrow	Non sauf si c'est l'Autorité de Certification qui génère la clé	Oui

une *master key* pour générer la clé privée de chaque utilisateur en s'appuyant sur son identité. Tous les utilisateurs connaissent la clé publique de leur correspondant (ou peuvent la générer suivant les schémas d'IBE voir section 4.4.5). Cette caractéristique apporte plus de flexibilité et ne nécessite ni génération ni stockage de certificats. De plus, dans le mécanisme d'IBE, le processus de gestion de l'identité est très important pour garantir le bon fonctionnement du système. Pour la PKI, l'obtention de la paire de clés est sujet à une démarche d'inscription et à un lourd processus de vérification de l'identité. L'utilisation des IBE apparaît comme la solution la plus souhaitable dans les petites organisations car elle offre plus de flexibilité qu'une solution PKI. En effet, dans le contexte de l'Université Paul Sabatier, les adresses email peuvent faire office d'identités utilisables dans un système IBE car la gestion des identités est déjà assurée par le service informatique de l'Université.

Dans le contexte de la PKI, l'utilisateur peut choisir de changer sa paire de clés à n'importe quel moment. Cependant pour les IBE la paire de clés est liée à l'identité, ce qui rend le processus de révocation délicat et complexe. Il existe des solutions permettant une telle révocation. Par exemple, concaténer une autre chaîne de caractères à son identité telle que la date du jour permettra de révoquer de manière automatique la clé à la fin de la journée.

La PKI n'est pas censée être un tiers de séquestre (*Key Escrow* [1]) au contraire des IBE où la clé privée est générée et stockée sur le générateur de clés privées Private Key Generator (PKG) en attendant les requête des clients ce qui amène au problème induit par le tiers de séquestre. En effet, en disposant de toutes les clés privées des utilisateurs, un PKG peut accéder à toutes les communications de ses utilisateurs et de potentiellement produire des documents ayant la signature d'un utilisateur sans qu'il le sache. Cet inconvénient n'est pas présent (en théorie) pour les PKI car l'utilisateur détient sa clé privée ce qui nous amène à avoir une propriété de non-répudiation forte. Cependant, dans certains pays, en France en particulier, la loi exige que les données chiffrées puissent être déchiffrées à la demande des autorités nationales. La mise en

1. Tiers ayant pour objectif de conserver (mise sous séquestre) les clés de chiffrement.

œuvre d'un séquestre répond à cette exigence.

Pour résumer, le mécanisme d'IBE est adapté pour un déploiement rapide dans les petites et moyennes structures qui ne nécessite pas beaucoup d'infrastructure, ce qui est un avantage, par rapport aux PKI tout en garantissant un niveau de sécurité similaire.

4.4.2 Les composants d'un système IBE

Chaque mécanisme d'IBE est basé sur les quatre composants suivants :

— **Setup** : Durant cette étape, l'algorithme génère les paramètres du système ainsi que la *master key* K_{PKG}. Cette dernière sera utilisée pour dériver les clés privées associées à chaque identité (clé publique).

— **Key extraction** : Cet algorithme permet de dériver la clé privée correspondant à une clé publique donnée. Il prend en paramètre la *master key*, une identité donnée et donne en sortie la clé privée associée à cette identité. C'est lors de cette étape que se pose le problème de *Key Escrow* car la personne en possession de la *master key* peut générer l'ensemble des clés privées du système.

— **Chiffrement** : Le message à transmettre est chiffré grâce à l'identité du receveur.

— **Déchiffrement** : Le déchiffrement du message reçu se fait à l'aide de la clé privée générée lors de l'étape de *Key extraction*.

4.4.3 IBE en ligne vs IBE hors ligne

Il existe deux principaux cas d'usage des IBE : les IBE en ligne et les IBE hors ligne. Les IBE en ligne reposent sur un générateur de clés privées (PKG) [1] accessible à travers le réseau. Lorsqu'un utilisateur reçoit un message chiffré avec son identité, il demande au PKG de lui générer sa clé privée et de la lui envoyer via le réseau. Cette propriété implique que l'utilisateur dispose d'un canal de communication sécurisé lui permettant d'interroger le PKG. Lorsqu'il a reçu la clé privée associée à son identité il peut dès lors déchiffrer le message. Ce schéma nécessite que l'utilisateur dispose de moyens de protection, de stockage et d'exécution sécurisés de sa clé privée.

Pour le cas d'un IBE hors ligne, l'utilisateur dispose du PKG et donc peut générer sa clé privée à sa guise. Lorsque cet utilisateur reçoit un message chiffré avec son identité, il génère sa clé privée et l'utilise pour déchiffrer le message. Cependant, il est nécessaire de protéger le PKG et notamment la *master key* et de ne lui permettre que de générer la clé privée de l'utilisateur qui l'héberge. Si aucun mécanisme de sécurisation du PKG n'est mis en place, un attaquant qui accède au PKG peut usurper les identités de l'ensemble des utilisateurs du système. Pour un déploiement au sein de notre architecture, nous proposons d'héberger le PKG sur le CSTEE côté smartphone et sur le SSTEE côté lecteur afin de sécuriser la *master key*.

1. Private Key Generator

4.4.4 Le schéma d'IBE de Callas

Callas dans [87] propose un schéma d'IBE qui peut être intégré dans l'écosystème RSA en fournissant les mêmes capacités cryptographiques qu'un schéma d'IBE classique. L'avantage de cette solution est de proposer un schéma d'IBE utilisant les primitives de chiffrement/déchiffrement et signature RSA pour offrir une grande compatibilité avec les systèmes à clé publique actuels. Évidemment, le niveau de sécurité de ce système se veut être au même niveau que celui de RSA. Cette caractéristique de compatibilité permet d'utiliser les bibliothèques de développement et les outils déjà déployés sans utiliser le pairing. Nous décrivons, dans ce qui suit les composants de l'IBE proposé par Callas.

4.4.4.1 Configuration du système

La *master key* du PKG (K_{pkg}) est utilisée pour calculer un jeton d'identité nommé *IDT* [1]. Cet *IDT* est calculé en appliquant une fonction de hashage de l'identité *IDF* [2] à la master key et l'identité de l'utilisateur suivant l'équation suivante :

$$IDT = IDF(K_{pkg}, Identity) \tag{4.1}$$

La fonction *IDF* peut être un algorithme *HMAC* (Keyed-Hash Message Authentication Code) ou un *CBC-MAC* [3] ou tout autre fonction pseudo-aléatoire sûre. Une autre solution consisterait à utiliser un cryptosystème RSA où K_{pkg} serait une clé RSA et l'*IDT* serait le résultat du chiffrement de l'identité par la master key du PKG. Comme l'IBE permet d'associer de manière unique une identité à une clé, il est nécessaire de faire en sorte que le mécanisme soit déterministe pour de futures générations. La sélection de la fonction IDF est cruciale dans la sécurité du système. En effet, la sécurité de l'IBE de Callas repose en grande partie sur la qualité de l'IDT générée à cette étape.

La seconde étape consiste à fixer la valeur de la graine (*seed*) du générateur pseudo-aléatoire avec la valeur de l'*IDT* obtenue. La fonction pseudo-aléatoire peut être une fonction de hashage ou toute autre fonction pseudo aléatoire. Néanmoins, afin d'augmenter la sécurité du système, il est préférable de choisir une fonction différente que celle utilisée pour générer l'*IDT*. Le résultat de cette opération est utilisé par la seconde phase d'extraction de la clé.

4.4.5 Extraction de la clé

L'algorithme présenté en Figure 4.3 décrit le processus d'extraction de la clé dans le schéma d'IBE de Callas. Il est indispensable de noter que l'utilisateur doit être authentifié au PKG afin qu'il puisse obtenir sa clé privée. Il est aussi nécessaire de disposer d'une communication

1. IDentity Token
2. Identity Digest Function
3. Algorithme d'authentification de messages basé sur le chiffrement symétrique selon le mode CBC (Cipher Block Chaining)

sécurisée entre le PKG et l'utilisateur afin que la clé ne puisse être interceptée par un attaquant. Cependant, la clé publique est délivrée à n'importe quel utilisateur, authentifié ou non.

```
IDT = IDF(K_pkg, Identity);
RNG_seed(IDT);
Key_Pair = KeyGenerator(RNG);
Key_Pair = (Pub_Key, Priv_Key);
if(!authenticated(user)) then
        return Pub_Key;
else
        return Key_Pair;
```

FIGURE 4.3 – Algorithme d'extraction de clé du schéma IBE de Callas

Le processus d'extraction de la paire de clés constitue la principale différence entre le schéma de Callas et les autres schémas d'IBE. En effet, contrairement aux autres schémas, la clé publique n'est plus l'identité en elle-même mais un dérivé de l'identité. Cette différence est due au fait que les clés RSA, sur lesquelles le système est basé doivent être générées d'une manière précise et donc l'identité en elle-même ne peut être utilisée directement.

4.4.6 Chiffrement/Déchiffrement

Le processus de chiffrement et de déchiffrement de ce schéma d'IBE est exactement le même que pour un schéma de RSA classique. L'émetteur du message utilise la clé publique pour chiffrer les messages et le récepteur utilise sa clé privée pour les déchiffrer.

4.5 Le protocole d'authentification

Après avoir décrit l'architecture de dématérialisation que nous avons choisie, passons maintenant à la partie protocolaire. Notre protocole d'authentification est basé sur IBAKE. Nous ferons en début de section un rappel de ce protocole avant d'exposer le protocole déployé sur l'architecture.

4.5.1 IBAKE

Identity-Based Authenticated Key Exchange (IBAKE) [83] est un protocole d'échange de clé authentifié basé sur la notion d'IBE et des mécanismes de courbes elliptiques [88]. Il a pour but de pouvoir négocier une clé de session entre deux entités sans avoir de secret partagé entre les deux. Les deux parties doivent cependant se mettre d'accord sur des paramètres publics tels que le choix de la courbe C et d'un point P avant le début du processus d'authentification pour pouvoir réaliser les opérations de pairing. La Figure 4.4 décrit le diagramme de séquence du protocole IBAKE.

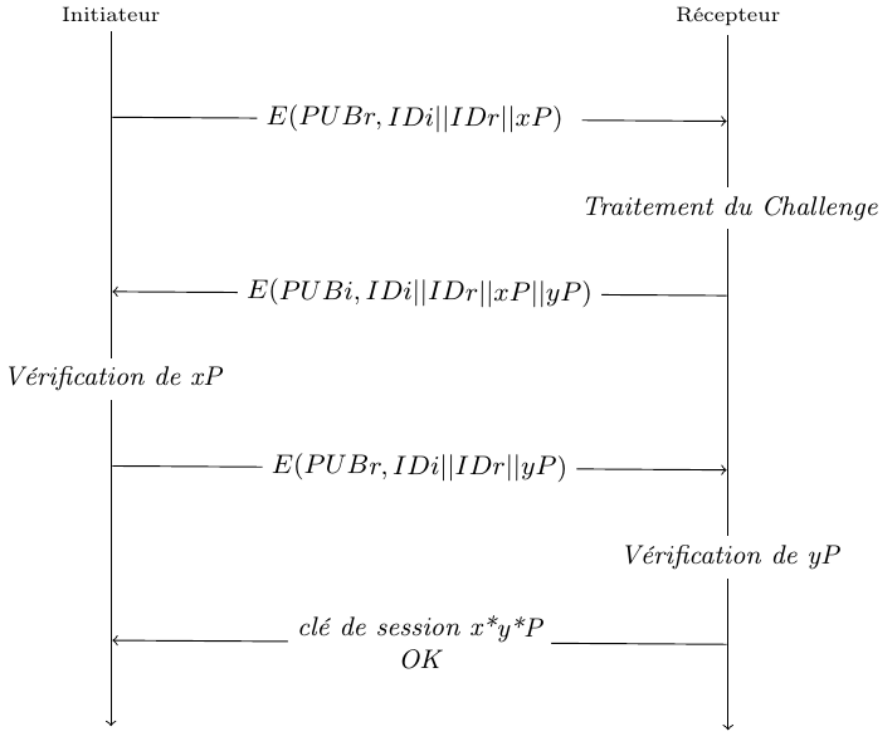

FIGURE 4.4 – Diagramme de séquence du protocole IBAKE

Dans un premier temps, l'initiateur du processus d'authentification génère de manière aléatoire une valeur x (challenge) représentant la partie privée du mécanisme de courbe elliptiques. Ensuite, il calcule le produit $x * P$ dont il inclut la valeur dans un paquet contenant son identité et l'identité de son interlocuteur. Ce paquet est chiffré avec l'identité du récepteur. Lorsque ce dernier reçoit le paquet, il commence par le déchiffrer en utilisant sa clé privée et extrait la valeur $x * P$.

De la même manière, il génère de manière aléatoire une valeur y et calcule le produit $y * P$. Il construit un paquet de réponse incluant son identité, l'identité de l'initiateur, la valeur $x * P$ reçue et la valeur $y * P$ calculée. Ce paquet est chiffré avec la clé publique de l'initiateur. Lorsque ce dernier reçoit le paquet, il commence par le déchiffrer à l'aide de sa clé privée et extrait les deux valeurs reçues : $x * P$ et $y * P$. Il compare la première valeur à celle qu'il a générée. Si la valeur ne correspond pas, il interrompt le processus d'authentification sur un échec sinon il construit un nouveau paquet de réponse contenant la valeur $y * P$ et les deux identités, il le renvoie chiffré avec sa clé publique au récepteur. A la réception, ce dernier déchiffre le paquet et extrait la valeur $y * P$. Si la valeur ne correspond pas, le processus d'authentification se termine sur un échec. Sinon, les deux parties sont d'accord pour que $x * y * P$ soit la clé de session de leur communication.

Cet algorithme d'échange de clé authentifié est sécurisé et repose sur l'utilisation d'algorithmes robustes. Cependant, il est nécessaire que les opérations effectuées de part et d'autres soient réalisées dans un environnement sécurisé pour éviter toute fuite de données.

4.5.2 Format des paquets d'authentification

Avant de décrire le protocole mis en place pour l'authentification, focalisons nous un moment sur les paquets échangés. Dans le protocole IBAKE, il est préconisé d'envoyer dans le paquet d'authentification l'identité de l'initiateur, l'identité du récepteur ainsi que le ou les challenges générés. Les identités sont envoyées avec le paquet pour éviter des attaques de réflexion menées par un attaquant Man in The Middle (MiTM). Pour illustrer le degré de dangerosité de cette attaque, prenons le scénario suivant : un attaquant a la possibilité d'écouter et de modifier les paquets d'authentification qui circulent d'un initiateur à un récepteur.

Cet attaquant intercepte le premier paquet d'authentification émis par l'initiateur et remplace l'adresse IP de ce dernier par la sienne avant d'adresser le message au récepteur. Ce dernier considérera que c'est l'attaquant qui réclame l'authentification et poursuivra le processus en interagissant directement avec lui et en chiffrant tous ses paquets avec l'identité de l'attaquant. Dès lors, l'initiateur de l'authentification ne peut accéder au service d'authentification. Cette attaque peut être vu comme une attaque de déni de service du point de vue d'un initiateur.

De plus, à partir du second échange, l'attaquant peut avoir accès au challenge généré par l'initiateur avec la même attaque décrite précédemment et jouer à ce moment le rôle de récepteur par rapport à l'initiateur. C'est pour cela qu'il est important d'inclure les deux identités dans la charge utile *payload* du paquet d'authentification.

Le paquet initial préconisé par les concepteurs d'IBAKE est décrit par la Figure 4.5.

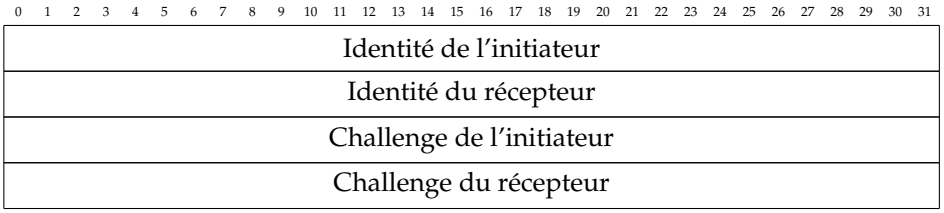

FIGURE 4.5 – Format du paquet IBAKE par défaut

Cependant, le protocole ne permet pas de répondre à certaines attaques connues : les attaques par rejeu et les attaques par altération de paquets (ou injection de faute dans le paquet). Pour ce faire, nous proposons d'augmenter le paquet avec deux champs supplémentaires : un numéro de séquence et un champ de contrôle d'intégrité.

Le numéro de séquence permet d'éviter les attaques par rejeu, où un attaquant aura enregistré une séquence d'authentification valide qu'il rejouera plus tard. Le champs vérifiant l'intégrité consiste en un hash qui va contenir l'empreinte du paquet et le récepteur pourra vérifier que le paquet reçu est au bit près celui qui a été envoyé. Le format du paquet que nous utilisons dans

notre protocole est décrit par la Figure 4.6. Nous laissons au développeur le soin de choisir le contrôle d'intégrité qu'il souhaite dans la limite de 48 octets (SHA-1, SHA-256, SHA-3).

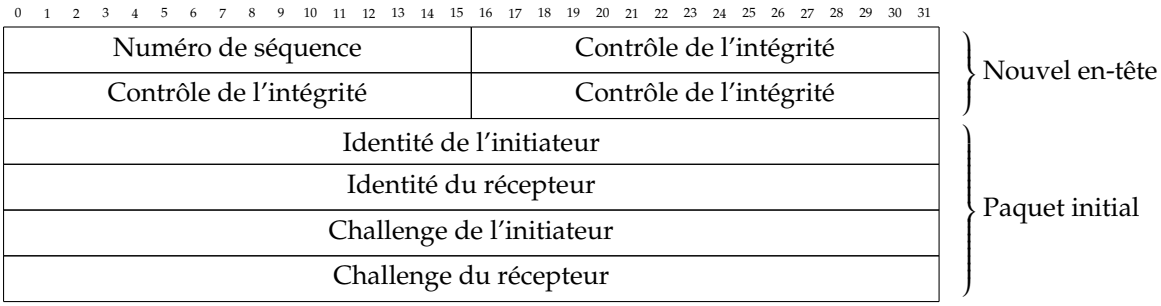

FIGURE 4.6 – Format du paquet IBAKE augmenté

L'ajout de ces champs au paquet d'authentification permet de rendre le protocole moins vulnérable aux attaques connues sur les protocoles d'authentification. Néanmoins, cet ajout augmente la taille du paquet de 128 octets à 192 octets soit un overhead d'un tiers. Cet overhead a un impact sur les performances de l'algorithme d'authentification. Cependant, la communication de plus faible débit dans l'architecture est la communication NFC avec un débit pouvant aller jusqu'à 424kbits/s. L'impact sur les performances sur cette communication n'est donc que de 0.001 secondes ce qui est largement acceptable.

4.5.3 Description du protocole d'authentification

Le protocole d'authentification que nous avons déployé sur notre architecture de dématérialisation est basé sur le protocole IBAKE décrit en section 4.5.1. Nous l'avons adapté à un usage Cloud et augmenté sa sécurité grâce aux ajouts sur le paquet standard pour répondre aux principales attaques courantes contre les protocoles d'authentification. Dans notre architecture, nous effectuons toutes les opérations cryptographiques et le stockage sécurisé dans le TEE déporté sur le CSTEE et le SSTEE. Notons qu'il est nécessaire que l'utilisateur s'authentifie à son CSTEE avant d'exécuter le protocole d'authentification pour le contrôle d'accès.

Le protocole IBAKE permet aux deux entités (l'utilisateur du smartphone et le lecteur) de négocier une clé de session qui sera utilisée dans les futurs échanges. Il est également à noter que cette authentification ne nécessite aucun secret partagé. En effet, tous les paramètres de l'authentification sont publics.

Le protocole IBAKE est implémenté sur un TEE Open Source nommé OP-TEE. Nous avons choisi, comme décrit précédemment, d'utiliser l'IBE proposé par Callas dans [87], basé sur

l'utilisation d'un RSA classique, pour des raisons de flexibilité et de compatibilité avec les bibliothèques déjà disponibles pour les développeurs. La Figure 4.7 représente le diagramme de séquences de l'algorithme d'authentification. Il est composé de 8 étapes principales :

- **1- Initier le processus d'authentification** : Lorsqu'un utilisateur veut accéder à un bâtiment sécurisé, il doit 'taper' son smartphone contre le lecteur mural. Le smartphone transmet, via la liaison NFC, une demande d'authentification auprès du lecteur. L'utilisation de la technologie NFC pour cette partie est très importante, car elle permet de vérifier que la personne qui demande l'accès est bien présente à moins de 10 centimètres de la porte. Cependant, pour l'instant, nous ne prenons pas en considération les attaques par relais sur ce scénario. En effet, les contres mesures telles que le *distance bounding* [89] peuvent être impactées par les communications avec le Cloud ce qui peut générer une latence rendant moins précises ces contre-mesures.
- **2- Demander la génération du challenge** : Lorsque le lecteur reçoit la demande d'authentification de l'utilisateur, il demande au SSTEE d'initier le processus d'authentification par la génération d'un challenge chiffré.
- **3 - 6 Messages IBAKE** : Cette partie correspond à l'échange des messages du protocole IBAKE. Ces messages sont acheminés via le lecteur et le smartphone à travers la connexion NFC. Les deux Clouds CSTEE et SSTEE ne peuvent en aucun cas communiquer directement sans passer par la liaison NFC. A la fin de cette étape, les deux parties se mettent d'accord sur une clé de session $x * y * P$. A partir de cette clé, nous pouvons dériver une clé symétrique AES et dès lors, un canal de communication chiffré par cette clé AES sera mis en place.
- **7-8 Contrôle d'accès authentifié** : L'utilisateur demande l'accès au bâtiment sensible et envoie ses données d'authentification (identité, jeton etc.) à travers le canal sécurisé mis en place lors de l'étape précédente. Le SAC (qui est isolé) se charge de vérifier dans sa base de données les autorisations de la personne authentifiée. Une notification est envoyée à l'utilisateur pour lui dire s'il est habilité à accéder à ce bâtiment ou pas.

Ce protocole sera mis en œuvre, sur une architecture virtualisée et sur une architecture matérielle, et évaluer en terme de performances et en terme de sécurité dans le Chapitre 5. Nous allons présenter dans ce qui suit des architectures alternatives en pointant leurs avantages et leurs inconvénients par rapport à l'architecture retenue.

4.6 Architectures alternatives

Après avoir présenté l'architecture principale considérée lors de ce travail, nous présentons maintenant deux architectures alternatives, pouvant exécuter le même protocole d'authentification. Ces architectures visent à accompagner le développement très rapide des TEE et misent sur une ouverture permettant le déploiement à grande échelle d'applications sécurisées.

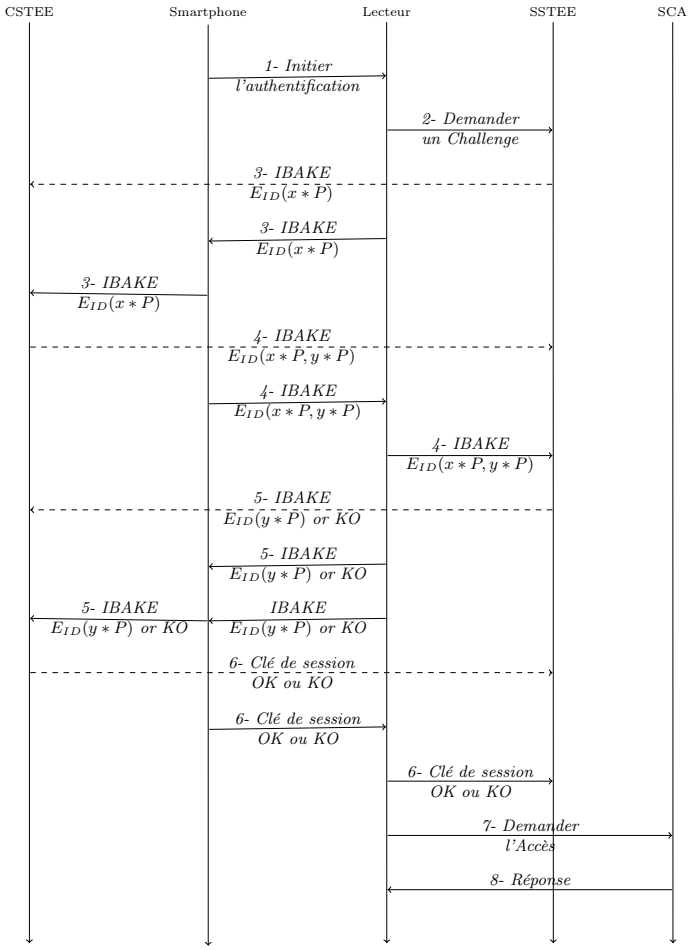

FIGURE 4.7 – Diagramme de séquences du protocole d'authentification proposé

4.6.1 Architecture : TEE embarqué

La Figure 4.8 décrit le deuxième scénario considéré. En effet, peu de smartphone (hormis les haut de gamme) sont dotés d'un TEE au moment de l'écriture de ce manuscrit. Cependant,

l'évolution du déploiement des TEE sur les smartphones de nouvelle génération nous laisse espérer que cette solution puisse être adoptée dans un futur proche.

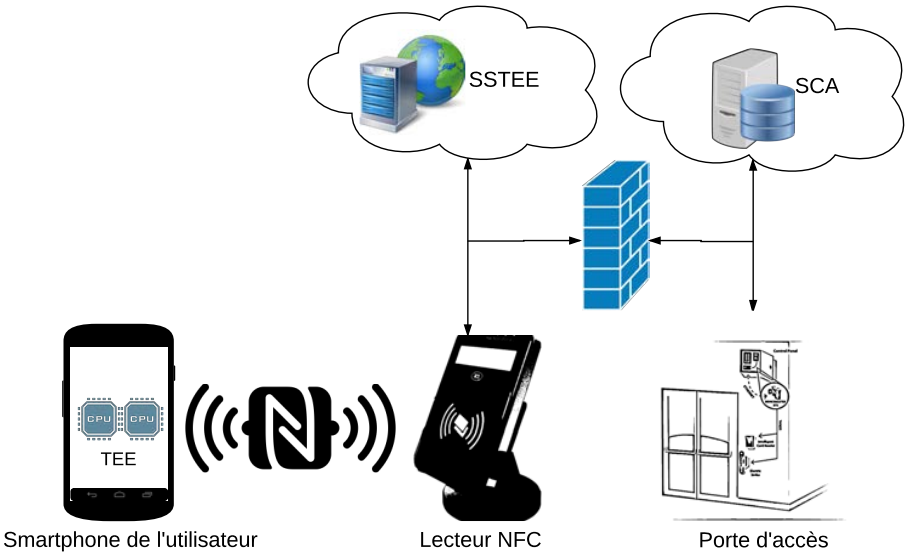

FIGURE 4.8 – Architecture 2 : TEE embarqué sur le smartphone

Sur le même principe que le scénario nominal, le smartphone s'authentifie au lecteur au travers du protocole défini. Cependant, au lieu que ce soit le TEE déporté qui se charge des opérations de stockage et des opérations cryptographiques, c'est le TEE embarqué sur le smartphone qui exécute l'instance du protocole d'authentification. Ce scénario présente de nombreux avantages parmi lesquels :

— Il n'y a nul besoin de s'authentifier au CSTEE avant de commencer le protocole d'authentification
— Le TEE embarqué gère les entrées/sorties avec l'utilisateur. Cette caractéristique permet d'interagir de manière sécurisée avec l'utilisateur. Il est dès lors possible de récupérer des informations supplémentaires d'authentification telles que des mots de passes, des codes PIN, des OTP ou empreintes digitales.
— Les performances seront meilleures car on élimine les échanges protocolaires vers Cloud.
— Il est possible de lutter contre des attaques par relais avec le *distance bounding* [89] du fait de l'absence de communication avec le Cloud.
— La négociation de la clé de session peut se faire hors ligne ce qui relâche la contrainte de connectivité initiale.

Nous remarquons aussi que nous gardons le serveur sécurisé du côté du lecteur. En effet, le lecteur ne possède pas la puissance de calcul suffisante pour exécuter l'algorithme d'authentification en fournissant des performances acceptables. De plus, il ne peut faire du stockage sécurisé

et est en première ligne face à un potentiel attaquant. Il n'est pas à exclure qu'un attaquant puisse accéder physiquement à un lecteur et donc la protection des données d'authentification présentes sur ce dernier est impérative.

4.6.2 Architecture 3 : TEE répartis

Avec l'apparition récemment de nombreux composants et cartes de développement supportant les TEE, il est légitime de se poser la question du remplacement du lecteur mural simple par un lecteur plus intelligent, embarquant un TEE et donc capable par lui même d'exécuter l'algorithme d'authentification. Le Raspberry Pi 3 muni d'un processeur type ARM et de la fonctionnalité TrustZone (TEE) est par exemple un candidat idéal à cette tâche. En effet, il suffit de lui ajouter une carte NFC type NXP [90] pour lui permettre de jouer le rôle de lecteur. La Figure 4.9 décrit cette nouvelle architecture.

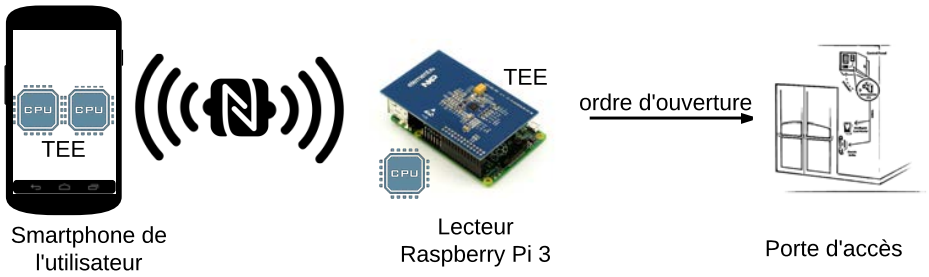

FIGURE 4.9 – Architecture 3 : TEE sur le smartphone et lecteur type Rasberry Pi 3

Cette solution possède l'avantage d'être très peu onéreuse. En effet, déployer un tel lecteur coûterait approximativement 80€. Ce montant correspond à 1/70 du prix d'achat d'une carte de développement type JUNO présentée en section 5.1.2.

Cette solution possède aussi de nombreux avantages :

— Possibilité d'ajouter aisément de nouvelles fonctionnalités telles qu'un écran tactile pour saisir un code PIN pour augmenter le processus d'authentification.
— Facilité et coût réduit pour un déploiement à grande échelle.
— Étant donné la capacité de stockage sécurisé du TEE embarqué, il est possible de stocker en local la base de données des personnes possédant les autorisations d'accès.
— Permettre de nombreuses fonctionnalités de journalisation grâce à la puissance de calcul du Raspberry telles que le nombre d'accès par heure ou lister les utilisateurs accédant à une salle spécifique dans le but de détecter de potentiels abus.
— Améliorer les performances du système de contrôle d'accès en éliminant les communications avec le serveur sécurisé distant ce qui lève l'hypothèse de la connexion Ethernet sécurisée. En effet, dans ce scénario, seule la connexion NFC est utilisée. Par rapport au scénario principal avec les deux serveurs, nous minimisons le temps de vol des paquets

d'authentification avant d'atteindre le destinataire.

Les solutions présentées jusqu'ici nécessitent l'exécution de l'algorithme d'authentification entre le smartphone et le lecteur ou entre leurs deux serveurs sécurisés. Cependant, en cas d'absence de connectivité, ces solutions seraient inutilisables. De plus, la consommation d'énergie (batterie) dûe à la connectivité peut être un problème pour les utilisateurs. Afin de remédier à ces inconvénients sur le contrôle d'accès sécurisé, nous proposons dans la section suivante de mettre en place un système de jetons de connexion hors connexion pour permettre à un utilisateur d'accéder aux bâtiments sécurisés de l'université.

4.7 Solution hors ligne : jetons de connexion

Nous avons exposé trois architectures de dématérialisation utilisant un composant sécurisé (TEE) et sur un algorithme d'authentification basé sur l'identité pour du contrôle d'accès. Dans cette section, nous exposons une quatrième alternative consistant à disposer d'un ou plusieurs jetons de connexion et de pouvoir les utiliser à travers l'infrastructure sans pour autant exécuter l'algorithme d'authentification.

4.7.1 Identification des menaces

Nous pouvons à ce stade identifier plusieurs problématiques posées par l'utilisation de ces jetons, parmi lesquelles nous pouvons citer :

— La génération de jetons doit être faite dans un environnement sécurisé. En effet, il est nécessaire de s'assurer qu'une tierce partie ne peut en aucun cas générer des jetons valides.

— L'acheminement des jetons vers le smartphone de l'utilisateur doit se faire au travers d'un canal de communication sûr. En effet, dans le cas contraire, un attaquant qui est à l'écoute de la communication peut intercepter et utiliser de manière frauduleuse ces jetons.

— Le stockage de ces jetons du côté du smartphone doit se faire de manière sécurisée. Tout défaut dans le stockage des jetons peut donner lieu à un vol de ces derniers.

— Il est nécessaire de pouvoir stocker les informations concernant les jetons générés et de pouvoir révoquer un jeton déjà utilisé. Dans le cas contraire, un attaquant écoutant la communication peut acquérir le jeton et rejouer la transaction pour pouvoir avoir accès frauduleusement à un bâtiment protégé.

4.7.2 Processus de génération du jeton

Afin de mettre en place le mécanisme d'authentification à l'aide de jetons, nous nous sommes basés sur l'architecture décrite par la Figure 4.10

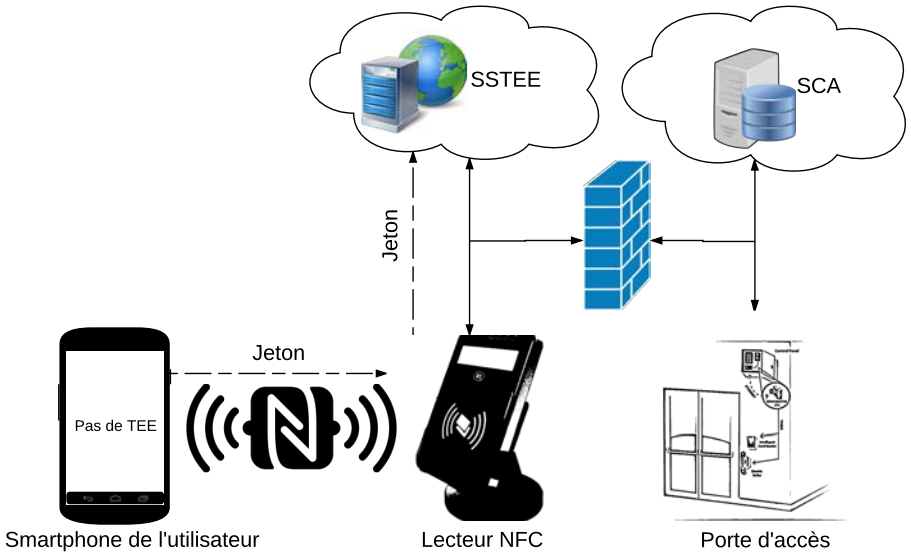

FIGURE 4.10 – Architecture d'authentification en utilisant les jetons de connexion

Afin de s'assurer de l'unicité de la source des jetons, nous décidons d'exécuter l'algorithme de génération des jetons sur le serveur sécurisé du lecteur. Le processus de génération du jeton peut être résumé comme suit :

- **Génération aléatoire d'un secret** : Le serveur sécurisé génère aléatoirement une valeur qui sera associée au jeton courant. Cette valeur sera concaténée à l'identité du possesseur du jeton et sera utilisée comme graine ou *seed* lors de la prochaine étape.
- **Génération du jeton par hash** : Le serveur sécurisé utilise le secret généré lors de l'étape précédente afin de fixer la graine du processus de HMAC. Le résultat de cette fonction représente la valeur du jeton.
- **Attribution d'un numéro de séquence** : Chaque jeton généré sur le serveur se voit attribuer un numéro de séquence unique permettant de l'identifier. Ce numéro est nécessaire afin de pouvoir révoquer un jeton déjà utilisé.
- **Chiffrement du jeton** : Le contenu du jeton est chiffré afin de garantir la confidentialité des données qui y sont présentes. Ce chiffrement est effectué à l'aide de la clé publique du serveur.
- **Calcul d'un hash global pour l'intégrité** : Pour garantir l'intégrité du paquet et s'assurer qu'il n a pas été altéré par un attaquant il est nécessaire d'intégrer au jeton une valeur qui correspond au hash des informations présentes.

La Figure 4.11 représente le contenu d'un jeton avant chiffrement.

Il est nécessaire de stocker le numéro de séquence et la valeur aléatoire générée lors de l'étape

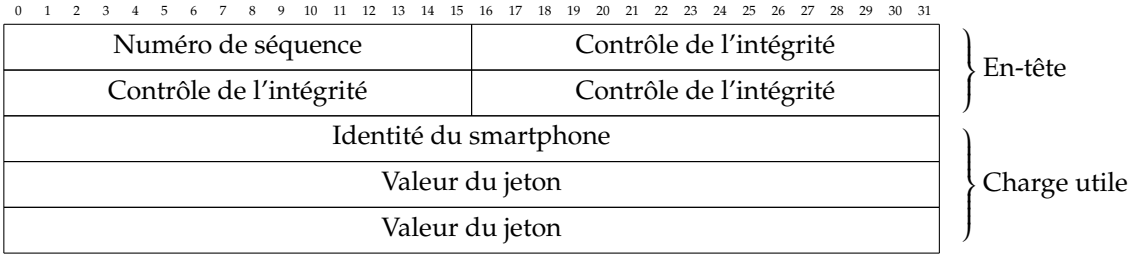

FIGURE 4.11 – Format du jeton de connexion hors ligne

1 du processus sur une base de données. En effet, afin de pouvoir révoquer les jetons utilisés et vérifier la validité d'un jeton reçu, ces informations sont essentielles. Une question importante se pose à présent : où stocker de manière sécurisée le jeton sur le smartphone ? La réponse à cette question n'est pas aisée. En effet, nous considérons que le smartphone n'est pas de confiance et ne peut dans ce cas faire du stockage et de l'exécution sécurisée. A ce stade, il est nécessaire d'intégrer un autre composant de sécurité sur le smartphone.

Au vu de la taille réduite des jetons, notre choix s'est fixé sur l'utilisation d'un Secure Element (SE) pour le stockage sécurisé des jetons. Le SE nous permet d'avoir une résistance importante aux attaques physiques et même si l'attaquant entre en possession d'un smartphone avec des jetons sur un SE, il lui est impossible de les extraire. Cependant, dans la solution neOCampus Access Control (OCAC) présentée en section 4.8, nous n'avons pas eu la possibilité de stocker le jeton sur le SE à cause des restrictions d'accès imposées par les opérateurs.

4.7.3 Processus d'authentification en utilisant un jeton

La Figure 4.12 représente le diagramme de séquence du processus d'authentification à l'aide des jetons. Lorsqu'un utilisateur souhaite s'authentifier à l'aide d'un jeton, il approche son smartphone du lecteur NFC pour transférer un des jetons en sa possession vers le serveur sécurisé à travers le lecteur. A la réception du jeton, le serveur sécurisé déchiffre le jeton et en extrait le contenu. Il commence d'abord par vérifier l'intégrité du paquet reçu. Dans un second temps, il vérifie que le numéro de séquence est valide. La validité du numéro de séquence signifie que le serveur a bien généré ce jeton et qu'il n'a pas été utilisé et révoqué. Ensuite, il extrait la valeur aléatoire stockée dans la base de données au moment de la génération du jeton. Il recalcule la valeur du jeton et la compare à la valeur reçue. Si les deux valeurs correspondent, il donne au lecteur l'ordre d'ouvrir la porte. Dans le cas contraire, la porte reste verrouillée et l'utilisateur se voit notifier le refus d'ouvrir la porte.

4.8 Prototypage de la solution neOCampus Access Control (OCAC)

Dans le cadre du projet neOCampus, nous avons développé un prototype d'un système de contrôle d'accès hors ligne avec des jetons de connexion dont nous venons d'en décrire la

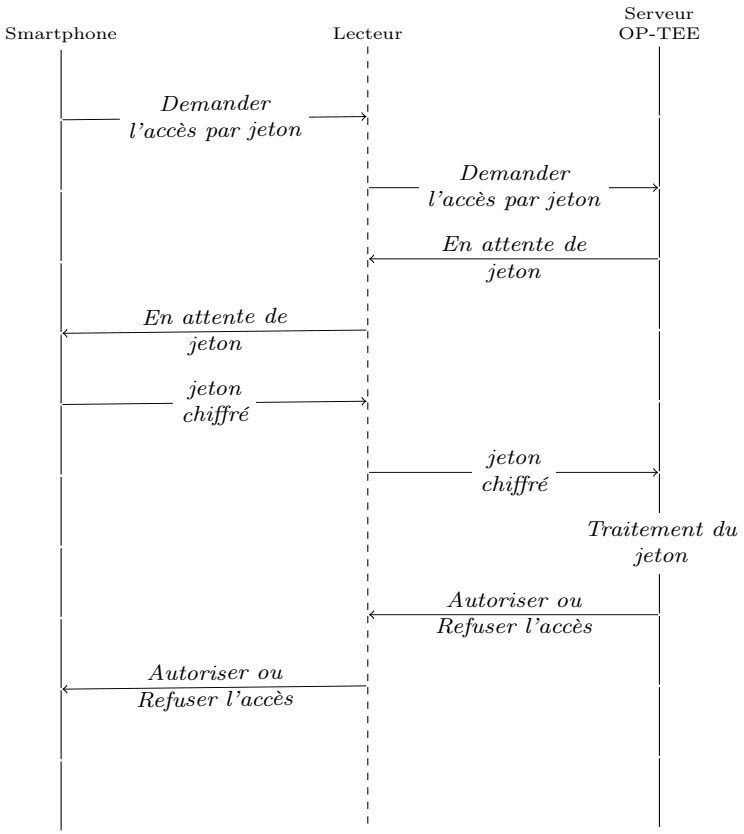

FIGURE 4.12 – Diagramme de séquence d'une authentification avec jeton

proposition. Cette solution met en œuvre la dématérialisation de la carte MUT (multi-services) destinées aux étudiants et aux personnels de l'université de Toulouse III. La Figure 4.13 permet de visualiser le processus de dématérialisation de cette carte MUT que nous allons détailler dans la suite.

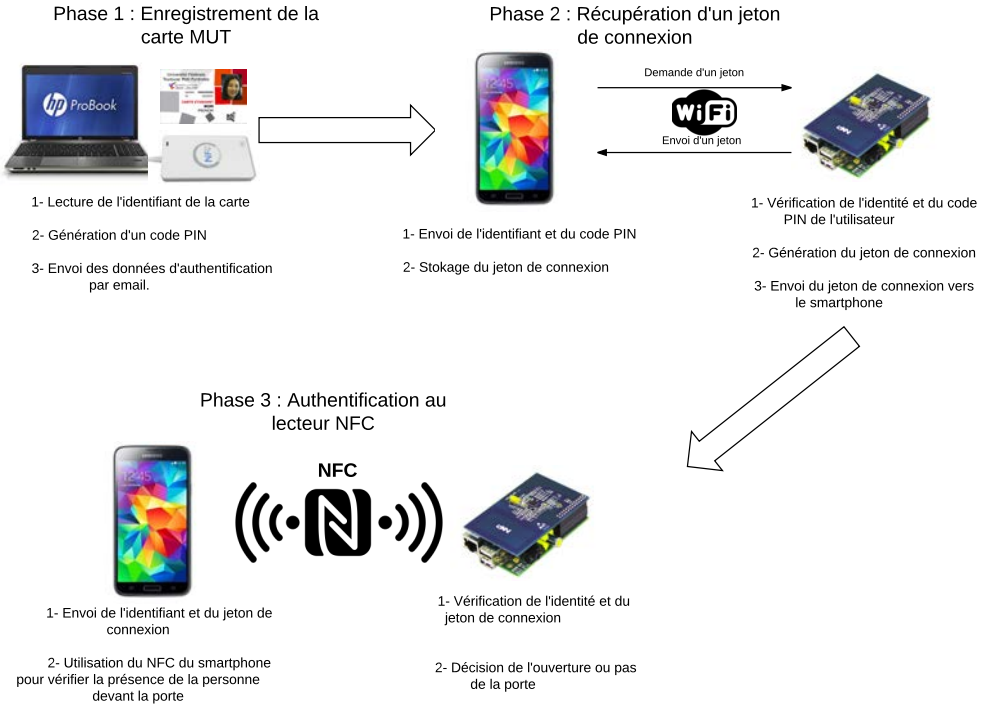

FIGURE 4.13 – Phases du prototypage de la solution OCAC

4.8.1 Phase 1 : Enregistrement de la carte MUT

Durant cette phase du processus, nous procédons à l'enregistrement de la carte MUT dans le système de contrôle d'accès. Pour ce faire, nous utilisons un lecteur NFC de type ACR122 pour lire l'UUID de la carte. La Figure 4.14 représente la capture d'écran du processus d'enregistrement de la carte MUT. La phase d'enregistrement se déroule comme suit :

1. **Lecture de l'identifiant de la carte** : L'utilisateur pose sa carte sur le lecteur et le programme d'enregistrement en lit l'UUID. Cet UUID sera utilisé comme identifiant de connexion lors de la phase suivante.
2. **Génération du code PIN** : A partir de cet identifiant, nous générons un code PIN par l'application d'un HMAC sur l'identifiant. Cette procédure permet d'être sûr qu'un attaquant ne puisse re-générer ce PIN (il est sur 6 ou 8 caractères) utilisé ultérieurement pour la récupération d'un jeton.
3. **Envoi des données d'authentification par mail** : Ces données sont envoyées à l'utilisateur automatiquement par mail. L'utilisateur peut désormais les utiliser pour récupérer son jeton de connexion auprès du serveur. Il est possible d'augmenter la sécurité de cet envoi

de mail par l'utilisation de l'outil logiciel openPGP par exemple pour la signature et/ou le chiffrement du message électronique.

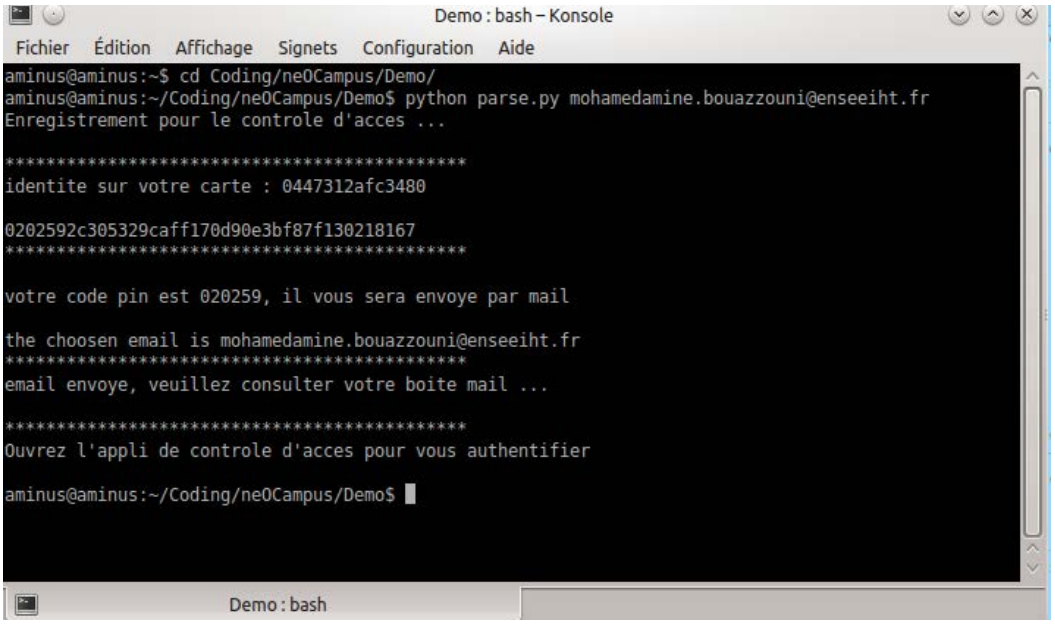

FIGURE 4.14 – Phase 1 : Enregistrement de la carte MUT

Après la génération et l'envoi des données d'authentification à l'utilisateur par mail et/ou SMS, il contacte via son smartphone un serveur sur Raspberry Pi pour la génération des jetons de connexion.

4.8.2 Phase 2 : Récupération d'un jeton de connexion

La Figure 4.15 permet de visualiser les actions effectuées par le serveur de jetons lorsqu'il est sollicité par un smartphone. Durant cette phase, le smartphone émet une requête vers le serveur de jetons en utilisant l'identifiant et le code PIN reçu via la liaison Wi-Fi. Le serveur réagit à la requête comme suit :

1. **Vérification de la validité de l'identifiant et du PIN** : Le serveur se charge de vérifier, dans la base de données partagée avec le serveur d'enregistrement des utilisateur, que ce dernier s'est enregistré pour le service. Dans le cas contraire, une erreur est transmise au smartphone. Dans le cas où l'utilisateur est présent dans la base de données, le serveur re-génère le code PIN à partir de l'identifiant de l'utilisateur et le compare à celui transmis par le smartphone. Dans le cas où les deux codes PIN correspondent, un jeton de connexion est généré sinon une erreur est notifiée à l'utilisateur.
2. **Génération du jeton de connexion** : La génération du jeton de connexion est effectuée par l'application d'une autre fonction HMAC (que celle utilisée pour la génération du

code PIN) sur l'identifiant. Nous procédons à un *seed* de la fonction HMAC avec un secret. Ce jeton n'est pas reproductible par un attaquant tant que le secret utilisé comme graine n'est pas compromis.

3. **Envoi d'un jeton de connexion vers le smartphone** : Le jeton de connexion est envoyé au smartphone via la liaison Wi-Fi. Nous ajoutons à ce jeton une date limite d'utilisation au delà de laquelle il ne sera plus valable.

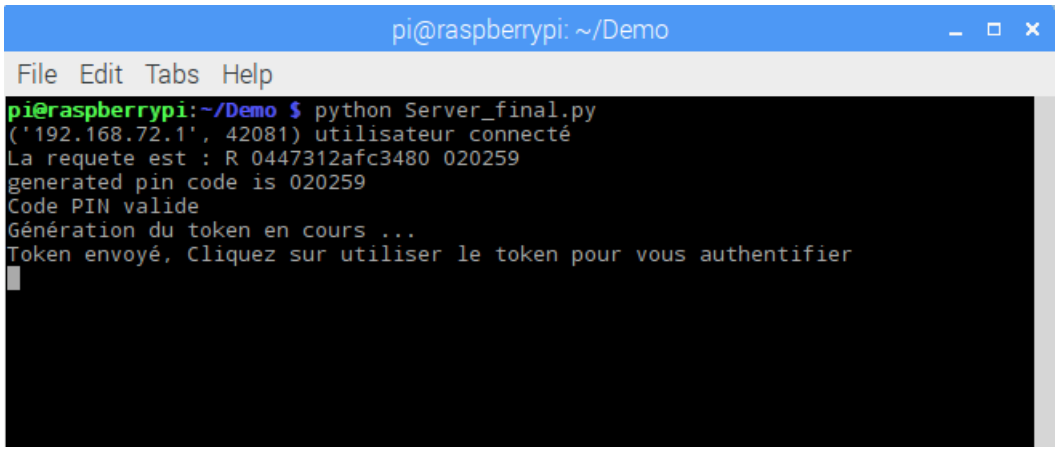

FIGURE 4.15 – Phase 2 : Récupération d'un jeton de connexion

La Figure 4.16 permet de visualiser l'application Android OCAC pour le contrôle d'accès dématérialisé avant la requête au serveur pour récupérer un jeton. Il est nécessaire de renseigner les champs ID et PIN sinon une erreur sera levée et le processus d'authentification s'arrête. Après avoir cliqué sur le bouton *Récupérer le token*, le serveur traite la requête, génère le jeton et l'envoi à l'utilisateur pour qu'il puisse l'utiliser. La Figure 4.17 permet de visualiser dans l'application OCAC la valeur hexadécimale du jeton reçu. A ce moment de l'authentification, l'utilisateur doit rapprocher son smartphone du lecteur NFC et cliquer sur le bouton correspondant pour l'utiliser.

Dès que le téléphone reçoit le jeton, il le stocke en vue d'une utilisation ultérieure. La phase suivante consiste à transmettre le jeton au serveur de contrôle d'accès via la liaison NFC. Il est donc nécessaire, pour mener à bien cette dématérialisation, que le smartphone soit muni de cette technologie.

4.8.3 Phase 3 : Authentification au lecteur NFC

La Figure 4.18 illustre le processus d'authentification du smartphone en utilisant le jeton. Une fois que le smartphone dispose de son jeton, l'utilisateur peut se présenter devant le lecteur NFC pour demander l'authentification. Pour se faire, il envoie une requête d'authentification au serveur de contrôle d'accès représenté par un Raspberry Pi muni d'une carte NFC du constructeur

FIGURE 4.16 – Phase 2 : Avant la requête de demande d'un jeton

FIGURE 4.17 – Phase 2 : Réception et affichage du jeton

NXP. Ce dernier fournit aussi les sources du mécanisme d'émulation de cartes afin de pouvoir le modifier à sa guise. Le serveur exécute les actions suivantes :

— **Vérification de l'identité** : Le serveur de contrôle d'accès vérifie à travers l'identité de l'utilisateur, que ce dernier est éligible pour utiliser ce service. Cette vérification est effectuée avec l'identifiant et la comparaison du code PIN envoyé dans la requête avec celui regénéré. En cas d'échec de cette vérification préalable, le smartphone se voit notifier une erreur d'authentification.
— **Vérification du jeton** : Le serveur de contrôle d'accès re-génère un jeton avec les informations de l'utilisateur. Si le jeton généré ne correspond pas au jeton envoyé, une erreur est transmise au smartphone signifiant l'échec de l'authentification. Dans le cas contraire, le serveur de contrôle d'accès vérifie que la validité du jeton n'a pas expiré avant de décider de l'ouverture de la porte pour l'utilisateur.

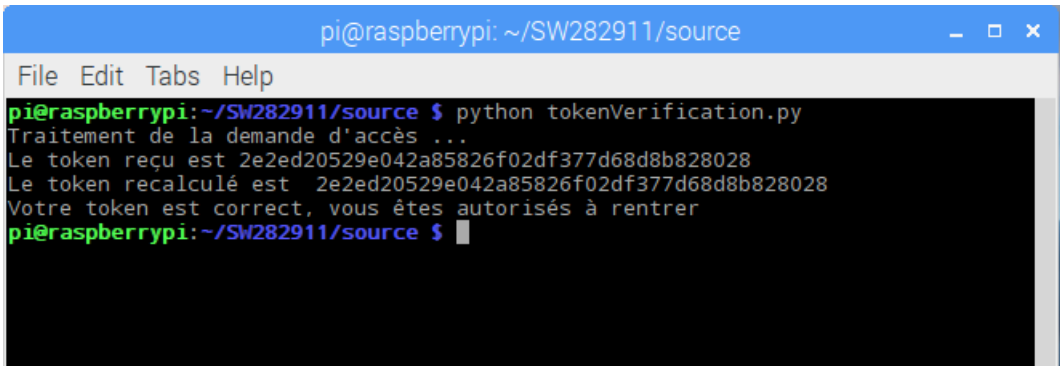

FIGURE 4.18 – Phase 3 : Authentification au lecteur NFC

4.8.4 Limites de la solution OCAC

Nous avons identifié trois principales limites à la solution OCAC actuelle :

— Le mécanisme de transmission du code PIN à l'utilisateur est vulnérable dû à l'interception possible par un attaquant. En effet, si un malware est déjà présent sur le smartphone, il pourra intercepter l'email contenant l'identifiant et le code PIN de l'utilisateur enregistré.
— La génération du jeton sur le Raspberry Pi doit être exécutée dans un environnement sécurisé. Nativement, le Raspberry (sauf si nous utilisons le TrustZone du RPi3) n'offre pas de sécurité dans le stockage et le traitement des données.
— Le stockage du jeton sur le smartphone est problématique. En effet, le système d'exploitation du smartphone ne peut être considéré comme un environnement de confiance. Vu le nombre croissant de malware sur Android, nous ne sommes pas à l'abri qu'un d'eux intercepte le jeton et l'utilise avant l'utilisateur de manière frauduleuse.

4.8.5 Améliorations

Nous avons repéré sur notre implémentation, un certain nombre d'améliorations augmentant le niveau de sécurité du prototype parmi lesquelles nous pouvons citer :

— **Utiliser un Raspberry Pi 3** : L'utilisation du Raspberry Pi 3 peut contribuer à améliorer les performances du système de contrôle d'accès. De plus, à partir de cette version, le Raspberry Pi supporte partiellement la technologie TrustZone d'ARM. Dès lors, nous pouvons déployer un Secure OS (OP-TEE) sur le Raspberry pour disposer d'un TEE complet pour le stockage et le traitement sécurisé des jetons.
— **Utiliser le SeekforAndroid [91]** : L'utilisation du SeekforAndroid permet d'utiliser les facultés des Secure Elements présents sur le smartphone. Cette caractéristique est nécessaire afin de garantir le stockage sécurisé des jetons.

4.9 Conclusion

Nous avons dans ce chapitre présenté le concept de dématérialisation des cartes sans contact sur un smartphone Android. Nous avons identifié l'inconvénient majeur de la dématérialisation sur un smartphone qui est le stockage et le traitement des données d'authentification. Pour y remédier, nous avons proposé diverses architectures de dématérialisation : une architecture optimale embarquant le TEE sur le smartphone, une architecture Cloud et une architecture répartie. Pour l'architecture avec un TEE embarqué, nous faisons face à la contrainte de disponibilité des TEE sur un large panel de smartphones et aussi au souci d'obtenir les accords nécessaires au déploiement d'applications sécurisées sur les TEE. L'architecture retenue permet de garder un niveau de sécurité de l'authentification équivalent en introduisant la notion de TEE déporté sur le Cloud. En effet, le smartphone et le lecteur possèdent respectivement chacun un TEE déporté sur un Cloud chargé de l'exécution de l'algorithme d'authentification. La troisième architecture consiste en une solution visant à utiliser la capacité TrustZone de périphériques à faible coût comme les Raspberry Pi 3 pour éviter l'utilisation des Cloud. Néanmoins, cette solution n'est envisagée que dans un futur proche pour accompagner le développement des TEE. Nous avons aussi proposé une architecture d'authentification hors ligne basée sur l'utilisation de jetons de connexions. Cette solution a donné lieu à une preuve de concept OCAC dans le cadre du projet neOCampus. Elle permet de remédier à la contrainte de connectivité forte pour le scénario retenu. Nous avons proposé un algorithme d'authentification basé sur IBAKE permettant une double authentification entre le smartphone et le lecteur. Dans la suite du manuscrit, nous aborderons l'environnement Open Source de TEE OP-TEE sur lequel nous avons déployé notre algorithme d'authentification ainsi que les résultats de l'évaluation de performances et l'évaluation en terme de sécurité de ce même algorithme.

5 Évaluation de performances et de la sécurité du système

Introduction

Après avoir exposé les problématiques que posent l'utilisation des cartes sans contact sur la sécurité des systèmes de contrôle d'accès, nous avons élaboré une architecture de dématérialisation permettant à un téléphone de se substituer à une carte sans contact tout en offrant un niveau de sécurité supérieur. Nous avons aussi proposé un protocole d'authentification basé sur l'identité s'appuyant sur le protocole IBAKE et en l'améliorant pour atteindre un meilleur niveau de sécurité afin de répondre aux attaques connues sur les algorithmes d'authentification. Nous proposons dans ce chapitre de procéder à une évaluation de performances pour le protocole d'authentification proposé sur les architectures de dématérialisation. De plus, nous effectuons une validation formelle de celui-ci à l'aide de l'outil Scyther [9].

5.1 Environnement de développement

5.1.1 OP-TEE

La Figure 5.1 [63] permet de visualiser l'architecture d'OP-TEE.

OP-TEE est une implémentation open-source du TEE qui peut être exécuté en mode virtualisé ou sur diverses plate-formes de développement matérielles. Il est conforme aux spécifications de GlobalPlatform. Il fournit aux développeurs des librairies (cryptographique et standards du langage C) afin de faciliter le développement d'applications sécurisées mais n'est actuellement pas disponible sur des TEE commercialisés. OP-TEE est divisé en deux parties distinctes : l'OS standard (Linux et Android) avec ses applications clientes (CA) et le Secure OS avec ses application sécurisées (Trusted Applications (TA)).

5.1.1.1 OS standard et applications clientes

Il s'agit du système d'exploitation standard ou Rich Execution Environment (REE). OP-TEE supporte le déploiement d'un système Linux ou Android sur la partie REE. Il fournit un pilote

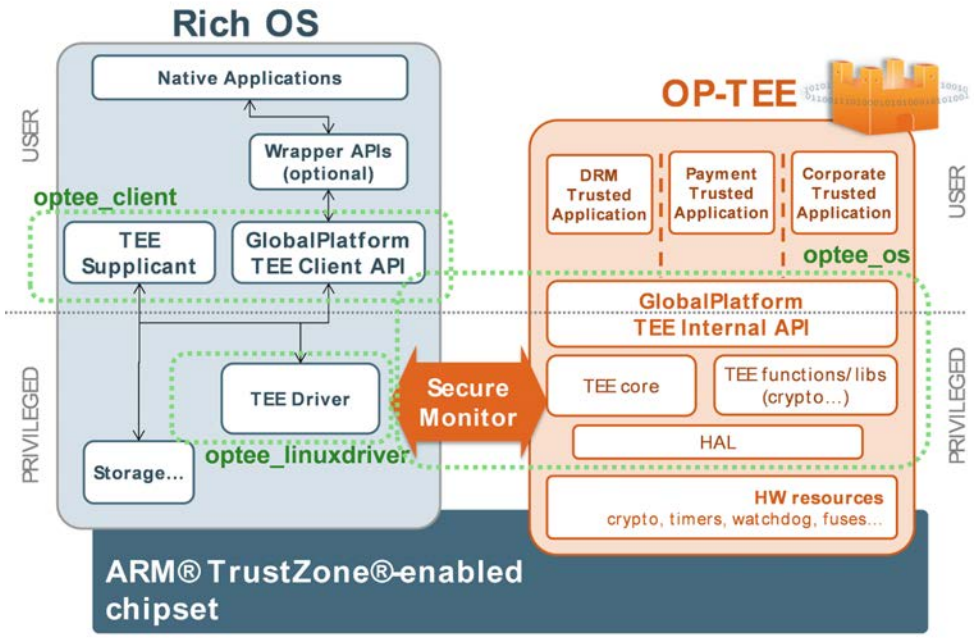

FIGURE 5.1 – Architecture d'OP-TEE

*/dev/tee*0 permettant aux applications clientes (CA) d'interagir avec les applications sécurisées (TA). En effet, il est possible qu'une CA ait besoin de partager des données avec une TA ou que cette CA fournisse les entrées (appelées paramètres) sur lesquelles la TA va exécuter son code. Les paramètres partagés avec la TA sont fortement typés et la mémoire réservée est limitée.

Parmi les paramètres, il en existe deux types : paramètre de type *value* et paramètre de type pointeur (*memref*). Le type *value* correspond à des valeurs numériques tandis que le type pointeur permet de transmettre des données sous forme de chaîne de caractères; par exemple une chaîne à hasher ou un hash à vérifier.

Les paramètres peuvent aussi servir à transmettre les sorties de la TA à la CA pour affichage ou traitement ultérieur. Le nombre de paramètres disponibles est de quatre au maximum.

5.1.1.2 Secure OS et applications sécurisées

Le Secure OS est capable de faire du *multithreading*, de gérer la mémoire allouée et d'exécuter et d'isoler les TA entre elles et avec les CA. Le Secure OS peut être perçu comme un esclave par rapport à l'OS standard dans le REE. En effet, c'est ce dernier qui émet une requête d'exécution d'une TA sur des données fournies en entrée. Le Secure OS peut exécuter simultanément autant d'instances de TA que le nombre de cœurs du processeur disponible à un instant donné.

5.1.2 La carte de développement ARM JUNO

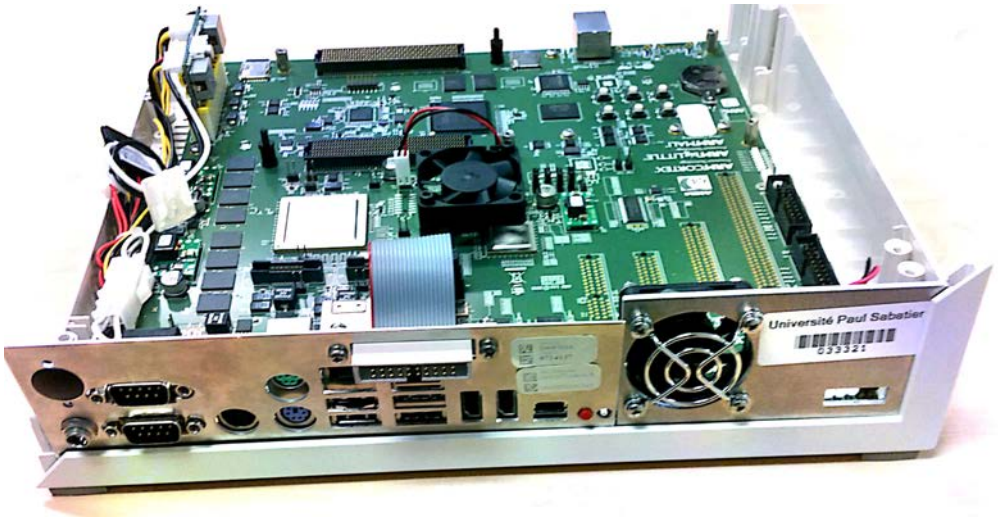

FIGURE 5.2 – Carte de développement ARM JUNO R0

L'ARM JUNO est une carte de développement de chez ARM basée sur une architecture 64 bits et une carte mère ARM Versatile Express V2M. Elle permet aux développeurs de tester des fonctionnalités des processeurs ARM qui sont, dans la majorité des cas, verrouillées sur les produits finaux. Dans notre cas, elle nous permet d'utiliser les capacités TrustZone du processeur sans que cela ne nécessite une signature de l'application par le fabriquant du processeur et/ou fabriquant du téléphone. La carte ARM JUNO R0 que nous utilisons possède les caractéristiques techniques suivantes :

— Un processeur Dual core Cortex-A57 MPCore.
— Un processeur Quad core Cortex-A53 MPCore.
— Un processeur GPU dédié Mali-T624 series.
— 8 Go de mémoire RAM dédiée.
— Des périphériques d'entrées/sorties (Ports USB, Ports HDMI, Ports séries, Ports Ethernet).

La Figure 5.2 présente cette carte de développement ARM JUNO R0. Elle permet de déployer OP-TEE et d'utiliser les fonctionnalités TrustZone des processeurs ARM présents. Ces derniers

sont les mêmes que ceux présents sur différents smartphones récents tels que le Samsung Galaxy S6 et le Note 5, le LG G4 et le HTC M8.

Pour notre étude expérimentale, nous avons déployé OP-TEE sur ARM JUNO R0 et nous avons développé une application exécutée sur la carte pour nous permettre d'avoir deux types d'évaluation : une évaluation sur l'environnement physique de la carte et une évaluation sur l'hyperviseur d'ARM FVP [1].

5.1.3 Développement d'applications sécurisées sur OP-TEE

Une application s'exécutant sur OP-TEE est une interaction entre deux composants logiciels : Client Application (CA) et Trusted Application (TA). La partie CA se charge des interactions avec l'utilisateur en récupérant les entrées. Ces dernières peuvent être des données à chiffrer/-déchiffrer, des hashs à vérifier, une taille de clé RSA à générer etc. La partie TA se charge de l'exécution des opérations sensibles telles que le chiffrement, le déchiffrement, la signature etc. Les interactions entre la CA et la TA permettent à cette dernière de récupérer les données à traiter et à la CA (et donc à l'utilisateur) de disposer des résultats générés. La Figure 5.3 permet de visualiser l'architecture d'une application OP-TEE. Afin d'illustrer la manière de développer une application pour cet environnement, nous prenons comme exemple une application permettant de calculer le hash par la TA d'une chaîne donnée en entrée par la CA et l'affichage du résultat à l'utilisateur par l'application cliente (CA).

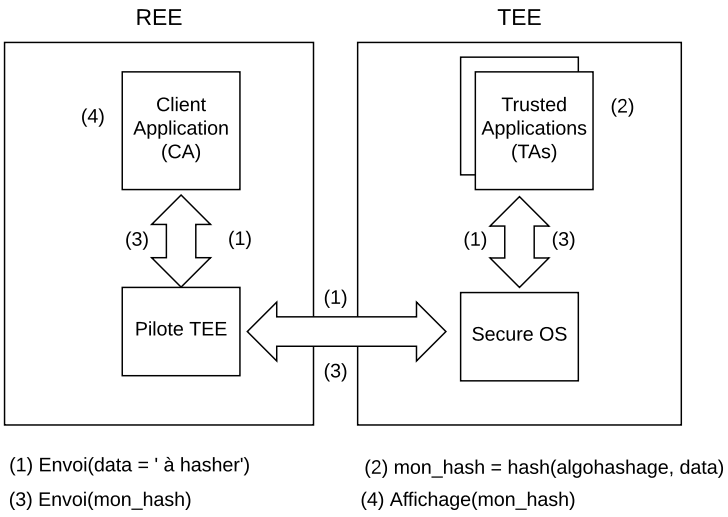

FIGURE 5.3 – Interactions CA/TA avec OP-TEE

1. Fixed Virtual Platform

5.1.3.1 La partie CA de l'application

Le code de la CA est indépendant de celui de la TA et se trouve dans un fichier et un répertoire différent. Chaque TA possède un Universally Unique IDentifier (UUID). Cet UUID est utilisé par la CA pour ouvrir une session avec la bonne TA présente sur la partie TEE d'OP-TEE.

Sur chaque CA, il est nécessaire d'inclure deux *headers* essentiels pour le processus de développement : le *tee_client_api.h* et *tee_shared_types.h* dont le contenu est le suivant :

— *tee_client_api.h* : Contient la définition de toutes les structures utilisées dans la CA comme le contexte, les sessions, les invocations, les types inhérents aux TEE et certaines constantes définies par des macros.

— *tee_shared_types.h* : Contient la définition d'un certain nombre d'erreurs relatives à l'ouverture de la session de l'invocation d'une commande sur le TEE, les erreurs liées à la mémoire partagée etc.

Il est à noter que sur la dernière version d'OP-TEE, le fichier *tee_client_api.h* contient la concaténation des deux fichiers décrits ci-dessus.

Après avoir inclus les *headers* nécessaires, le développeur peut commencer à écrire le code de sa CA. Le Code 5.1 présente la CA utilisée pour l'application de hashage.

Listing 5.1 – Squelette d'une CA

```
1  #include<stdio.h>
2  #include<err.h>
3  #include<tee_client_api.h>
4  #define TA_EXAMPLE1_CMD_VALUE 0
5
6  int main(int argc, char *argv[]){
7
8      TEEC_Result res;
9      TEEC_Context ctx;
10     TEEC_Session sess;
11     TEEC_Operation op;
12     TEEC_UUID uuid = TA_EXAMPLE1_UUID;
13     uint32_t err_origin;
14     void* tohash = "thesebouazzouni2017";
15     uint32_t digest_length = 20;
16     /* Contient la sortie du hash SHA-1 de tohash */
17     uint8_t digest[SHA1_SIZE];
18
19     /* Initialisation du contexte */
20     res = TEEC_InitializeContext(NULL, &ctx);
21
22     /* Tester le resultat de l'initialisaition */
23
24     if(res != TEEC_SUCCESS)
25       errx(1, "Failed to create context with code 0x%x",res,err_origin);
26
27     /* Ouverture d'une session vers une TA specifique avec son UUID */
28     res = TEEC_OpenSession(&ctx, &sess, &uuid,
29           TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);
30
31       /* Tester la bonne ouverture de la session*/
32
```

```
33      if(res != TEEC_SUCCESS)
34        errx(1, "Failed to create session with code 0x%x",res,err_origin);
35
36      /* Invoquer une operation dans la TA */
37
38      memset(&op, 0, sizeof(op));
39
40      /* op est de type TEEC_Operation
41       * Avec un tableau de 4 param tres */
42
43      /* Indiquer quel parametre est utilise, son type et sa taille*/
44      op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INOUT,
45        TEEC_NONE,
46        TEEC_NONE, TEEC_NONE);
47
48      /* Le premier parametre est une memoire temporaire partagee
49       * qu'on initialise avec le pointeur tohash */
50      op.params[0].tmpref.buffer = tohash;
51      op.params[0].tmpref.size = strlen((char*)tohash);
52
53      /* La fonction InvokeCommand prend en param tre
54       * la session creee, un ID pour la commande    executer,
55       * et une variable contenant le message d'erreurs
56       * le cas echeant*/
57      res = TEEC_InvokeCommand(&sess, TA_EXAMPLE1_CMD_VALUE, &op, &err_origin);
58
59      /* Tester le succes de InvokeCommand*/
60
61      if(res != TEEC_SUCCESS)
62        errx(1, "Failed to invoke the command with code 0x%x",res,err_origin);
63
64      /* Nous sommes arrivees    la fin de l'execution de la TA,
65       * Nous detruisons le contexte et la session */
66
67      TEEC_CloseSession(&sess);
68      TEEC_FinalizeContext(&ctx)
69  }
```

A partir de ce code, nous pouvons dénombrer quatre principales étapes d'une CA :

— **Établissement d'un contexte entre la CA et la TA** : Permet la création d'un contexte d'exécution commun pour la CA et la TA.

— **Établissement de la session** : La CA établie une session avec une TA spécifique via l'UUID de cette dernière. Cette ouverture de session peut être soumise à une connexion préalable selon le type de TA (publique ou privée). Par défaut, toutes les TA sont publiques.

— **Définition des paramètres** : La CA dispose de quatre paramètres à partager avec la TA. Lors de cette étape il est nécessaire de spécifier le nombre de paramètres utilisés, ainsi que leur type, et définir s'ils sont des paramètres en entrée, en sortie ou en entrée/sortie de la TA.

— **Invoquer une commande sur la TA** : Il est maintenant possible d'invoquer une opération à exécuter sur la TA. L'ID de la commande à exécuter permet de différencier les différentes opérations que peut exécuter une TA. Par exemple, l'ID 0 correspond au hashage de notre chaine de caractères et l'ID 1 correspond au chiffrement de cette même chaine.

Nous observons que pour notre exemple, nous utilisons un seul paramètre sur les quatre disponibles. Ce paramètre est de type *TEEC_MEMREF_TEMP_INOUT* ce qui signifie qu'il est de type mémoire partagée et qu'il est utilisé comme entrée et sortie pour la TA. Il lui sert notamment à recevoir la chaîne de caractères à hasher et à transmettre le résultat vers la CA pour affichage.

5.1.3.2 La partie TA de l'application

Le code de la TA se divise en deux parties principales :

— **Phase de configuration** : Durant cette phase, la TA répond à l'ouverture de session effectuée en amont par la CA et récupère les paramètres définis. La TA identifie l'opération à exécuter et lance la fonction correspondante.

— **Les opérations à exécuter** : A chaque opération qu'exécute la TA correspond une fonction. Par exemple, l'opération de hashage d'une chaîne de caractères est exécutée par la fonction ExecuterHash, et sur cette même TA, il est possible d'avoir une opération de chiffrement caractérisée par la fonction ChiffrerChaine.

Phase de configuration

Le Code 5.2 correspond à la phase de configuration de la TA que nous avons développée. A l'issue de cette phase, une session est ouverte avec la CA et les paramètres préalablement définis sont disponibles pour la phase suivante.

Listing 5.2 – Phase de configuration d'une TA

```
1
2  TEE_Result TA_CreateEntryPoint(void){
3
4       return TEE_SUCCESS;
5
6    }
7
8    /*Nouvelle session creee, recuperation des parametres et des operations */
9
10   TEE_Result TA_OpenSessionEntryPoint(uint32_t param_types, TEE_Param params[4],
11                                       void **sess_ctx){
12
13     uint32_t exp_param_types = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE, TEE_PARAM_TYPE_NONE,
14                                        TEE_PARAM_TYPE_NONE, TEE_PARAM_TYPE_NONE);
15
16     if(param_types != exp_param_types)
17         return TEE_ERROR_BAD_PARAMETERS;
18
19     (void) &params;
20     (void) &sess_ctx;
21
22     return TEE_SUCCESS;
23
24   }
25
```

```
26    /*Appelee quand la commande est invoquee sur la CA*/
27
28  TEE_Result TA_InvokeCommandEntryPoint(void *sess_ctx, uint32_t cmd_id,
29                      uint32_t param_types, TEE_Param params[4]){
30
31    (void) &sess_ctx;
32
33        cmd_id++;
34        return calculationSHA1(param_types, params);
35
36  }
```

Les opérations à exécuter

Le Code 5.3 permet de visualiser la fonction *calculationSHA1* à exécuter sur la TA. Dans un premier temps, la fonction récupère les paramètres définis par la CA. Ces paramètres sont testés pour vérifier qu'ils correspondent à ceux initialisés par la CA. Si la correspondance n'est pas parfaite, un message d'erreur est transmis à la TA (ligne 20). Dans le cas contraire, une opération est allouée et l'opération de hashage est effectuée avec l'algorithme SHA-1. Comme le paramètre de la fonction est en entrée/sortie, il est utilisé pour transmettre le résultat à la CA pour l'affichage.

Listing 5.3 – Opération à exécuter

```
1
2   static TEE_Result calculationSHA1(uint32_t param_types,
3                                     TEE_Param params[4])
4   {
5           TEE_OperationHandle operation = NULL;
6           TEE_Result res = TEE_ERROR_GENERIC;
7           uint32_t size1 = 20;
8           uint32_t digest_len;
9           uint8_t digest[size1];
10          char *message = NULL;
11          int message_len = 0;
12          uint32_t exp_param_types = TEE_PARAM_TYPES(TEE_PARAM_TYPE_MEMREF_INPUT,
13                                                    TEE_PARAM_TYPE_MEMREF_OUTPUT,
14                                                    TEE_PARAM_TYPE_NONE,
15                                                    TEE_PARAM_TYPE_NONE);
16          message = params[0].memref.buffer;
17          message_len = params[0].memref.size;
18
19          if (param_types != exp_param_types || !message)// || !digest)
20                  return TEE_ERROR_BAD_PARAMETERS;
21
22          res = TEE_AllocateOperation(&operation, TEE_ALG_SHA1, TEE_MODE_DIGEST, 0);
23
24
25          if (res != TEE_SUCCESS) {
26                  DMSG("TEE_AllocateOperation failed! res: 0x%x", res);
27                  goto out;
28          }
29
30          res = TEE_DigestDoFinal(operation, message , message_len, digest, &digest_len);
```

```
31
32          if (res != TEE_SUCCESS) {
33                  DMSG("TEE_DigestDoFinal failed! res: 0x%x", res);
34                  goto out;
35          }
36  out:
37          if (operation)
38                  TEE_FreeOperation(operation);
39
40
41          params[0].memref.buffer = digest;
42          params[0].memref.size = digest_len;
43          return res;
44  }
```

Après avoir présenté l'environnement de développement OP-TEE ainsi que le processus de développement d'une application sur OP-TEE, nous passons dans la suite de ce chapitre à l'évaluation en terme de performances et en terme de sécurité de l'algorithme d'authentification proposé.

5.2 Évaluation de performances

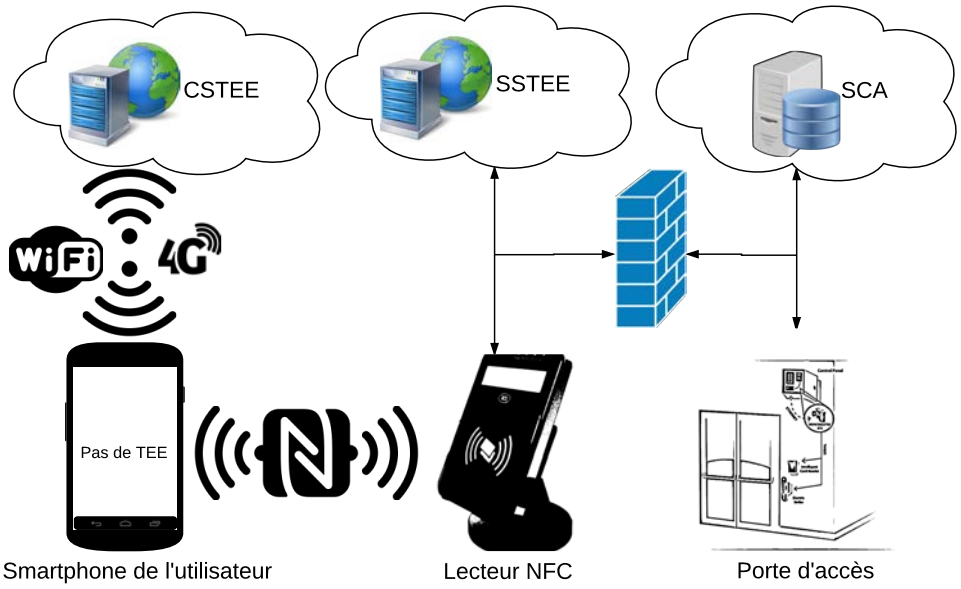

FIGURE 5.4 – Architecture proposée pour un contrôle d'accès sécurisé sans contact

Dans un premier temps, nous effectuons l'évaluation de performances de l'architecture décrite par la Figure 5.4 que nous avons proposée. Nous nous concentrons dans cette évaluation au temps consommé par l'algorithme d'authentification depuis le début jusqu'à la fin de la négociation de la clé de session qui permet l'envoi sécurisé des données d'authentification.

Ce temps dépend en majeure partie du temps d'exécution des opérations cryptographiques qui, mises bout-à-bout, reconstituent le déroulement du protocole d'authentification de part et d'autre. De plus, la faible taille des paquets d'authentification, 192 octets, rend le temps de communication au sein du réseau négligeable devant le temps d'exécution des opérations cryptographiques. En effet, la liaison la plus lente de l'architecture se trouve être la communication NFC avec un débit maximum de 424kb/s ce qui signifie que le temps d'acheminement d'un paquet d'authentification au travers de cette connexion est autour d'une milliseconde.

5.2.1 Environnements d'évaluation

Nous procédons à l'évaluation du protocole proposé sur l'architecture de la Figure 5.4 en utilisant pour le CSTEE et le SSTEE un environnement TEE virtualisé (OP-TEE) où la plate-forme et l'OS sont virtualisés sur FVP [92] et un TEE matériel qui est la carte de développement ARM JUNO R0.

5.2.1.1 TEE virtualisé

Le TEE virtualisé consiste en une machine dôtée d'un processeur Intel I7 avec 1 Gbits de mémoire volatile RAM (Random Access Memory). Cette machine exécute un hyperviseur développé par ARM appelé Fixed Virtual Platform [92] (FVP). FVP permet de simuler un système ARM complet incluant le processeur, la mémoire et les périphériques d'entrées/sorties. Le système ARM supporté par FVP permet d'avoir un environnement de test performant et réaliste. Il est notamment utilisé par les développeurs pour le test d'applications, de firmware et de pilote bien avant de pouvoir disposer du matériel physique. Il permet aussi de travailler sur une large panoplie de matériel et de tester en amont la compatibilité des logiciels avec les différentes plate-formes cibles.

5.2.1.2 TEE physique

Le TEE physique utilise la carte de développement qu'ARM met à la disposition des développeurs afin de pouvoir tester les applications sécurisées sans contrainte de la signature de l'application par le fabriquant du matériel. Dans notre cas, nous déployons le secure OS OP-TEE sur la carte ARM JUNO afin de disposer d'un TEE complet. Cette solution nous permet de disposer de la capacité TrustZone des processeurs ARM y compris le stockage et l'exécution sécurisés. La carte ARM JUNO R0 est décrite en détail dans la section 5.1.2.

5.2.2 Résultats expérimentaux

Nous nous concentrons dans notre évaluation sur les opérations cryptographiques exécutées dans le cadre du protocole d'authentification que nous avons proposé en section 4.5. Pour ce protocole, nous nous intéressons à l'évaluation de :

— La génération de clés pour chaque identité,

— Le chiffrement IBE des paquets d'authentification,
— La génération des challenges pour IBAKE,
— Le temps de chiffrement à l'aide des courbes elliptiques (ECC) des données d'authentification,
— Le temps de chiffrement AES des données d'authentification.

5.2.2.1 Génération des clés de chiffrements IBE

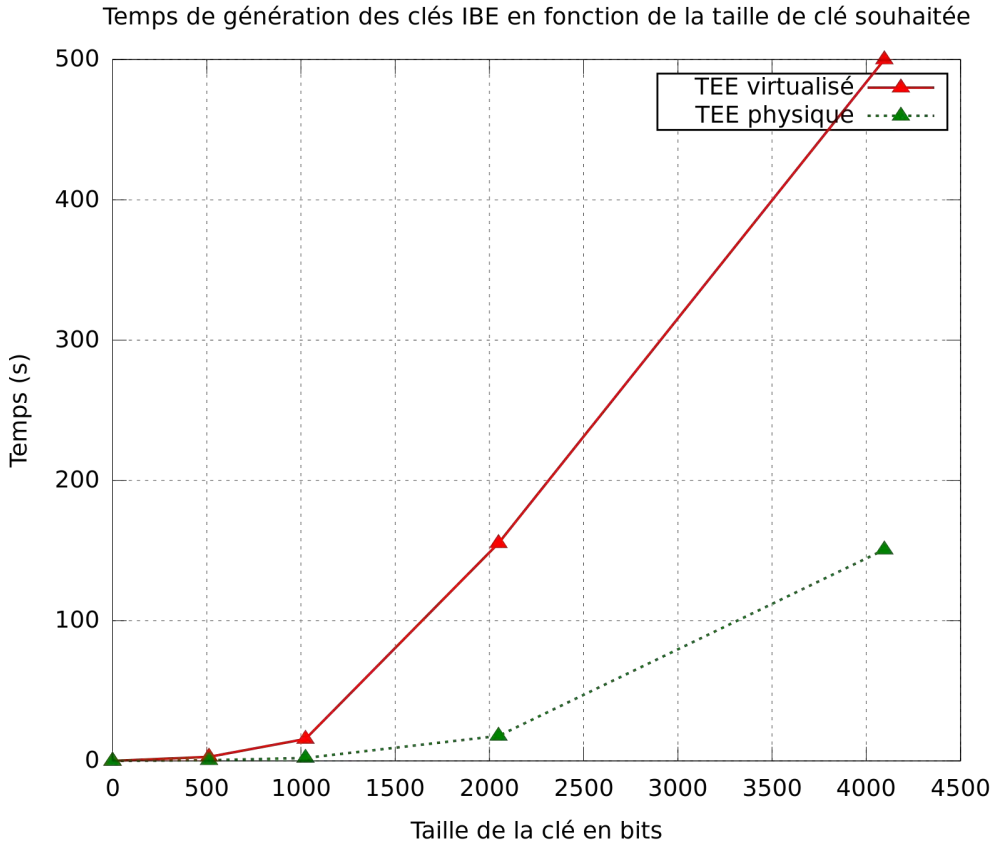

FIGURE 5.5 – Temps de génération de clés IBE de Callas

La Figure 5.5 montre le temps d'exécution nécessaire pour la génération de la paire de clés IBE pour chacune des identités. Rappelons à ce stade que cette génération se fait à l'aide d'un générateur RSA déterministe permettant pour une identité définie d'avoir la même clé publique associée. Nous observons que le temps de génération augmente de manière exponentielle en fonction de la taille de clé souhaitée. Par exemple, pour une taille de clé de 1024 bits, le TEE virtualisé consomme aux alentours de 16 secondes de temps d'exécution tandis que le TEE physique consomme 2.09 secondes. Le différentiel de performances entre les deux environnement

est important. En effet, le TEE virtualisé prend 8 fois plus de temps que le TEE physique pour la même opération. Pour une taille de 2048 bits, le TEE virtualisé prend 10 fois plus de temps que le TEE physique pour la génération de la paire de clés. Cependant, ces temps ne sont pas en adéquation avec la contrainte temps réel du protocole d'authentification. Pour cette raison, nous préconisons une génération de clé hors ligne pour chaque utilisateur du système. En effet, la faculté du CSTEE à stocker ces clés de manière sécurisée nous permet d'envisager ce scénario.

5.2.2.2 Temps d'exécution de l'opération $x * P$ et $y * P$

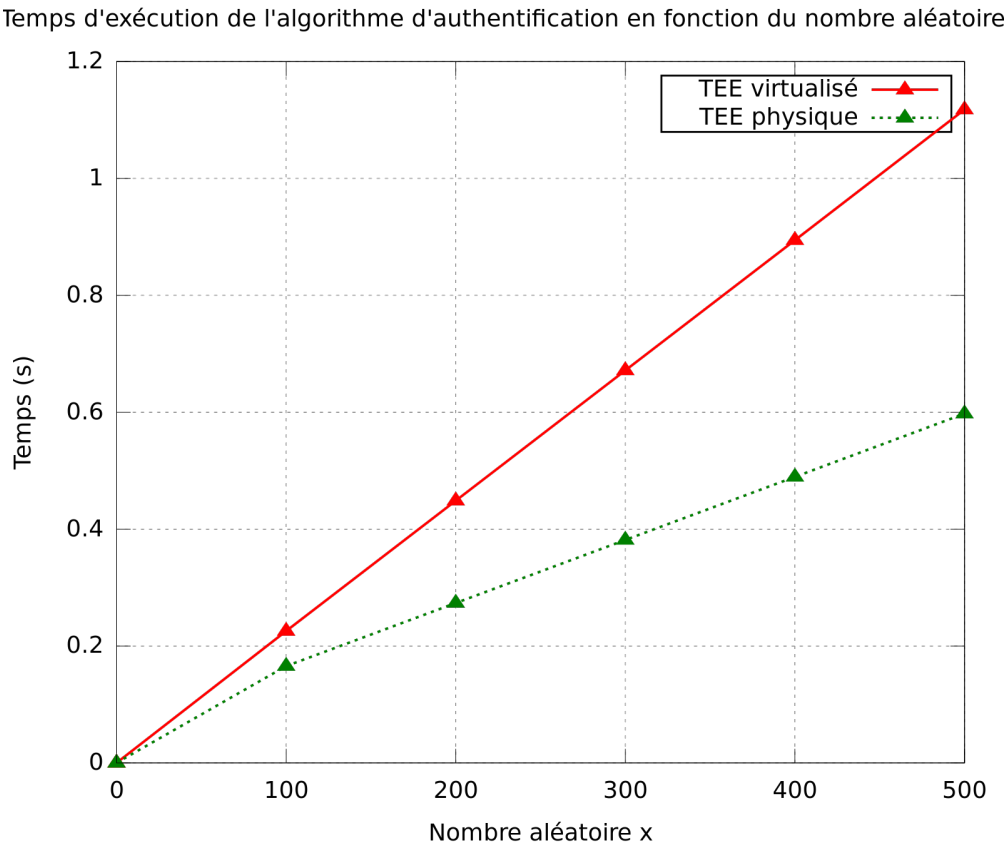

FIGURE 5.6 – Temps d'exécution de l'opération $x * P$ et $y * P$

La Figure 5.6 montre le temps d'exécution nécessaire à chacun des deux environnements pour calculer le résultat de la multiplication $x * P$ et $y * P$ de la partie IBAKE du protocole. Dans le contexte du protocole d'authentification, P représente un point de la courbe elliptique (paramètre public) et x représente un nombre tiré de manière aléatoire qui sera multiplié par ce point pour former le challenge du protocole. Nous remarquons que pour le TEE virtualisé, le temps d'exécution évolue de manière linéaire en fonction de la valeur de x. Pour une valeur

de $x < 400$, ce temps est inférieur à la seconde. Les performances avec le TEE physique sont deux fois meilleures que celles avec le TEE virtualisé. Par exemple pour une valeur de $x = 500$, le temps d'exécution de l'opération est de 1.1 secondes tandis qu'il est de 0.59 seconde pour le TEE physique. Notons que ces temps représentent l'ensemble des calculs $x * P$ et $y * P$ effectués durant l'exécution de l'algorithme d'authentification proposé.

5.2.2.3 Chiffrement IBE avec une clé 2048 bits en fonction des données

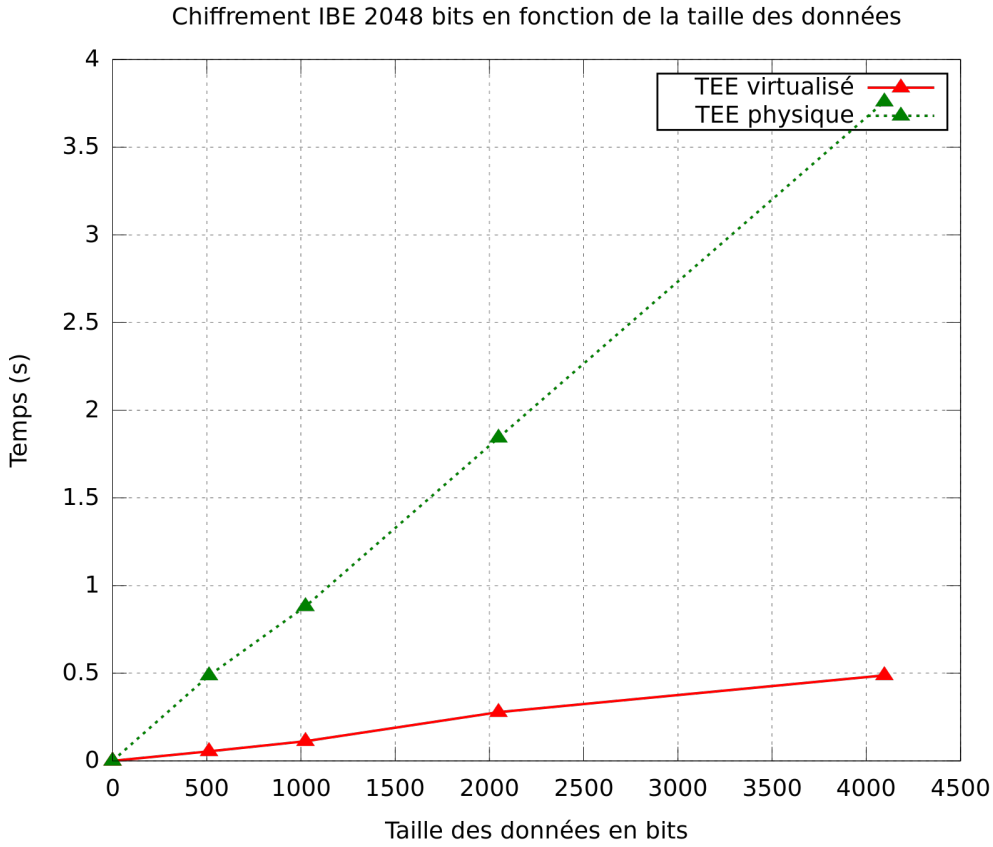

FIGURE 5.7 – Temps d'exécution du chiffrement IBE en fonction de la taille des données

La Figure 5.7 montre le temps de chiffrement IBE de Callas [87] (basé sur RSA) avec une clé de 2048 bits en fonction de la taille des données en entrée sur les deux environnements. Chaque paquet d'authentification échangé est chiffré avec la clé publique du destinataire. Pour le TEE virtualisé, le temps de chiffrement d'un paquet d'authentification d'authentification de 1024 bits (128 octets) est de 0.8 secondes alors que le TEE physique effectue la même opération en 0.1 secondes. Les performances affichées par le TEE physique sont très acceptables par rapport au niveau de sécurité souhaité. De plus, avec le TEE physique il est possible de faire passer la taille

du paquet au double ou au quadruple en restant toujours sous la demi seconde.

5.2.2.4 Chiffrement des données d'authentification avec la clé de session

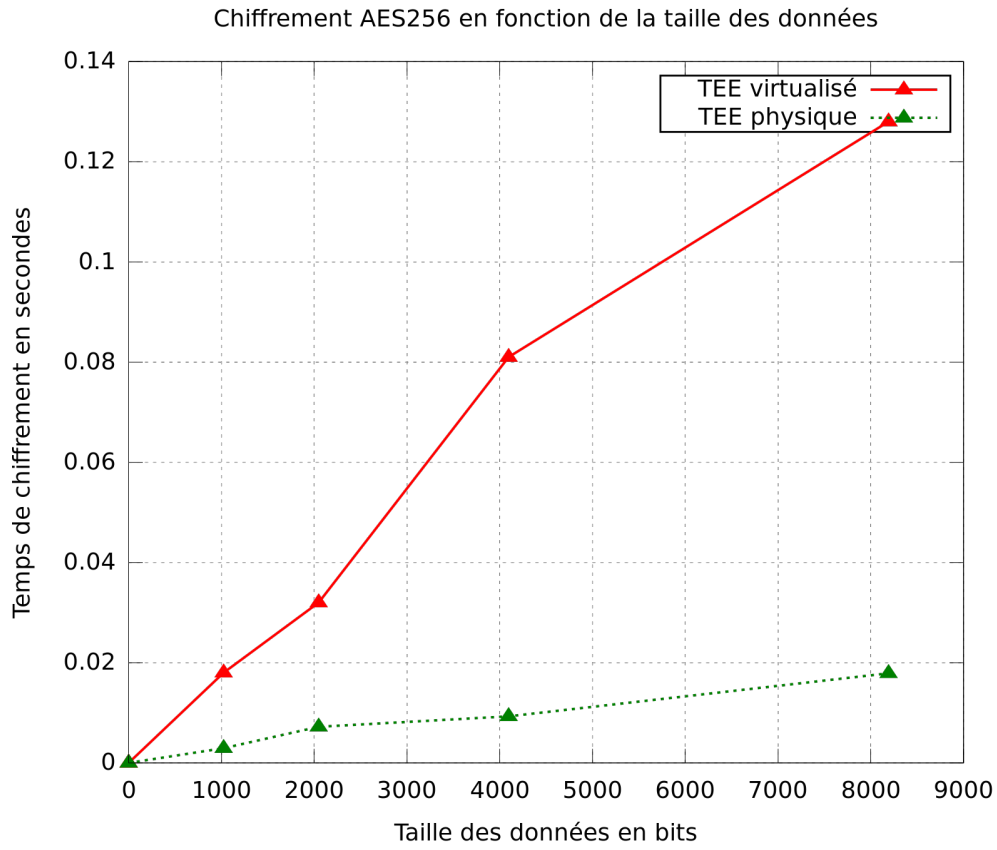

FIGURE 5.8 – Temps d'exécution du chiffrement AES256 en fonction de la taille des données

Une fois que le CSTEE et le SSTEE ont fini la négociation de la clé de session $x*y*P$, l'utilisateur envoie ses données d'authentification au SCA pour demander l'accès au bâtiment sensible. Ces données d'authentification sont chiffrées avec un AES-256 pour éviter qu'un attaquant puisse les intercepter et les réutiliser. La clé de ce chiffrement est dérivée à partir du secret $x * y * P$. La Figure 5.8 permet de montrer le temps de chiffrement AES-256 pris par chacun des deux environnements pour chiffrer les données d'authentification. Nous remarquons qu'avec le TEE virtualisé, le fait de chiffrer 4096 bits de données ne prend que 0.08 secondes ce qui se révèle être très rapide en comparaison avec le temps global du protocole. Comme prévu, le TEE physique est bien plus rapide que son alter ego virtualisé. En effet, pour la même taille de données (4096 bits), le temps d'exécution est divisé par 7 (environ 0.012 secondes).

Le secret négocié entre le CSTEE et le SSTEE peut être aussi utilisé pour dériver une clé d'un

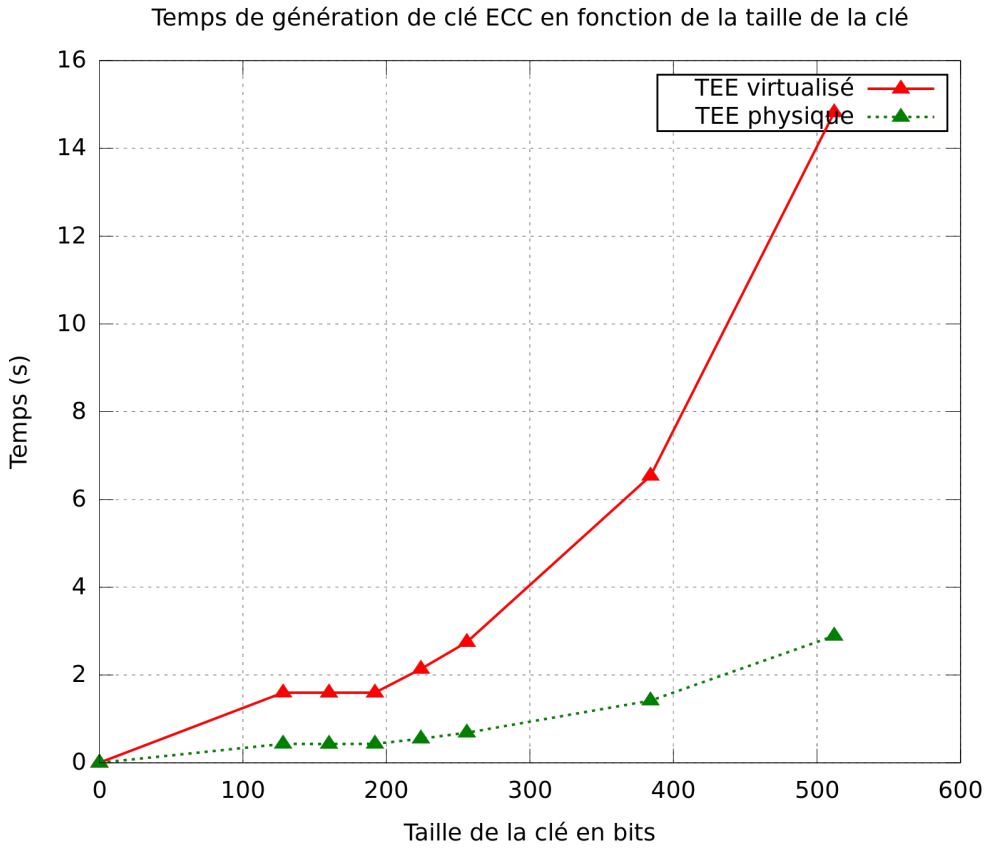

FIGURE 5.9 – Temps de génération d'une clé ECC en fonction de la taille de la clé

chiffrement ECC. Afin d'avoir une base de comparaison, nous avons étudié la possibilité de chiffrer les données d'authentification non pas avec un algorithme classique AES-256 mais avec un chiffrement ECC car la taille des paquets est très faible. La Figure 5.9 permet de montrer le temps de génération d'une clé ECC à partir du secret négocié $x * y * P$ en fonction des différentes tailles de clés possibles. Nous remarquons que le temps de génération augmente de manière quasi exponentielle pour le TEE virtualisé pour atteindre quasiment les deux secondes pour une taille de clé de 256 bits. Pour cette même taille de clé, le TEE physique prend 0.6 secondes environ pour le processus de génération de la clé. Bien qu'avec le TEE physique les performances sont nettement meilleures, il n'est pas envisageable d'intégrer ce type de chiffrement au sein de notre protocole d'authentification au vu de ces faibles performances.

Après la phase de génération de clé, nous avons évalué la partie chiffrement de l'algorithme ECC. La Figure 5.10 permet de montrer les performances du chiffrement ECC pour une donnée de 16 octets (l'identité de l'utilisateur) pour différentes tailles de clés. Nous remarquons que le temps de chiffrement augmente de manière significative en fonction de la taille de la clé

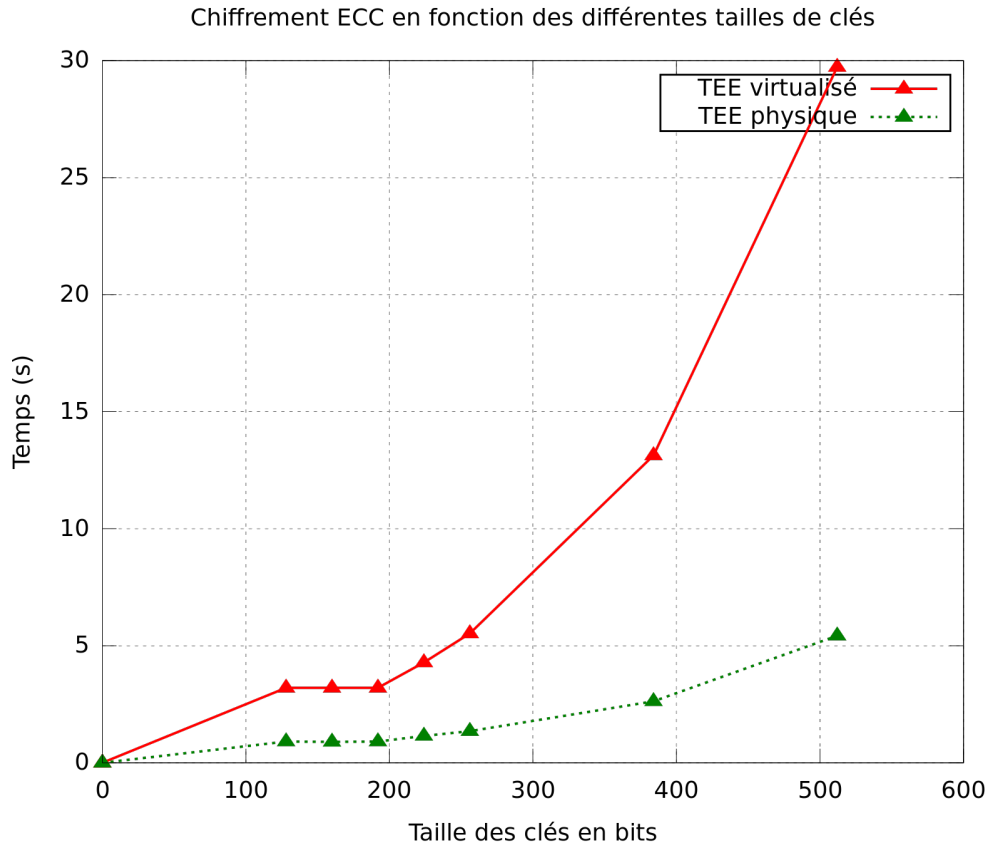

FIGURE 5.10 – Temps de chiffrement ECC en fonction de la taille de la clé

considérée. Le TEE physique présente des performances 5 fois meilleures que le TEE virtualisé. Cependant, il est loin d'approcher les performances afficher par AES-256 même en présence d'une donnée de très petite taille (16 octets). Après avoir analysé ces résultats, nous observons que même pour une taille de paquet très réduite, le chiffrement ECC n'est pas une solution viable dans le cadre de notre protocole de contrôle d'accès. L'utilisation de l'algorithme de chiffrement symétrique AES-256 reste la meilleure option en offrant de très bonnes performances.

5.2.2.5 Analyse des résultats

Nous remarquons au vu des résultats obtenus que la meilleure solution consiste à déployer un TEE physique sur le CSTEE et le SSTEE et d'utiliser l'algorithme AES-256 après avoir négocier la clé de session. Nous observons aussi que la configuration de la solution virtualisée déployée est proche de celle d'un smartphone mais ne permet pas d'avoir des performances acceptables pour un déploiement temps réel. Avec l'utilisation de FVP, nous savions que les performances du TEE virtualisé seraient inférieures à celles affichées par le TEE physique. Cependant, vu

les promesses affichées par les développeurs de FVP avec de performances très proches du natif, nous pensions que le différentiel serait beaucoup plus faible et que la solution serait utilisable pour du temps réel en faisant un compromis sécurité/performance. Cependant, après les évaluations que nous avons menées, nous nous sommes rendus compte que ce différentiel de performances était bien plus important que prévu.

En s'appuyant sur les mesures précédentes, il est possible d'estimer le temps global pris par l'exécution du protocole d'authentification en entier. En effet, nous avons 6 opérations de chiffrement IBE prenant chacune 0.1 seconde (pour un paquet de 128 octets) donnant un total de 0.6 secondes. Nous devons y ajouter la négociation du secret partagé via IBAKE nécessitant dans le pire cas considéré 0.6 secondes. Pour finir, le chiffrement via AES-256 des données d'authentification prend moins de 0.01 secondes pour un temps global inférieur à 1.2 secondes ce qui en fait un algorithme efficace et bâti pour faire du contrôle d'accès sécurisé temps réel.

5.3 Évaluation de sécurité

Après avoir effectué l'évaluation de performances de l'algorithme d'authentification sur l'architecture proposée (Figure 5.4), nous passons à l'évaluation en terme de sécurité du protocole proposé basée sur l'outil Scyther [9]. Cet outil fait partie des vérificateurs les plus performants devant Avispa [93] et CryptoVerif [94] avec un taux de détection d'attaques quasi-identique [95].

5.3.1 Présentation de Scyther

Scyther est un outil clé en main avec une interface graphique pour la vérification et l'analyse de protocoles cryptographiques. L'outil Scyther utilise une approche de vérification de modèles. A partir de ces modèles, il génère l'ensemble de toutes les traces d'exécution possibles puis il les classe et vérifie qu'aucune ne peut contenir d'attaques. Une trace correspond à un ensemble partiellement ordonné d'événements qui se produisent lors de l'exécution du protocole. L'occurrence des événements au sein du protocole est vérifiée en les faisant correspondre aux critères énoncés dans la sémantique du protocole.

Nous allons dans ce qui suit présenter brièvement le langage de description du protocole ainsi que l'algorithme de vérification de Scyther.

5.3.1.1 Langage de description du protocole

L'outil Scyther prend en entrée un fichier *.spdl* [1] décrivant les spécifications du protocole et les propriétés de sécurité revendiquées. La syntaxe du langage utilisée par Scyther est un mélange entre C et Java. Scyther utilise un modèle formel définissant le protocole sous forme de rôles exécutant une séquence d'événements.

1. Security protocol description language

Un fichier d'entrée simple, décrivant un protocole avec deux rôles A et B envoyant deux messages sous forme de chaîne de caractères peut être modélisé comme suit :

Listing 5.4 – Code simple Scyther

```
1  protocol SimpleProtocol (A,B) {
2  role A {
3    send_1(A,B,'ping');
4    recv_2(B,A,'pong');
5      }
6
7  role B {
8    recv_1(A,B,'ping');
9    send_2(B,A,'pong');
10 }
11 };
```

Comme le montre le Code 5.4, les événements sont soit des envois soit des réceptions de messages ou des propriétés de sécurité revendiquées. A chaque événement d'envoi doit correspondre un événement de réception. Si ce n'est pas le cas, Scyther refusera de compiler le code source. Cependant, si un envoi ou une réception unique doit être défini, il est possible de l'exprimer en ajoutant un point d'exclamation '!' à l'événement comme suit :

Listing 5.5 – Envoi unique

```
send_!(A, B, secretKey);
```

Les termes atomiques sont décrits comme des chaines de caractères ou des chaines alpha-numériques représentant des identifiants (constantes, graines fraichement générées, variables etc.). Ces termes peuvent être combinés pour décrire des opérations complexes telles que le chiffrement, le déchiffrement et le hash. Si un terme devient très complexe, il est possible d'utiliser des macros qui se substitueront à une longue chaîne de termes comme sur l'exemple suivant :

Listing 5.6 – Substitution par macro

```
macro m1 = hash h(A, B, nonce1, terme1, terme2);
```

Toute la chaine de termes décrite par le code 5.6 peut être remplacée par la macro *m1* par exemple.

N'importe quel terme peut agir comme une fonction de chiffrement symétrique. Par exemple, le terme *nikir* fait référence au chiffrement du terme *ni* par la clé *kir*. De plus, une infrastructure de clé symétrique est prédéfinie, permettant l'utilisation de la clé par défaut $k(A, B)$ comme clé partagée entre les rôles A et B. Par exemple, le Code 5.7 représente l'envoi du terme $n1$ de A vers B chiffré avec la clé symétrique partagée entre A et B

Listing 5.7 – Chiffrement du terme n1 par la clé partagée entre A et B

```
send_1(A, B, {n1}k(A,B));
```

Types prédéfinis et types utilisateurs

Scyther offre plusieurs types prédéfinis prêt à l'usage : les *Agents*, les *Functions* (Définies comme fonctions de termes, prend une liste de paramètres en entrée, sont de type hash par défaut), les *Nonces* (Graines) fraîchement générées pour les fonctions de hashage) et les *Tickets* (type pouvant être remplacé par n'importe quel type de variable). De nouveaux types peuvent être définis comme des *Usertypes*.

Les évènements 'claim'

Les propriétés de sécurité sont définies à l'aide d'événements spéciaux appelés *claims* qui font partie de la définition d'un rôle. Les agents (chaque rôle) ont une vision locale des états du système, basée sur les messages reçus. Les propriétés sont revendiquées à partir de cette vision locale et donc ne sont pas valides que depuis le point de vue d'un agent spécifique exécutant une description précise. Il existe plusieurs types de *claim* possibles. Les plus intéressants pour notre étude de cas sont :

— **Confidentialité** : la notation *claim*(*Initiator*, *secret*, *ni*) permet de déclarer que le terme *ni* est secret depuis la perspective du rôle de l'intiateur. Il est aussi possible de déclarer un terme secret *SessionKey* explicitement comme clé de session en utilisant le terme 'SKR'[1] en écrivant *claim*(*Initiator*, *SKR*, *SessionKey*). Cette revendication sera déclarée fausse si un adversaire peut exécuter une attaque révélant le contenu de cette clé de session.

— **Authentification** : la revendication *claim*(*R*, *Alive*, *R'*) demande la réactivité du rôle R' depuis la vision locale du rôle R. Cette propriété impose à R' de 'discuter' avec R, donc R' a envoyé un message à R, incluant un secret que seul R' peut connaître. Cette propriété ne permet de faire aucune hypothèse sur la connaissance de R' du protocole d'authentification exécuté. Une forme plus forte d'authentification consisterait à demander le *claim*(*R*, *Weakagree*, *R'*) ce qui demanderait en plus à R' qu'il soit entrain d'exécuter le protocole d'authentification avec R.

5.3.1.2 Algorithme de vérification de Scyther

L'algorithme de recherche vers l'arrière[2] de Scyther est un algorithme de raffinement de modèle appliquant une distinction de cas sur la source des messages pour permettre la résolution des contraintes. Cette distinction est possible grâce à la nature des événements d'envoi et de réception. En effet, ces derniers prennent en paramètre obligatoirement la source du message, sa destination ainsi que son contenu. De manière générale, les modèles sont vérifiés à l'aide d'outils

1. Session Key Reveal
2. L'algorithme de recherche peut être borné avec une valeur à définir afin de limiter l'espace d'états

de vérification de modèles automatisés pour vérifier, par exemple, la violation d'une contrainte de confidentialité sur le contenu d'une variable. Ceci est réalisé par la définition d'un ensemble de traces d'exécution représentant la contradiction de la propriété de sécurité et d'essayer de prouver l'occurrence d'une de ces traces contradictoires.

Par exemple dans le cas d'une violation de confidentialité, cela correspondrait à la connaissance par un attaquant du contenu d'une variable devant rester secrète dans la définition sémantique du protocole.

Il existe trois possibilités d'arrêt de l'algorithme de recherche de Scyther :

— Une correspondance de trace d'exécution est établie sur le protocole, ce qui veut dire que le modèle est réalisable. La trace peut correspondre à une exécution saine ou à une attaque. Si elle correspond à une attaque, cela signifie qu'une attaque est possible et qu'une trace avec une taille minimale peut être sélectionnée à partir de l'ensemble des traces générées pour le protocole pour construire un contre exemple aux revendications de sécurité de ce protocole.

— Aucune correspondance de trace n'est établie et aucune limite de l'algorithme de recherche n'est atteinte. Nous en déduisons que la réalisation de cette trace est impossible. Si la trace appartient à l'ensemble des attaques possibles générées, nous en déduisons qu'aucune attaque de ce genre n'est possible sur le protocole défini.

— Aucune correspondance, mais une borne limite [1] atteinte. La vérification de la propriété n'est valide que pour le nombre limite de sessions bornées défini en amont.

Comme résultat de l'évaluation d'un protocole par Scyther, un résumé de la vérification ou de la violation des propriétés de sécurité est affiché. Il est possible d'afficher des graphes visuels des constructions des attaques possibles. La configuration par défaut de Scyther définie une borne afin de garantir la terminaison certaine de l'algorithme de recherche. Il est aussi possible de borner manuellement le processus de recherche. Dans ce cas, si la borne est atteinte sans que l'algorithme ne trouve d'attaque possible, la mention '*No attack within bound*' est affichée à l'utilisateur.

5.3.2 Vérification du protocole proposé sur les différentes architectures

Après avoir présenté l'outil de vérification Scyther, nous abordons la phase d'évaluation du protocole que nous avons proposé sur les différentes architectures présentées en section 4.3.

5.3.2.1 Architecture TEE embarqué

Pour cette architecture présentée en Figure 5.11, nous disposons d'un smartphone (ligne 23 à 37) NFC possédant un TEE qu'il utilise pour exécuter l'algorithme d'authentification. De l'autre côté, nous disposons d'un lecteur intelligent sous la forme d'un Raspberry Pi 3 (SSTEE ligne 8 à 21) disposant de la capacité TrustZone d'ARM et pouvant communiquer en NFC

1. Cette limite correspond au nombre d'occurrence d'un rôle ou nombre de sessions du protocole

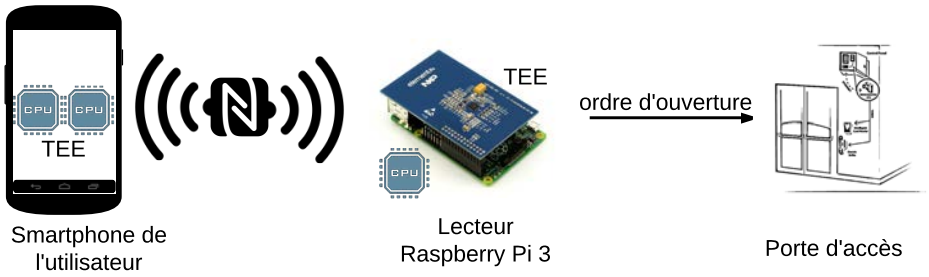

FIGURE 5.11 – Scénario 1 : Architecture TEE embarqué

avec le smartphone et répondre à l'algorithme d'authentification. Le code Scyther modélisant l'algorithme d'authentification déployé est présenté par le Code 5.8. Chacun des rôles exécute le protocole d'authentification proposé. Les événements *send* et *recv* représentent les échanges de paquets d'authentification.

Listing 5.8 – Code de vérification du scénario 1

```
hashfunction integrity;
hashfunction mult;

protocol secureAccess(smartphone,SSTEE){

const P;

role SSTEE
{
  fresh x: Nonce;
  var yprime: Nonce;
  send_1(SSTEE, smartphone, {SSTEE,smartphone
      ,mult(x,P)}pk(smartphone));

  recv_2(smartphone, SSTEE,
            {SSTEE,smartphone,mult(x,P),
                mult(yprime,P)}pk(SSTEE));
  claim_i1(SSTEE, SKR, mult(x, mult(yprime,P)));
  claim_i3(SSTEE, Weakagree, smartphone);
  claim_i5(SSTEE, Nisynch);
}

role smartphone
{
  var xprime: Nonce;
 var idsmartphone;
  fresh y: Nonce;
  recv_1(SSTEE, smartphone,
  {SSTEE,idsmartphone,mult(xprime,P)}pk(smartphone));
  match(smartphone, idsmartphone);
  send_2(smartphone, SSTEE,
          {SSTEE,smartphone,mult(xprime,P),
            mult(y,P)}pk(SSTEE));

  claim_i2(smartphone, SKR, mult(xprime,mult(y,P)));

}
}
```

Le résultat de la vérification du protocole par Scyther est représenté par la Figure 5.12. Nous remarquons que la clé de session reste secrète (*claim* de la ligne 18 et 35 du Code 5.8) tout au long du processus d'authentification et qu'un attaquant n'a pas la possibilité de récupérer cette clé. Ce résultat est très important pour s'assurer que les données d'authentification envoyées chiffrées avec cette clé ne peuvent pas être interceptées ni réutilisées et qu'aucune fuite de ces données n'est possible. Il est aussi garanti que tous les messages reçus par le lecteur intelligent représenté par le Raspberry Pi 3 ont été bien envoyés par le smartphone.

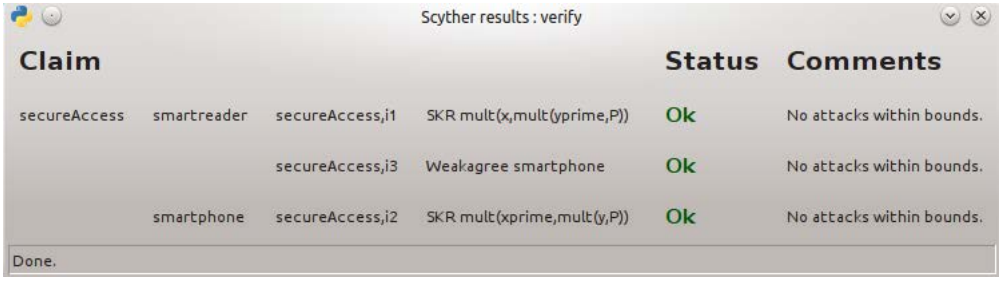

FIGURE 5.12 – Validation du protocole proposé pour le scénario 1

5.3.2.2 Architecture semi-connectée

La Figure 5.13 présente l'architecture semi-connectée vérifiée par le Code 5.9. Cette architecture est composée de trois rôles : le smartphone, le lecteur NFC standard et le serveur SSTEE pour le TEE déporté. Le smartphone et le SSTEE exécutent le protocole d'authentification proposé au travers des événements *send et recv* présents dans chacun des codes. Nous observons aussi que le lecteur a pour rôle de 'forwarder' les paquets entre les deux entités.

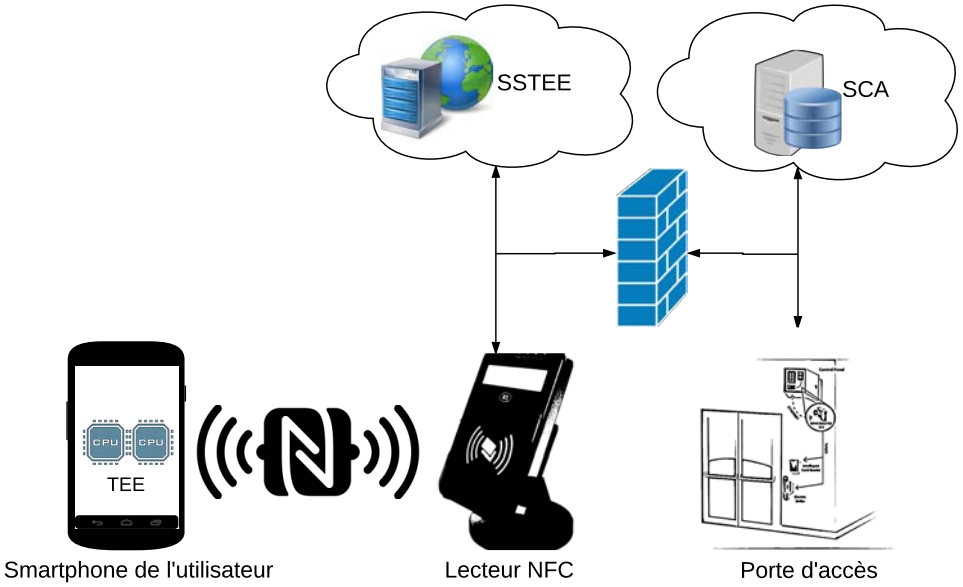

FIGURE 5.13 – Scénario 2 : Architecture semi-connectée

Listing 5.9 – Code de vérification du scénario 2

```
1  hashfunction integrity;
2  hashfunction mult;
3
4  protocol secureAccess(smartphone, reader,SSTEE){
5
6  const P;
7
8  role SSTEE
9  {
10   fresh x: Nonce;
11   var yprime: Nonce;
12   send_1(SSTEE, reader, {reader,SSTEE,smartphone,mult(x,P),
13         integrity(mult(x,P))}pk(smartphone));
14     recv_4(reader, SSTEE, {reader,SSTEE,smartphone,mult(x,P),
15         mult(yprime,P)}pk(SSTEE));
16   claim_i2(SSTEE, Secret, mult(yprime,mult(x,P)));
17   claim(SSTEE, Weakagree, smartphone);
18
19 send_5(SSTEE,reader, {SSTEE}mult(yprime,mult(x,P)));
20 }
```

```
21
22 role reader
23 {
24 var x: Nonce;
25 var y: Nonce;
26  recv_1(SSTEE, reader, {reader,SSTEE,smartphone,mult(x,P),
27          integrity(mult(x,P))}pk(smartphone));
28  send_2(reader, smartphone, {reader,SSTEE,smartphone,mult(x,P),
29            integrity(mult(x,P))}pk(smartphone));
30  recv_3(smartphone, reader, {reader,SSTEE,smartphone,mult(x,P),
31        mult(y,P)}pk(SSTEE));
32  send_4(reader, SSTEE, {reader,SSTEE,smartphone,
33              mult(x,P), mult(y,P)}pk(SSTEE));
34
35
36  recv_5(SSTEE,reader, {SSTEE}mult(y,mult(x,P)));
37  send_6(reader,smartphone, {SSTEE}mult(y,mult(x,P)));
38  claim_i3(reader, Secret,  mult(y,mult(x,P)));
39
40 }
41
42 role smartphone
43 {
44   var xprime: Nonce;
45   fresh y: Nonce;
46   recv_2(reader, smartphone, {reader,SSTEE,smartphone,
47              mult(xprime,P), integrity(mult(xprime,P))}pk(smartphone));
48
49   send_3(smartphone, reader, {reader,SSTEE,smartphone,
50            mult(xprime,P), mult(y,P)}pk(SSTEE));
51   recv_6(reader,smartphone, {SSTEE}mult(y,mult(xprime,P)));
52
53   claim_i1(smartphone, Secret, mult(y,mult(xprime,P)));
54
55 }
56 }
```

Le résultat de la vérification du protocole par Scyther (les *claim* des lignes : 16 pour le SSTEE, 36 du lecteur et 51 du smartphone) est représenté par la Figure 5.14. Nous émettons l'hypothèse que le canal de communication entre le lecteur et le SSTEE est sécurisé. En effet, même dans

l'infrastructure actuelle les lecteurs sont directement câblés avec le serveur de contrôle d'accès à travers une liaison Ethernet que l'on assume être sécurisée. Pour modéliser ce réseau que nous supposons sécurisé, nous utilisons une clé de session entre le lecteur et son SSTEE. Nous observons que la clé de session négociée à travers le protocole d'authentification reste secrète (voir *claim* des lignes 16, 36 et 51 du Code 5.9) et qu'un attaquant ne peut y accéder pour intercepter l'envoi des données d'authentification. De plus, nous pouvons certifier qu'aucune injection de paquet n'est possible au niveau du lecteur NFC qui transfère les messages du smartphone vers le SSTEE et inversement. Nous pouvons aussi affirmer que SSTEE a finalisé l'exécution du protocole d'authentification avec le smartphone qui a de son côté fini d'exécuter sa partie du protocole.

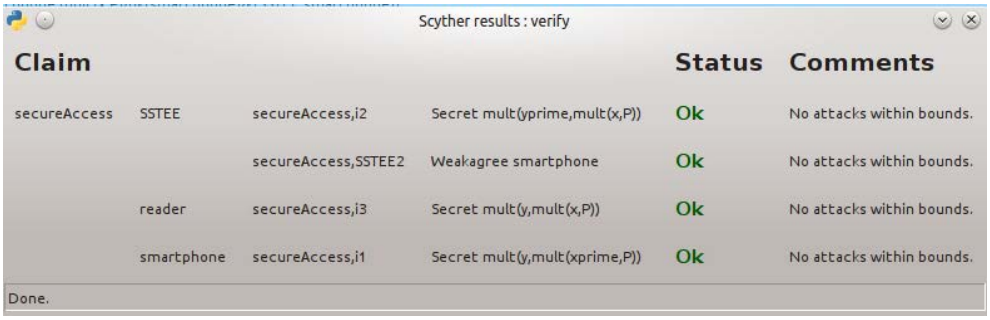

FIGURE 5.14 – Validation du protocole proposé pour le scénario 2

5.3.2.3 Architecture entièrement connectée

Nous procédons maintenant à l'évaluation de sécurité du scénario nominal représenté par la Figure 5.4 rappelée en section 5.2.

Listing 5.10 – Code de vérification du scénario nominal

```
1  hashfunction integrity;
2  hashfunction mult;
3
4  protocol secureAccess(smartphone, reader,SSTEE, CSTEE){
5
6  const P;
7
8
9  role SSTEE
10 {
11   fresh x: Nonce;
```

```
12    var yprime: Nonce;
13    send_1(SSTEE, reader, {{reader,SSTEE,smartphone,CSTEE,
14            mult(x,P)}pk(smartphone)}k(SSTEE,reader));
15    recv_6(reader, SSTEE, {{reader,SSTEE,smartphone,CSTEE,
16            mult(x,P), mult(yprime,P)}pk(SSTEE)}k(SSTEE,reader));
17    claim_i2(SSTEE, Secret, mult(yprime,mult(x,P)));
18    claim(SSTEE, Weakagree, CSTEE);
19  }
20
21  role reader
22  {
23    var x: Nonce;
24    var y: Nonce;
25    recv_1(SSTEE, reader, {{reader,SSTEE,smartphone,CSTEE,
26            mult(x,P)}pk(smartphone)}k(SSTEE,reader));
27    send_2(reader, smartphone, {reader,SSTEE,smartphone,
28            CSTEE,mult(x,P)}pk(smartphone));
29    recv_5(smartphone, reader, {reader,SSTEE,smartphone,
30            CSTEE,mult(x,P), mult(y,P)}pk(SSTEE));
31    send_6(reader, SSTEE, {{reader,SSTEE,smartphone,
32            CSTEE,mult(x,P), mult(y,P)}pk(SSTEE)}k(SSTEE,reader));
33
34
35
36  }
37
38  role smartphone
39  {
40    var x: Nonce;
41    var y: Nonce;
42    recv_2(reader, smartphone, {reader,SSTEE,smartphone,
43          CSTEE,mult(x,P)}pk(smartphone));
44    send_3(smartphone, CSTEE, {{reader,SSTEE,smartphone,
45          CSTEE,mult(x,P)}pk(smartphone)}k(smartphone, CSTEE));
46    recv_4(CSTEE, smartphone, {{reader,SSTEE,smartphone,
47          CSTEE,mult(x,P), mult(y,P)}pk(SSTEE)}k(smartphone,CSTEE));
48    send_5(smartphone, reader, {reader,SSTEE,smartphone,
49          CSTEE,mult(x,P), mult(y,P)}pk(SSTEE));
50
```

```
51
52
53  }
54
55
56
57  role CSTEE
58  {
59    var xprime: Nonce;
60    fresh y: Nonce;
61    recv_3(smartphone, CSTEE, {{reader,SSTEE,smartphone,
62            CSTEE,mult(xprime,P)}pk(smartphone)}k(smartphone, CSTEE));
63    send_4(CSTEE, smartphone, {{reader,SSTEE,smartphone,
64            CSTEE,mult(xprime,P), mult(y,P)}pk(SSTEE)}
65            k(smartphone,CSTEE));
66    claim_i3(CSTEE, SKR, mult(y,mult(xprime,P)));
67
68
69  }
70  }
```

Le résultat de la vérification du protocole par Scyther (vérification des *claim* des lignes 17 et 18 pour le SSTEE et ligne 66 pour le CSTEE) est représenté par la Figure 5.15. Tout comme pour le scénario précédent, nous utilisons une clé symétrique partagée entre le lecteur et le SSTEE pour simuler un canal sécurisé entre les deux entités. De plus, le smartphone doit être authentifié au CSTEE et le lien de communication doit être sécurisé. Nous avons opté aussi pour une clé symétrique servant à chiffrer leurs échanges. Cette clé peut être embarquée sur le Secure Element (SE) du téléphone et permettre d'établir un canal de communication sécurisé avec le CSTEE. L'authentification du smartphone vers le CSTEE est en dehors du champs de cette thèse.

Nous observons (voir ligne 17 et 66 du Code 5.10) que la clé symétrique négociée entre le CSTEE et le SSTEE reste secrète durant tout le déroulement du protocole. En effet, un attaquant ne peut avoir accès à ce secret négocié et ne peut donc pas, déchiffrer les données d'authentification envoyées chiffrées avec cette clé. Nous pouvons aussi affirmer que SSTEE tout comme le smartphone ont complété l'exécution du protocole d'authentification.

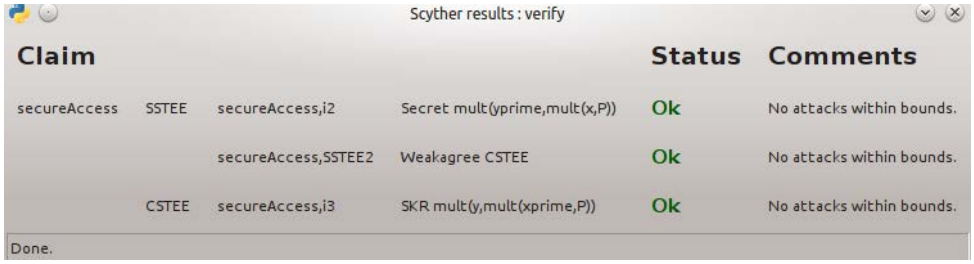

FIGURE 5.15 – Validation du protocole proposé pour le scénario nominal

Conclusion

Dans ce chapitre, nous avons présenté l'environnement OP-TEE que nous avons utilisé dans l'évaluation de notre protocole d'authentification. Nous avons ensuite effectué une double évaluation du protocole d'authentification proposé : une évaluation de performances et une évaluation de sécurité. Pour l'évaluation de performances, nous avons évalué le temps d'exécution des opérations cryptographiques qui sont les plus coûteuses en terme de performances. Nous avons montré que le temps d'exécution de bout en bout (en omettant les temps de transmission) du protocole était de 1.2 secondes dans le pire des cas. Nous avons ensuite présenté de l'outil de vérification d'algorithmes cryptographiques Scyther. Nous avons alors évalué le protocole proposé sur les trois architectures considérées. Nous avons montré que la clé de session négociée restait toujours secrète et donc qu'elle peut être utilisée dans la phase suivante d'envoi des données d'authentification de manière sûre. Nous disposons alors d'un algorithme d'authentification performant et sûr en même temps.

6 Conclusion et Perspectives

6.1 Conclusion

Nous avons dans ce travail de thèse, proposé une nouvelle solution de contrôle d'accès basée sur la dématérialisation des cartes sans contact sur un smartphone dans le cadre du projet neOCampus. Ces cartes sont largement utilisées pour le contrôle d'accès et présentent des vulnérabilités connues. Pour la première génération, les MIFARE 1K, permettent à un attaquant de cloner la carte et d'utiliser le clone de manière frauduleuse à la place de la personne légitime sans que l'attaque puisse être repérée. Pour les cartes MIFARE DESFire, un mécanisme d'authentification basé sur l'utilisation d'algorithmes de chiffrements symétriques standards tels que DES, 3DES ou AES est venu améliorer la sécurité du système de contrôle d'accès. Cependant, Oswald et al. dans [1] ont montré qu'une attaque side channel permet de récupérer l'ensemble de la clé 3DES négociée lors de l'authentification. De plus, la multiplication des cartes sans contact, généralement une par service, les rend encombrantes à transporter pour l'utilisateur.

Afin d'augmenter la sécurité des systèmes de contrôle d'accès et de mettre fin à la prolifération des cartes sans contact, nous avons proposé de dématérialiser ces cartes sur un smartphone muni de la technologie NFC. Cette solution permet notamment de déployer des algorithmes d'authentification plus fiables pour le système de contrôle d'accès et, grâce à l'espace de stockage important du smartphone, il est possible d'accueillir simultanément plusieurs services. Nous avons identifié un certain nombre de menaces sur le système d'exploitation du smartphone rendant le stockage et le traitement des données sensibles non fiables.

Afin de remédier à la problématique du stockage et du traitement sécurisé sur les smartphones, nous avons identifié deux principales familles de solutions : les solutions logicielles et les solutions matérielles. La solution logicielle consiste à virtualiser un Secure OS chargé de stocker et de traiter les données sensibles. Parallèlement, nous avons comparé trois solutions matérielles : les Secure Elements (SE), les Trusted Platform Modules (TPM) et les Trusted Execution Environment (TEE). Pour le stockage et l'exécution de données de manière sécurisée, les TEE représentent le meilleur compromis sécurité/performances. Cependant, tous les smartphones ne sont pas munis de TEE et faute d'accords avec les fabricants, il n'est pas non plus toujours possible de déployer d'applications sur un TEE existant.

Afin de fournir une solution de contrôle d'accès fiable basée sur la dématérialisation, et considérant les contraintes sus-citées, nous proposons une architecture Cloud accueillant un TEE déporté que seul le smartphone peut contacter à l'aide d'une connexion Wi-Fi ou 4G. Le lecteur NFC possède aussi son serveur sécurisé sur une autre zone du Cloud lui permettant d'exécuter le protocole d'authentification avec le Cloud du smartphone et de négocier une clé de session utilisée pour l'envoi des données d'authentification. Sur cette architecture proposée entièrement en ligne, nous avons déployé un algorithme d'authentification basé sur l'identité reposant sur le paradigme challenge/réponse pour la négociation du secret partagé. Ce protocole, inspiré d'IBAKE, a été amélioré par la modification de son paquet standard afin de répondre aux attaques les plus courantes sur les algorithmes d'authentification.

Trois autres scenarii complémentaires ont été proposés. L'utilisation des TEE tend à se démocratiser et de nombreux systèmes embarqués peu onéreux ont vu le jour avec le support de la technologie ARM TrustZone. Afin d'accompagner cette évolution, deux scenarii basés sur l'utilisation d'un TEE sur le smartphone et sur un Raspberry Pi 3 ont été proposés. De plus, un scénario hors ligne permettant l'authentification avec des jetons de connexion est discuté et a été prototypé dans le cadre du projet neOCampus. Nous avons montré dans la preuve de concept OCAC, qu'en cas d'absence de connectivité, le système permet à l'utilisateur de s'authentifier auprès du lecteur pour accéder au bâtiment sécurisé.

Afin d'étudier la viabilité de la solution et sa pertinence dans le contexte du contrôle d'accès, nous avons procédé à une évaluation de la solution en terme de performances et de sécurité. Pour l'évaluation de performances, nous avons choisi d'évaluer les opérations cryptographiques de l'algorithme de contrôle d'accès en négligeant les temps de communications vu la faible taille des paquets transmis. Notre évaluation des opérations cryptographiques nous fournit une estimation du temps global d'authentification qui est dans le pire cas de 1.2 secondes. Ces performances sont acceptables pour une utilisation en temps-réel d'un utilisateur avec son smartphone NFC face à un lecteur de contrôle d'accès.

La suite de l'évaluation a porté sur la sécurité du protocole déployé. Nous avons choisi d'utiliser l'outil Scyther pour effectuer cette évaluation. Cette dernière a porté sur les trois scenarii présentés afin d'avoir des évaluations du protocole d'authentification dans les contextes Cloud, hors ligne et hybride. Ces évaluations ont montré que la clé de session générée n'est pas compromise et restait bien secrète. Cela signifie que les données d'authentifications chiffrées avec cette clé et transmises au serveur de contrôle d'accès restent confidentielles et qu'aucune compromission du protocole d'authentification n'est possible.

6.2 Perspectives

Comme perspectives à ce travail, nous projetons d'améliorer la preuve de concept OCAC avec l'introduction de l'authentification à l'aide du protocole proposé et de l'utilisation d'un SE via le seekforAndroid afin de stocker les jetons de manière sécurisée. Cette amélioration

va permettre d'avoir un niveau de sécurité élevée pour une solution hors ligne utilisée en cas d'absence de connectivité sur le smartphone. De plus l'utilisation d'un Raspberry Pi 3 supportant la technologie TrustZone d'ARM nous permettra de sécuriser le processus de génération des jetons.

Il est aussi nécessaire de tester le passage à l'échelle de la solution au niveau du campus de l'Université Toulouse III. En effet, c'est plus de 32000 étudiants et 4500 personnels qui utilisent des ressources susceptibles d'être protégées. Ce grand nombre d'utilisateurs peut mettre à mal les performances du système de contrôle d'accès. Une étude devra être menée afin d'identifier la meilleure configuration possible en terme de nombre de serveurs et de leur localisation dans le campus. Il est donc nécessaire de recueillir des données sur le nombre d'étudiants qui s'authentifient simultanément, les moments de fortes affluences etc.

Dans le cas de l'utilisation des TEE, il serait intéressant d'utiliser des conteneurs légers et sécurisés permettant d'évaluer par simulation la robustesse d'un tel système capable de gérer quelques dizaines de milliers d'utilisateurs. En effet, cette solution permettrait de modéliser, par exemple, un conteneur isolé par personne authentifiée et un conteneur par lecteur d'accès afin de permettre une parfaite isolation et de s'assurer de la sécurité du système de contrôle d'accès. De plus, du fait de la légèreté de ces conteneurs, les performances d'un tel système pourrait s'avérer très intéressantes.

Afin d'avoir des performances élevées au sein des Clouds déployés tout en gardant un niveau de sécurité élevé, SGX fait office de solution prometteuse. En effet, cette technologie est présente sur la plupart des nouveaux processeurs d'Intel et permet de garantir le stockage et le traitement sécurisé des données sensibles. SGX offre une protection de l'espace mémoire alloué et permet d'isoler l'exécution au sein de conteneurs appelés enclave. Les performances affichées par cette technologie en font une candidate légitime pour le remplacement des TEE déportés au seins des deux Clouds de l'architecture proposée.

Le protocole d'authentification peut également être amélioré par l'utilisation de données biométriques. En effet, beaucoup de téléphones permettent le recueil de différentes données biométriques telles que les empreintes digitales et plus récemment de la reconnaissance faciale (Iphone X). Cet ajout permettrait, dans le cas d'un TEE local de déverrouiller l'application sécurisée. Dans le cas d'un lecteur intelligent type Raspberry Pi 3, recueillir les données biométriques des utilisateurs sur l'écran du lecteur permettrait de certifier la présence de la personne devant la porte d'accès et d'éviter les attaques par relai.

Le cas d'usage pour ce travail est le contrôle d'accès aux bâtiments de l'université Toulouse III. Cependant, il est possible de le transposer pour d'autres cas d'usages : accès aux transports publics et paiement sans contact. Il serait intéressant de disposer d'une suite protocolaire où le contexte d'utilisation permettrait de choisir la suite la plus adéquate pour le contexte. Cette suite dépendrait du niveau de sécurité et des performances souhaitées.

Liste des communications

Article de journaux internationaux avec comité de lecture

1. Bouazzouni, M. A., Conchon, E., et Peyrard, F. (2016). Trusted mobile computing: An overview of existing solutions. Future Generation Computer Systems, https://doi.org/10.1016/j.future.2016.05.033.

Conférences internationales avec comité de lecture

2. Bouazzouni, M. A., Conchon, E., Peyrard, F., et Bonnefoi, P. F. (2016, July). Trusted Access Control System for Smart Campus. In Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences (pp. 1006-1012). IEEE.
3. Bouazzouni M., Conchon E., Peyrard F. and Bonnefoi P. (2017). A Card-less TEE-based Solution for Trusted Access Control.In Proceedings of the 14th International Joint Conference on e-Business and Telecommunications - Volume 6: SECRYPT, (ICETE 2017) ISBN 978-989-758-259-2, pages 456-461. DOI: 10.5220/0006431704560461

Communications

4. Juillet 2015 : Présentation d'un Poster lors de la journée neOCampus.
5. Juillet 2016 : Présentation d'un Poster lors de la journée neOCampus.
6. Septembre 2016 : Présentation d'un Poster lors de l'école d'été Cyber Physical Systems (CPS 2016) : Dématérialisation sécurisée de cartes sans contacts.
7. Juillet 2017 : Présentation d'un Poster lors de la journée neOCampus.
8. Juillet 2017 : Présentation du prototype OCAC lors de la journée neOCampus.

Bibliographie

[1] D. Oswald and C. Paar, "Breaking mifare desfire mf3icd40: Power analysis and templates in the real world," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 207–222.

[2] N. Courtois, K. Nohl, and S. O'Neil, "Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards." *IACR Cryptology ePrint Archive*, vol. 2008, p. 166, 2008.

[3] F. D. Garcia, G. de Koning Gans, and R. Verdult, "Tutorial: Proxmark, the swiss army knife for rfid security research," *Technical Report, Radboud University Nijmegen*, 2012. [Online]. Available: http://proxmark.nl/files/Documents/HOWTOs/Tutorial_Proxmark_the_Swiss_Army_Knife_for_RFID_Security_Research-RFIDSEC_2012.pdf

[4] neOCampus, "Projet neOCampus : campus intelligent et durable." https://www.irit.fr/neocampus/fr/, aug 2017.

[5] "Digital wallet moneo," http://www.moneo.com/acm_index.php, sep 2017.

[6] CROUS, "Izly : Le paiement sur le campus," http://www.izly.fr/, aug 2017.

[7] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 95–109.

[8] McAfee Mobile Security Team, "Mobile Threat Report : What's on the Horizon for 2016," McAfee, Tech. Rep., 2016. [Online]. Available: https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf

[9] C. Cremers, "The Scyther Tool : Presentation of Scyther," https://www.cs.ox.ac.uk/people/cas.cremers/scyther/, Tech. Rep., sep 2017.

[10] P. SemiConductors, "MIFARE Classic 1k, MF1ICS50 specifications," 1998. [Online]. Available: http://www.smartstripe.com/wp-content/uploads/2012/10/MifareClassic.pdf

[11] NXP SemiConductors, "MIFARE DESFire EV1 contactless multi-application IC," NXP, Tech. Rep., 2016. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MF3ICDX21_41_81_SDS.pdf

[12] NXP Team, "MIFARE Ultralight C : Presentation and characteristics," aug 2017. [Online]. Available: https://www.mifare.net/en/products/chip-card-ics/mifare-ultralight/mifare-ultralight-c/

[13] ISO/IEC, "NFC Technology Full specification," march 2015. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:53424:en

[14] V. Sharma, P. Gusain, and P. Kumar, "Near Field Communication," *Department of Computer Science & Engineering Tula's Institute, The Engineering and Management College, Dehradun, Uttarakhand*, vol. 248001, 2013.

[15] K. Finkenzeller, *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John Wiley & Sons, 2010.

[16] CNRFID, "Comment fonctionne le NFC ?" aug 2017. [Online]. Available: http://www.centrenational-rfid.com/comment-fonctionne-le-nfc-article-133-fr-ruid-17.html

[17] NFCForum, "NFC Data Exchange Protocol : Technical specifications," NFC-Forum, Tech. Rep., 2006. [Online]. Available: http://archive.eet-china.com/www.eet-china.com/ARTICLES/2006AUG/PDF/NFCForum-TS-NDEF.pdf?SOURCES=DOWNLOAD

[18] C. Mulliner, "Vulnerability analysis and attacks on nfc-enabled mobile phones," in *Availability, Reliability and Security, 2009. ARES'09. International Conference on*. IEEE, 2009, pp. 695–700.

[19] G. M. Miraz, I. L. Ruiz, and M. Á. Gómez-Nieto, "University of things: Applications of near field communication technology in university environments," *The Journal of E-working*, vol. 3, no. 1, pp. 52–64, 2009.

[20] International Organization for Standardization, "ISO/IEC 18092:2013 Preview Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)," sep 2017. [Online]. Available: https://www.iso.org/standard/56692.html

[21] M. Roland, "Software card emulation in nfc-enabled mobile phones: great advantage or security nightmare," in *Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use*, 2012, pp. 1–6.

[22] R. Want, "Near Field Communication," *IEEE Pervasive Computing*, vol. 10, no. 3, pp. 4–7, 2011.

[23] ECMA Standardization, "Near Field Communication Interface and Protocol -2 (NFCIP-2)," ECMA, Tech. Rep., jun 2013. [Online]. Available: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-352.pdf

[24] Organisation internationale de normalisation (ISO), "ISO/IEC 14443-4:2016 Preview Cartes d'identification – Cartes à circuit intégré sans contact – Cartes de proximité – Partie 4: Protocole de transmission," 2016. [Online]. Available: https://www.iso.org/fr/standard/70172.html

[25] ——, "ISO/IEC 7810:2003 Preview Identification cards – Physical characteristics," 2003. [Online]. Available: https://www.iso.org/standard/31432.html

[26] ISO (International Organization for Standardization), "ISO/IEC 14443-3: Cartes d'identification – Cartes à circuit(s) intégré(s) sans contact – Cartes de proximité – Partie 3: Initialisation et anticollision," Tech. Rep., 2001.

[27] ECMA International, "Near Field Communication - Interface and Protocol (NFCIP-1)," ECMA International, Tech. Rep., 2013. [Online]. Available: https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-340.pdf

[28] Organisation internationale de normalisation (ISO), "ISO/IEC 15693-3:2009 Preview Cartes d'identification – Cartes à circuit(s) intégré(s) sans contact – Cartes de voisinage – Partie 3: Anticollision et protocole de transmission," 2009. [Online]. Available: https://www.iso.org/fr/standard/43467.html

[29] NFCForum, "Site officiel du NFCForum," aug 2017. [Online]. Available: https://nfc-forum.org/

[30] Sony, ""Contactless" convenience with Sony FeliCa," sep 2017. [Online]. Available: https://www.sony.net/Products/felica/about/index.html

[31] NXP SemiConductors, "MF0ICU2 MIFARE Ultralight C," NXP, Tech. Rep., 2009. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MF0ICU2_SDS.pdf

[32] M. Merhi, J. C. Hernandez-Castro, and P. Peris-Lopez, "Studying the pseudo random number generator of a low-cost rfid tag," in *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 381–385.

[33] K. Nohl, "Cryptanalysis of crypto-1," *Computer Science Department University of Virginia, White Paper*, 2008. [Online]. Available: https://www.cs.virginia.edu/~kn5f/Mifare.Cryptanalysis.htm

[34] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia, "A practical attack on the mifare classic," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2008, pp. 267–282.

[35] C. Meijer and R. Verdult, "Ciphertext-only cryptanalysis on hardened mifare classic cards," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 18–30.

[36] M. A. Bouazzouni, E. Conchon, and F. Peyrard, "Trusted mobile computing: An overview of existing solutions," *Future Generation Computer Systems*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16301510

[37] K. Barr, P. Bungale, S. Deasy, V. Gyuris, P. Hung, C. Newell, H. Tuch, and B. Zoppis, "The VMware mobile virtualization platform: is that a hypervisor in your pocket?" *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 124–135, 2010.

[38] T. Cooijmans, J. de Ruiter, and E. Poll, "Analysis of Secure Key Storage Solutions on Android," in *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2014, pp. 11–20.

[39] "ARM Virtulization Extensions." [Online]. Available: http://www.arm.com/products/processors/technologies/virtualization-extensions.php

[40] C. Dall and J. Nieh, "KVM/ARM: The design and implementation of the Linux ARM Hypervisor," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 333–348.

[41] "The Common Criteria (CC)," sep 2017. [Online]. Available: http://www.commoncriteriaportal.org/

[42] S. Bouzefrane and P. Paradinas, *Les cartes à puce*. Hermès science, 2013. [Online]. Available: https://www.lavoisier.fr/livre/informatique/les-cartes-a-puce/bouzefrane/descriptif-9782746239135

[43] T. L. Vinh and S. Bouzefrane, "Trusted Platforms to secure Mobile Cloud Computing," in *The 16th IEEE International Conference on High Performance Computing and Communications* , August 2014, pp. 1096–1103.

[44] Trusted Labs, "EAL7 Certification for a Gemalto smartcard embedded embedded software," march 2015. [Online]. Available: http://trusted-labs.com/trusted-labs-achieves-the-first-\security-eal7-certificate-for-gemalto-smart-\card-embedded-software/

[45] "Mastercard : Secure Elements, kinds and certification." [Online]. Available: https://mobile.mastercard.com/Partner/MobilePayPass/SefcureElements

[46] "Trusted Computing Group official website." [Online]. Available: http://www.trustedcomputinggroup.org/

[47] J. Wayman, A. Jain, D. Maltoni, and D. Maio, "An introduction to biometric authentication systems," *Biometric Systems*, pp. 1–20, 2005.

[48] A.-D. Vu, J.-I. Han, H.-A. Nguyen, Y.-M. Kim, and E.-J. Im, "A homogeneous parallel brute force cracking algorithm on the GPU," in *ICT Convergence (ICTC), 2011 International Conference on*. IEEE, 2011, pp. 561–564.

[49] A. K. J. Salil Prabhakar, Sharath Pankanti, "Biometric Recognition: Security and Privacy Concerns," *IEEE Security and Privacy*, vol. 1, pp. 33–42, 2003.

[50] "RFC 3580 : IEEE 802.1X Remote Authentication Dial In User Service (RADIUS)." [Online]. Available: https://tools.ietf.org/html/rfc3580

[51] Microsoft, "How to protect your network from pass the hash attack." [Online]. Available: https://www.microsoft.com/security/sir/strategy/default.aspx#\protect\kern-.1667em\relaxpassword_hashes

[52] T. Nyman, J.-E. Ekberg, and N. Asokan, "Citizen Electronic Identities using TPM 2.0," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*. ACM, 2014, pp. 37–48.

[53] S. Bajikar, "Trusted Platform Module (TPM) based security on notebook pcs-white paper," Tech. Rep., 2002. [Online]. Available: http://www.ogobin.org/TCPA/Trusted_Platform_Module_White_Paper.pdf

[54] GlobalPlatform, "TEE System Architecture," GlobalPlatform, Technical Report, 2017. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[55] ARM, "ARM Security Technology. Building a Secure System using TrustZone Technology," Tech. Rep., 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[56] "M-Shield mobile security technology," Tech. Rep., 2008. [Online]. Available: http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf

[57] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 2012, p. 9.

[58] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for Android," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 72–81.

[59] "Sierra TEE virtualization system for TrustZone." [Online]. Available: http://www.sierraware.com/open-source-ARM-TrustZone.html

[60] "Genode operating system framework," Genode Team. [Online]. Available: http://genode.org/documentation/articles/trustzone

[61] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Trusted Language Runtime (TLR): enabling trusted applications on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*. ACM, 2011, pp. 21–26.

[62] "OP-TEE official wiki page." [Online]. Available: https://wiki.linaro.org/WorkingGroups/Security/OP-TEE

[63] "LINARO security working group official website." [Online]. Available: https://wiki.linaro.org/WorkingGroups/Security

[64] "T6 : The TrustedKernel secure OS for TrustZone processors." [Online]. Available: http://trustkernel.org/wp-content/uploads/2015/03/T6_TEE_datasheet.pdf

[65] V. Costan and S. Devadas, "Intel SGX explained," vol. 2016, 2016, p. 86. [Online]. Available: http://eprint.iacr.org/2016/086

[66] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*. New York, NY, USA: ACM, 2013.

[67] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han, "OpenSGX: An Open Platform for SGX Research," in *Proceedings of the Network and Distributed System Security Symposium*, 2016.

[68] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," pp. 10:1–10:9, 2016. [Online]. Available: http://doi.acm.org/10.1145/2948618.2954331

[69] J. E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security Privacy*, vol. 12, no. 4, pp. 29–37, July 2014.

[70] B. Castle, "Bouncy Castle crypto APIs," 2007. [Online]. Available: http://www.bouncycastle.org/

[71] S. Knox, "White Paper: An Overview of Samsung KNOX," 2013. [Online]. Available: https://www.samsung.com/global/business/businessimages/resource/white-paper/2013/06/Samsung_KNOX_whitepaper_June-0.pdf

[72] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 173–187.

[73] D. Liu, J. Lee, J. Jang, S. Nepal, and J. Zic, "A cloud architecture of virtual Trusted Platform Modules," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE, 2010, pp. 804–811.

[74] P. Urien, "Cloud of Secure Elements(CoSE)," Internet Draft, Internet Engineering Task Force, 2014. [Online]. Available: http://tools.ietf.org/html/draft-urien-cfrg-cose-00.html

[75] "OpenTEE official GitHub page." [Online]. Available: https://github.com/Open-TEE

[76] A. Vahidi and P. Ekdahl, "VETE : Virtualizing the Trusted Execution Environment," Tech. Rep., 2013. [Online]. Available: http://soda.swedish-ict.se/5456/2/20130305b_VETE_final_report.pdf

[77] Trusty TEE, Google, "TrustyTEE, le TEE de Google," https://source.android.com/security/trusty/, 2017, online; accessed 29 January 2017.

[78] C. Shepherd, G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis, R. N. Akram, D. Sauveron, and E. Conchon, "Secure and trusted execution: Past, present, and future-a critical review in the context of the internet of things and cyber-physical systems," in *SPA, 2016 IEEE*. IEEE, 2016, pp. 168–177.

[79] Trustonic , "Solution TEE Trustonic," https://www.trustonic.com/solutions/trustonic-application-protection-tap/, 2017, online; accessed 29 January 2017.

[80] T. Dierks and C. Allen, "RFC 2246 The TLS protocol," *IETF, January*, 1999. [Online]. Available: https://www.ietf.org/rfc/rfc2246.txt

[81] C. Mulliner, R. Borgaonkar, P. Stewin, and J.-P. Seifert, "SMS-based one-time passwords: attacks and defense," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 150–159.

[82] S. Bouzefrane, A. F. B. Mostefa, F. Houacine, and H. Cagnon, "Cloudlets authentication in NFC-based mobile computing," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*. IEEE, 2014, pp. 267–272.

[83] V. Cakulev and G. Sundaram, "IBAKE: Identity-based authenticated key exchange," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6539.html

[84] A. Shamir *et al.*, "Identity-based cryptosystems and signature schemes." in *Crypto*, vol. 84. Springer, 1984, pp. 47–53.

[85] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology—CRYPTO 2001*. Springer, 2001, pp. 213–229.

[86] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[87] J. Callas, "Identity-based encryption with conventional public-key infrastructure," in *4th Annual PKI R&D Workshop*, no. 7224, 2005, pp. 102–115. [Online]. Available: https://pdfs.semanticscholar.org/b1af/f7e07f6a4733ee3bc6231e4071c0aff46c85.pdf#page=112

[88] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1985, pp. 417–426.

[89] S. Drimer, S. J. Murdoch *et al.*, "Keep your enemies close: Distance bounding against smartcard relay attacks." in *USENIX security symposium*, vol. 312, 2007.

[90] NXP Team, "PNEV512R: EXPLORE-NFC - Exclusive from element14 : NFC Shield for Raspberry Pi," aug 2017. [Online]. Available: http://www.nxp.com/products/identification-and-security/nfc-and-reader-ics/nfc-frontend-solutions/explore-nfc-exclusive-from-element14:PNEV512R

[91] "Secure Element Evaluation Kit for the Android platform." [Online]. Available: http://seek-for-android.github.io/

[92] ARM. Fixed Virtual Platforms (ARM). [Online]. Available: http://www.arm.com/products/tools/models/fast-models/foundation-model.php

[93] "Automated Validation of Internet Security Protocols and Applications." [Online]. Available: http://www.avispa-project.org/

[94] B. Blanchet, "CryptoVerif: Cryptographic protocol verifier in the computational model." [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/

[95] C. J. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *CAV*, vol. 8. Springer, 2008, pp. 414–418.

Résumé

Dans cette thèse, nous proposons de remplacer les cartes sans contact vulnérables utilisées dans le contrôle d'accès par une application sur un smartphone. Cette application fournit les mêmes fonctionnalités que les cartes tout en augmentant le niveau de sécurité global du système de contrôle d'accès. En particulier nous proposons d'utiliser un environnement d'exécution et de stockage sécurisé pour le stockage et le traitement de données d'authentification. Quatre scénarii ont été proposés : un scénario avec un TEE embarqué, un scénario avec un TEE réparti, un scénario avec un TEE déporté dans le Cloud et un scénario hors ligne basé sur l'authentification à l'aide de jetons de connexion. Nous proposons, évaluons et vérifions également la sécurité d'un protocole d'authentification basé sur IBAKE compatible avec les trois scénarios en ligne précédents. Nous proposons enfin un prototype (OCAC) d'authentification à base de jetons de connexion dans le cadre du projet neOCampus.

Mots clés : TEE, dématérialisation, contrôle d'accès, authentication, NFC, cloud sécurisé

Abstract

In this thesis work, we propose to replace vulnerable contact-less cards used in access control systems by a smartphone application providing the same functionalities with higher security level. However, the storage and the processing of authentication data on a smartphone is a challenge implying the use of trusted execution environments to secure the data. Four scenarios were proposed for dematerializaion: A first scenario with an with an embedded TEE, a second scenario with a TEE on the smartphone and on the reader, a third scenario with a TEE deported to the cloud and a fourth scenario with an offline authentication process based on the use of an authentication tokens. We propose, evaluate and formally verify the security of an identity-based authentication protocol based on IBAKE. We finally propose a prototype, called OCAC, which is a proof of concept of the offline authentication based on the use of tokens. This implementation was done as part of the neOCampus project.

Keywords: TEE, dematerializaion, access control, authentication, NFC, secure Cloud