UNIVERSITÉ DE MONTRÉAL

L'AUTHENTIFICATION SÉCURISÉE UTILISANT LE *TRUSTED COMPUTING*

MARWEN JADLA
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
JUILLET 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

L'AUTHENTIFICATION SÉCURISÉE UTILISANT LE *TRUSTED COMPUTING*

présenté par : JADLA Marwen
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doctorat, présidente
M. FERNANDEZ José M., Ph. D., membre et directeur de recherche
M. BARRERA David, Ph. D., membre et codirecteur de recherche
M. DAGENAIS Michel, Ph. D., membre

DÉDICACE

À ma famille, mes amis,
mes collègues et toutes les personnes
qui m'ont aidé

REMERCIEMENTS

Je souhaite remercier toutes les personnes qui m'ont aidé pour réaliser ce mémoire. Tout d'abord mon directeur de recherche José Fernandez, pour son encadrement et ses précieux conseils. Puis mon codirecteur David Barrera, pour ses corrections et les idées qu'il m'a proposées.

Je remercie aussi ma famille : mes parents, Ibrahim et Saida, ma sœur, Khouloud et mon frère, Kais pour le soutien inconditionnel et leurs beaux mots.

Je tiens à remercier mes collègues du laboratoire de recherche : François Labrèche, François Menet, Mikaela Ngamboe, Fanny Lalonde et Corentin Bresteau. Mention spéciale à Militza Jean et à Paul Berthier pour leur aide dans et en dehors du cadre de la maîtrise.

Je remercie également Jim McAlear, pour ses suggestions et d'avoir fourni le matériel. Je remercie enfin mes amis qui m'ont soutenu tout au long de mes études.

RÉSUMÉ

La sécurité des systèmes informatiques demeure un sujet pertinent malgré les efforts sans cesse apportés pour l'améliorer. À cet effet, un nouveau concept prometteur, qui touche simultanément le matériel et le logiciel, a été déployé. Il s'agit du *trusted computing*. L'utilisation de ce concept a pour but de s'assurer de l'intégrité du système que nous utilisons à travers différents mécanismes, comme la vérification d'intégrité ou l'isolation logicielle et matérielle. Ce concept pourrait être utilisé dans différentes applications nécessitant une sécurité accrue. L'un des principaux domaines où nous sommes encore concernés par la sécurité est l'authentification en ligne, particulièrement pour les services critiques comme les services financiers ou le courriel. En effet, la sécurité des données d'authentification en ligne peut être compromise par des logiciels malveillants, tels les chevaux de Troie et les enregistreurs de frappes, ou encore par des attaques matérielles, les enregistreurs de frappes physiques à titre d'exemple. Dans le cadre de ce mémoire, nous utilisons le concept de *trusted computing* afin de proposer une solution d'authentification offrant un compromis entre la sécurité, la facilité d'utilisation et la facilité de déploiement tout en mettant l'accent sur la sécurité. Dans notre schéma nous voulons prévenir les attaques sur les données d'authentifications effectuées à partir de l'ordinateur de l'utilisateur. Un autre but est s'assurer de la compatibilité de notre solution avec la méthode d'authentification actuelle qui consiste en un nom d'utilisateur et un mot de passe.

Notre objectif principal est d'implémenter une application utilisant les mécanismes du *trusted computing* sur un système embarqué. Cette application a pour rôle de fournir une authentification sécurisée aux sites web pour les utilisateurs. Nous avons débuté par étudier les différentes approches possibles en termes de *trusted computing* et d'exécution isolée. Ensuite, nous avons démontré l'utilisation du *trusted computing* sur un système embarqué. Nous avons alors implémenté notre application d'authentification sécurisée en utilisant les possibilités offertes par le *trusted computing*. Cette implémentation a permis d'étudier la viabilité de l'utilisation du *trusted computing* pour régler les problèmes de sécurité.

Finalement, nous avons discuté des différents modèles d'attaques possibles ainsi que des limitations de notre système pour l'évaluer. Notre système nous protège contre plusieurs attaques possibles dans les schémas d'authentification actuels, nous pouvons encore le rendre plus résistant et plus sécuritaire. Nous constatons aussi que le *trusted computing* peut être utilisé dans d'autres contextes de sécurité, car même avec quelques difficultés de développement, il offre beaucoup d'avantages.

ABSTRACT

Computer systems security remains a top priority for researchers and industry as cyber attacks are improving and costing millions of dollars to companies. One of the recent concepts introduced is trusted computing. It is used to guarantee the integrity of the system we are using. The main mechanisms used by trusted computing are integrity verification and software and hardware isolation; they are achieved via hardware and software implementations. The application domain of trusted computing is large and usually used in sensitive contexts. One of the areas where security is critical but with no definitive solution is online authentication, in particular for sensitive services like online banking or email. Currently, online authentication is vulnerable to multiple software and hardware attacks such as trojans, software and hardware keyloggers.

We propose a scheme for online authentication offering a compromise between security, ease of use and backward compatibility, but focusing more on security. We use trusted computing concepts for our solution. We try to prevent attacks aiming user credentials exchanged from the computer of the user. Another objective was to assure backward compatibility with current authentication methods which usually consist of username and password through websites.

Our goal throughout this dissertation is to implement an application using the mechanisms of trusted computing on an embedded system. This application will provide users with a secure authentication mechanism to connect to websites. We began by studying the different possible approaches in terms of trusted computing and isolated execution. Then we ran trusted computing on an embedded system. Next, we implemented the secure authentication application using trusted computing.

Finally, we discussed the possible threat models and limitations of our solution. Our system protects us against a range of attacks under current authentication schemes, although there is room for improvement to make it even more resistant and secure. The concept of trusted computing can be explored further to solve security-related problems. Despite some development difficulties, it can offer many advantages.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AES Advanced Encryption Standard

API Application Programming Interface

DHCP Dynamic Host Configuration Protocol

DoS Denial of Service

DRM Digital Rights Management

eMMC embedded Multi-Media Controller

EXT Extended File System

FAT File Allocation Table

fTPM firmware Trusted Platform Module

GPIO General Purpose Input/Output

HDMI High-Definition Multimedia Interface

HTTPS HyperText Transfer Protocol Secure

HTTP HyperText Transfer Protocol

IP Internet Protocol

MMU Memory Management Unit

NSA National Security Agency

OP-TEE Open Portable Trusted Execution Environment

OTP One Time Password

PIN Personal Identification Number

PUF Physically Unclonable Function

RAM Random Access Memory

REE Rich Execution Environment

ROM Read Only Memory

ROP Return Oriented Programming

RPC Remote Procedure Call

SD Secure Digital

SGX Software Guard Extensions

SMC Secure Monitor Call

SoC System On Chip

SRAM Static Random Access Memory

SSH Secure Shell

SSL Secure Sockets Layer

TCB Trusted Computing Base

TEE Trusted Execution Environment

TLS Transport Layer Security

TPM Trusted Platform Module

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

VGA Video Graphics Array

XSS Cross-site Scripting

CHAPITRE 1 INTRODUCTION

1.1 La cybersécurité : des mécanismes défensifs à parfaire

Malgré des décennies de recherche dans le domaine de la sécurité informatique, les attaques malveillantes contre les systèmes sont toujours légions et leurs conséquences négatives, indéniables. Cette difficulté à faire face à la menace cybernétique peut s'expliquer par le fait que la sécurité des systèmes n'est généralement pas dans les priorités des développeurs ou encore parce que les systèmes informatiques sont de plus en plus hétérogènes. Conséquemment, certaines solutions, valables pour les uns, deviennent inapplicables pour les autres. De plus, comme la performance est généralement privilégiée à la sécurité, les principaux mécanismes de défense sont implémentés de façon purement logicielle. Ainsi, la tâche de protection contre les menaces est laissée au système d'exploitation et aux logiciels antivirus. Aucune protection n'est généralement réalisée au niveau matériel. Cela rend les systèmes particulièrement vulnérables, puisqu'un logiciel malveillant peut modifier la façon dont le système d'exploitation s'exécute ou encore désactiver la protection du logiciel antivirus. Il est donc primordial d'utiliser des mécanismes de sécurité à la fois logiciels et matériels, implémentés dés la conception des systèmes.

1.2 Un concept de défense alternatif : le *trusted computing*

L'une des pistes de recherche explorées par les chercheurs pour faire face aux menaces cybernétiques est le *trusted computing* (informatique digne de confiance). Le *trusted computing* désigne les technologies permettant d'établir la confiance dans un système. Cela signifie que chaque fois que le système est utilisé, cette technologie s'assure que les processus critiques sont sécurisés et s'exécutent dans un environnement isolé. Ce concept repose, généralement, sur des mécanismes matériels et logiciels afin d'effectuer ses vérifications.

1.2.1 Les composantes du *trusted computing*

Afin de faire les vérifications nécessaires et établir la confiance dans un système, le *trusted computing* s'appuie sur des composants dignes de confiance de base : Trusted Computing Base (TCB). À partir de celles-ci, l'intégrité de tout le reste du système peut être vérifiée. L'exemple le plus courant pour l'ensemble des composants dignes de confiance comporte le processeur (ou un mode sécurisé du processeur), du stockage sécurisé et une partie logicielle.

Les domaines d'application du *trusted computing* comportent :
Le démarrage sécurisé (*trusted boot*) : à chaque démarrage du dispositif ou de l'ordinateur, il y a une chaîne de vérification de l'intégrité du logiciel. Cela est généralement fait à l'aide du calcul des *hash* cryptographiques du code.
L'exécution isolée (*isolated execution*) : cela permet d'exécuter des processus sensibles en isolation des autres processus, ou même du système d'exploitation, afin de protéger leurs exécutions et les données qu'ils utilisent. L'isolation peut être logicielle, matérielle ou une combinaison des deux.
Les mises à jour sécurisées(*secure updates*) : la mise à jour des dispositifs doit assurer que la version à installer est authentique et supérieure à celle déjà installée.
Le stockage sécurisé(*secure storage*) : afin de protéger les données des utilisateurs, et même le système d'exploitation et les codes du démarrage des attaques physiques, le stockage sécurisé est indispensable. Les données doivent résister aux différentes attaques cryptographiques même lorsqu'elles sont stockées sur un système de fichiers auquel nous ne faisons pas confiance.
La gestion des droits numériques (*digital rights management*) : la gestion des droits numériques vise à protéger les contenus médias des fournisseurs de services. Par exemple, si un fournisseur loue un film, il veut s'assurer que ce film sera présent seulement la durée du prêt. Cela est garanti par des mécanismes de *trusted computing*.

1.2.2 Méthodes d'implémentation du *trusted computing*

Plusieurs méthodes d'implémentation du *trusted computing* ont déjà été proposées. Parmi les premières solutions fournissant de l'exécution sécurisée, nous pouvons noter l'apparition de la carte intelligente (externe) qui se branche à l'ordinateur afin de fournir des services de sécurité tels que l'authentification sécurisée, ou encore le chiffrement de données (par exemple les cartes FORTEZZA). Cette méthode permet de garantir un grand degré de sécurité logicielle et matérielle.

Par la suite, il y a eu modification de ce concept : au lieu de fournir la sécurité par un module externe à brancher, le module est désormais intégré directement sur la carte mère. Bien que ce schéma diminue légèrement les garanties de sécurité, il augmente considérablement les performances, surtout pour les opérations cryptographiques, vu que ces opérations sont optimisées au niveau matériel du processeur.

Parallèlement, il y a eu apparition d'une technique d'isolation logicielle : la virtualisation (Xen ou Vmware par exemple). Le but principal de cette technologie est d'éviter le gas-

pillage des ressources matérielles et de les utiliser au maximum en tout temps. Mais ce qui a contribué à rendre les machines virtuelles populaires et largement utilisées en sécurité informatique, c'est la facilité de leur installation et de leur configuration, et le fait qu'elles soient isolées. D'ailleurs, elles sont utilisées dans plusieurs contextes en sécurité. Par exemple, dans l'ingénierie inverse des logiciels malveillants, elles fournissent des environnements vulnérables afin d'amorcer des attaques qui permettent d'analyser les techniques utilisées par les pirates. Elles peuvent fournir un environnement d'exécution isolé, même si la machine hôte n'est pas digne de confiance et pourrait être infectée par des logiciels malveillants. Étant une solution souvent gratuite (la plupart des logiciels de virtualisation étant gratuits et à code source ouvert), et ne nécessitant pas de modifications matérielles, cela a contribué à sa popularité et procure une bonne alternative pour faire de l'exécution isolée.

Finalement, il y a eu apparition des extensions de sécurité pour les processeurs, comme Intel Software Guard Extensions (SGX), ou encore ARM Trustzone. Ces extensions permettent de fournir des mécanismes de protection des processus sensibles en les exécutant isolément des autres programmes et du système d'exploitation. Le processeur est incapable de fournir seul une solution sécuritaire complète, mais il y a des implémentations logicielles qui font que cette technologie offre un grand degré de sécurité et d'isolation.

Le *trusted computing* peut donc être implémenté selon différentes méthodes, comme la virtualisation, les modules de sécurité externes ou les processeurs sécurisés, cette dernière étant la plus prometteuse.

1.2.3 Le *trusted computing*, un concept à explorer davantage

Le *trusted computing* commence à être utilisé dans des applications de la vie courante, par exemple, dans le cas du paiement électronique utilisant le téléphone cellulaire. Cependant, l'adoption de cette technologie reste assez limitée et elle est principalement utilisée sur les téléphones cellulaires, même si matériellement elle peut être utilisée sur presque toutes les plates-formes (ordinateurs, systèmes embarqués, etc.). En effet, la majorité des nouveaux processeurs, même ceux qui ne sont pas très chers, offrent des technologies permettant l'exécution isolée. Le potentiel de sécurité de ce concept n'est pas donc pas exploité à son maximum. Parmi les raisons pour lesquelles les recherches utilisant le *trusted computing* ne sont pas assez fréquentes, malgré les avantages offerts, on peut citer l'absence de documentation et le fait que la majorité du code qui inclut la séquence de démarrage et le fonctionnement du processeur sécurisé sont fermés et propriétaires des constructeurs de processeurs. Cela rend difficile aux chercheurs et développeurs d'expérimenter le *trusted computing*, développer des applications sécurisées en l'utilisant, chercher des failles de sécurité afin d'en assurer la sécu-

rité, etc.
Néanmoins ce concept reste très prometteur, il pourrait constituer une solution à plusieurs problèmes de sécurité informatique et de gestion des données privées. Nous allons donc l'explorer au travers de ce mémoire.

1.3 Un cas d'étude : l'authentification

Cependant, la problématique du *trusted computing* est trop large pour que l'on puisse faire une étude quantitative satisfaisante de tous ses domaines d'application possibles. Nous déciderons donc de nous concentrer sur une étude de cas, avec un prototype fonctionnel, pour évaluer l'efficacité et la viabilité du *trusted computing* dans ce cas d'application précis selon des critères de sécurité, de coût, de rétrocompatibilité, et de facilité de développement.

En ce qui concerne le domaine d'application, nous avons constaté que l'authentification constitue un des problèmes de sécurité auxquels aucune solution définitive ou générale n'a encore été trouvée, et qui rassemble une majorité de la communauté des chercheurs en sécurité et l'industrie. Il s'agit du processus qui permet à un client de prouver son identité à un serveur afin d'ouvrir une session personnalisée.

La méthode la plus répandue est l'entrée d'un nom d'utilisateur et d'un mot de passe. Cette méthode présente plusieurs problèmes de sécurité. Plusieurs attaques permettent de voler les identifiants des utilisateurs. Malgré plusieurs propositions et concepts des milieux académiques et professionnels, aucune solution ne règle tous les problèmes ni n'offre un bon compromis entre sécurité, facilité de déploiement et facilité d'utilisation. De plus, il n'y aura probablement pas, à court ou à moyen terme, une solution qui satisfera tous les critères. Les attaques visant les identifiants des utilisateurs sont toujours aussi fréquentes et diversifiées. Parmi ces attaques : les logiciels malveillants qui enregistrent tout ce qui se tape sur le clavier (*keylogger*), les logiciels malveillants de type cheval de Troie qui peuvent, entre autres, intercepter les requêtes et voler les identifiants, les attaques d'hameçonnage où l'attaquant crée un faux site web qui ressemble à un site web légitime que l'utilisateur utilise et lui demande, à travers des courriels frauduleux, par exemple, d'y entrer ses identifiants pour les voler. Sans oublier le *spear phishing*, dans lequel un attaquant possède assez d'informations sur une seule cible, et lui envoie un courriel spécialement conçu pour la tromper et la convaincre d'utiliser ses identifiants.

Nous pouvons donc constater que l'authentification traditionnelle par mot de passe souffre de beaucoup de faiblesses. En conséquence, elle est insuffisante pour faire face à ces différents types d'attaques, et elle doit être remplacée ou améliorée. C'est donc cette problématique de

l'authentification que nous allons choisir comme domaine d'application du *trusted computing*. Un de nos collaborateurs, Jim McAlear, nous a proposé, auparavant, une architecture, où nous utilisons un dispositif séparé afin d'effectuer l'authentification. Dans cette architecture, il y a modification des protocoles réseau afin de détecter les terminaux attachés aux dispositifs externes. Ainsi, l'entrée des données d'authentification se fait sur le dispositif dédié et non pas sur l'ordinateur. Nous allons, la reprendre avec des modifications en la combinant avec le *trusted computing*, dans le but d'avoir une solution d'authentification sécuritaire.

1.4 Questions de recherche et objectifs

Le concept du *trusted computing* offre plusieurs avantages de sécurité s'il est implémenté correctement. Ces avantages sont obtenus à travers les différentes primitives possibles, telles que le démarrage sécurisé et le stockage sécurisé. Dans ce mémoire, nous allons étudier sa viabilité et son applicabilité dans le contexte d'une tâche qui reste une cible des attaques informatiques : l'authentification.
Nous allons réaliser cette étude en posant les questions de recherche suivantes :
1) Quelles sont les différentes approches pour fournir du *trusted computing* sur du matériel dédié ?
2) Parmi ces approches, laquelle offre le meilleur compromis entre sécurité, coût, rétrocompatibilité et facilité de développement dans le domaine de l'authentification ?
3) L'approche choisie du *trusted computing* est-elle avantageuse par rapport aux méthodes d'authentification actuelles selon ces mêmes critères ?
4) Peut-on en pratique concevoir un prototype d'authentification utilisant cette approche du *trusted computing* qui soit rétrocompatible, facile à développer et à coût réduit ?
5) Quel est le risque résiduel de compromission d'une telle solution d'authentification basée sur le *trusted computing* ?

Pour nous guider, nous aurons comme fil conducteur tout au long de ce mémoire, la conception d'une application du *trusted computing* permettant de faire de l'authentification en utilisant un dispositif sécurisé. Pour répondre à cet objectif, nous étudierons et implémenterons le *trusted computing* sur un dispositif embarqué bon marché. Ensuite, nous implémenterons un protocole sécuritaire de transfert de données d'authentification entre un client et un serveur d'authentification en utilisant le concept du *trusted computing*.

1.5 Plan du mémoire

Le reste du mémoire sera organisé comme suit :
Tout d'abord, une revue de la littérature est faite dans le chapitre 2 où nous détaillerons les recherches sur le *trusted computing*, ARM Trustzone, Intel SGX. Dans le chapitre 3, nous présentons les différentes méthodes d'authentification et nous allons les comparer avec notre solution selon différents critères axés sur la sécurité, la convivialité et le déploiement. Nous expliquons comment notre méthode sera différente selon des compromis choisis. Nous détaillons notre implémentation du prototype dans le chapitre 4. Nous expliquons aussi les choix techniques que nous avons faits en termes du choix de l'implémentation du *trusted computing* et le matériel à utiliser. Le chapitre 5 contient une discussion sur les modèles d'attaque possibles ainsi que les limitations de la solution. Finalement, nous résumons nos travaux et détaillons les travaux futurs dans le chapitre 6.

CHAPITRE 2 ÉTUDE EXPLORATOIRE DU TRUSTED COMPUTING

Introduction

Dans ce chapitre, nous allons décrire les travaux antérieurs qui ont porté sur le *trusted computing*. Nous allons présenter ses origines, les principaux concepts et les principales techniques pour l'implémenter.

2.1 *Trusted computing*

2.1.1 Origines

Les premiers travaux proposant du *trusted computing* sont apparus vers la fin des années 1990 et le début des années 2000. L'un des plus remarquables est celui de Arbaugh et al. (1997). Les auteurs y proposent une architecture de démarrage sécurisé. Ils abordent, ainsi, le *trusted boot*, qui est le processus de vérification du système d'exploitation qui se fait au lancement de la machine. Cela permet de s'assurer que la machine que nous sommes en train d'utiliser n'est pas infectée. Dans ce travail, la réalisation du *trusted boot* se fait par une série de vérifications d'intégrité qui commencent à l'allumage de la machine jusqu'au passage du contrôle au système d'exploitation. En effet, la vérification d'intégrité se fait en calculant un haché cryptographique et en le comparant avec celui d'une signature numérique sauvegardée. Cela sera fait pour chaque composant. La première portion du code et les signatures numériques sont stockées dans une mémoire en lecture seule Read Only Memory (ROM). Afin d'éviter les attaques par déni de service : Denial of Service (DoS), lorsqu'il y a une différence entre le *hash* calculé et la signature numérique, il y a un processus de récupération et de correction de l'inconsistance à partir d'une source de confiance avant l'exécution du composant. Dans cette architecture, les composants qui forment le TCB (l'ensemble des composants dignes de confiance) sont : la carte mère, le processeur et une partie de la ROM.
Ce processus de vérification d'intégrité au démarrage commence à être adopté dans l'industrie comme dans le cas des les iPhone Apple (2017) ; c'est celui utilisé dans quelques systèmes modernes afin de garantir l'intégrité.

2.1.2 Notions générales

Dans cette section, nous résumons les principales notions du *trusted computing*, telles que décrites dans le travail de Asokan et al. (2014).

L'intégrité de la plate-forme

Elle peut être vérifiée durant le processus de démarrage ou pendant l'exécution. En ce qui concerne la première technique, il existe deux options :
1 : Le démarrage sécurisé (*secure boot*) : l'approche commune est d'utiliser la signature du code en gardant le code du début de la séquence de démarrage inaltérable. Cette portion du code sera conservée dans le TCB. Le processeur doit inconditionnellement commencer l'exécution de cette partie. Ensuite, des vérifications se font en chaîne d'une portion à l'autre. Si une vérification échoue, le processus de démarrage est avorté.
2 : Le démarrage authentifié (*authenticated boot*) : les composants démarrés sont mesurés, mais pas comparés à d'autres valeurs. À la place, ils sont conservés dans des logs dans une mémoire volatile dont l'intégrité est protégée. Les mesures enregistrées représentent l'état de la plate-forme et peuvent être utilisées pour les attestations à distance par exemple.
Dans le cas où l'intégrité du dispositif est vérifiée pendant l'exécution, une partie logicielle vérifiée en qui nous faisons confiance surveille l'intégrité du code de façon continue.

Stockage sécurisé

C'est un mécanisme pour stocker des données dans le dispositif de manière sécurisée (généralement des informations sensibles, telles que des clés cryptographiques). Il faut que ces données restent non divulguées à un attaquant, même si le dispositif est compromis.

Exécution isolée

L'exécution isolée donne la possibilité d'exécuter du code sensible en dehors du contrôle de l'environnement non sécurisé du système d'exploitation Rich Execution Environment (REE). L'exécution isolée, combinée avec le stockage sécurisé, constitue un environnement d'exécution sécurisée Trusted Execution Environment (TEE). L'exécution dans le TEE doit résister au fait que le REE puisse être compromis. La figure 2.1 montre l'isolation entre le REE et le TEE. Cette architecture reste identique, même si les techniques pour accomplir la séparation entre les deux environnements diffèrent (virtualisation, processeur, etc.). Le TEE permet de faire une exécution sécurisée isolée du REE. Le passage du REE vers le TEE se fait à travers une Application Programming Interface (API).

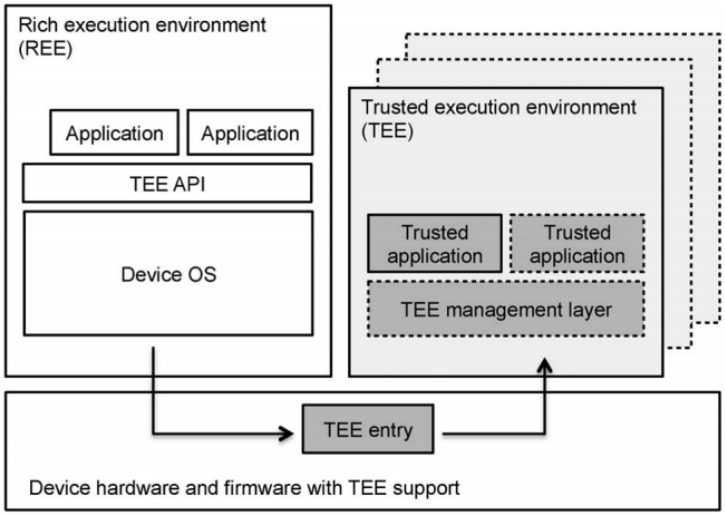


Figure 2.1 Architecture de TEE, Asokan et al. (2014)

Authentification du dispositif

Chaque dispositif doit pouvoir être authentifié par un fournisseur de services. L'authentification de dispositif peut être effectuée en vérifiant une signature numérique de l'identifiant unique du dispositif.

Attestation et provisionnement

L'attestation à distance (*remote attestation*) permet à un fournisseur de service de s'assurer que le dispositif fonctionne avec la bonne version du logiciel. Cela se fait en envoyant les mesures du micrologiciel et du logiciel chargé au démarrage signés avec une clé du dispositif certifiée. Le provisionnement consiste à envoyer un code ou des clés au TEE du dispositif.

2.1.3 Travaux de recherche

Dans la littérature, plusieurs travaux ont pour objectifs de fournir un environnement d'exécution isolé. Dans cet environnement, nous pouvons exécuter des programmes en étant assurés que les mécanismes en place préviendront la modification de l'exécution ou la fuite des données.
Dans le travail de Costan et al. (2016), les auteurs proposent une forte isolation logicielle en utilisant un minimum de changements matériels. Dans ce travail, les auteurs estiment que les systèmes d'exploitation et les hyperviseurs contiennent des vulnérabilités et ne peuvent pas fournir des garanties quant à l'isolation logicielle, surtout pour les attaques du type attaque

par canal auxiliaire (*side channel attack*). Pour rappeler, un hyperviseur est un ensemble de logiciels et/ou matériel qui permet de virtualiser des systèmes d'exploitation, donc de faire fonctionner plusieurs systèmes d'exploitation sur le même matériel. Dans ce travail les auteurs ciblent Xen en particulier. Ces attaques sont basées sur l'exploitation des informations résultant de l'implémentation physique, telles que la consommation de l'énergie, le timing, ou les fuites électromagnétiques.
Pour contourner ces failles, l'architecture proposée repose sur un moniteur de sécurité qui fonctionne avec la priorité la plus élevée. Ce moniteur crée des enclaves pour les processus qui doivent s'exécuter de manière totalement isolée des autres. La gestion des ressources, comme la mémoire vive ou les cœurs d'exécution, est déléguée au système (qui n'est pas nécessairement de confiance). Le moniteur de sécurité vérifie ensuite les décisions prises en utilisant les métadonnées des enclaves et les corrige si elles ne sont pas correctes (mauvaise allocation par exemple). Le TCB se compose du processeur et d'une grande partie logicielle qui se compose du moniteur de sécurité, d'une racine de mesure et d'une enclave de signature.

Suh et al. (2003) ont proposé des architectures de processeur pour faire de l'exécution capable de détecter les altérations (*tamper evident*) et de l'exécution privée et résistante à l'altération (*private tamper resistant*). Les mécanismes importants sur lesquels le design repose sont :

— la vérification de l'intégrité de la mémoire ;
— le chiffrement/déchiffrement de la mémoire ;
— un gestionnaire de contexte sécurisé.

Ces mécanismes sont utilisés pour fournir une exécution valide. Une exécution valide d'un programme peut être garantie en sécurisant trois sources potentielles d'attaques : l'état initial, l'état aux interruptions (incluant le changement de contexte) et la mémoire sur et hors puce. Ils ont offert deux variantes selon le degré de confiance que nous voulons attribuer au système d'exploitation :
TCB constitué du processeur et une partie du système d'exploitation (noyau de sécurité) : dans cette variante, le processeur vérifie l'intégrité du noyau de sécurité au démarrage (en calculant son haché cryptographique). Ensuite, le noyau de sécurité sera responsable du lancement et de l'interruption des programmes, ainsi que de la protection de la mémoire des attaques logicielles (en utilisant la mémoire virtuelle et les niveaux de privilège) et des attaques matérielles (avec un mécanisme de vérification de l'intégrité matérielle de la mémoire). Les opérations de chiffrement et de déchiffrement se font par un engin matériel pour des raisons de performance. L'avantage de cette approche est de réduire les modifications nécessaires pour le processeur, vu que plusieurs fonctionnalités seront effectuées de manière logicielle.
TCB constitué seulement du processeur : dans cette variante, le seul composant auquel

nous faisons confiance est le processeur. Pour garantir l'exécution sécurisée des processus, un gestionnaire de contexte sécurisé sera donc implémenté dans le processeur. Son rôle est d'assurer la protection pour les processus sécurisés. Pour chacun, il assigne un SPID (*secure process* ID). Il maintient une table qui contient plusieurs informations de protection pour les processus sécurisés : le haché cryptographique du programme, les registres, haché pour la vérification de l'intégrité de la mémoire, etc.
Cette approche nécessite plus de modifications à apporter au processeur, mais en contrepartie, cela réduit le code auquel nous allons faire confiance, qui doit être vérifié. De plus, cela peut améliorer les performances.

Le mécanisme de la vérification de l'intégrité de la mémoire est basé sur les arbres de Merkle (*cached hash trees*), qui souvent utilisés pour vérifier l'intégrité de données dynamiques dans un stockage auquel nous ne faisons pas confiance. Dans ce concept, l'espace mémoire est divisé en morceaux. Ces morceaux sont les feuilles de l'arbre, et un parent est le haché cryptographique de la concaténation de ses enfants. Pour vérifier l'intégrité d'un nœud, le processeur vérifie l'intégrité de l'arbre en remontant jusqu'au nœud racine. Les auteurs ont enfin présenté plusieurs cas d'utilisation de leur architecture : l'exécution certifiée comme dans le cas d'un calcul distribué où les nœuds ne sont pas fiables, ou encore la gestion des droits numériques afin d'éviter le piratage des données média.

Certains travaux de recherche ciblent les configurations matérielles bon marché ou dont les ressources sont limitées (pour les systèmes embarqués, par exemple). Dans ces configurations, il n'y a pas de modes sécurisés de processeur. L'approche de Strackx et al. (2010) est de fournir une isolation pour l'exécution des tâches sensibles. L'isolation se fait principalement par l'intermédiaire du logiciel, avec quelques modifications matérielles qui consistent principalement à des instructions du processeur qui doivent être implémentées. Ce concept repose sur une isolation de la mémoire à l'aide d'un module d'auto protection (*self-protecting module*). C'est un espace de mémoire avec une disposition et des paramètres de protection particuliers. En effet, chaque module est divisé en trois parties :

— une partie secrète contenant des données, auquel un code non fiable ne pourra y accéder directement ;
— une partie publique, qui contient des données en mode lecture seulement et le code du module ;
— une partie qui contient les points d'entrée au code public qui est sous forme de pointeurs vers le module d'auto protection.

La demande d'un module et l'interrogation sur le statut de la protection se fait par des instructions directes au processeur.
La recherche de Seshadri et al. (2004) propose une solution pour l'attestation à distance dans le cas où le matériel n'est pas adapté (pas de clé protégée par le matériel). Le principe de l'attestation est de faire des défis-réponses dans un intervalle donné. Les réponses aux challenges permettent de déduire si la mémoire a été modifiée ou pas. Plusieurs mécanismes utilisés permettent de s'assurer de l'authenticité des réponses. Par exemple pour éviter les attaques du type *Replay* où la réponse est calculée d'avance, la semence du générateur des nombres pseudo aléatoires est envoyée avec le challenge. Aussi, toute modification à l'algorithme du calcul du challenge sera détectée. En effet, le code est petit, si bien que l'ajout d'un «if» sera détecté par le nombre de cycles du processeur.

2.1.4 Application dans l'industrie

Un exemple de l'application du *trusted computing* dans la pratique est le Google Chromebook. Dans ce dispositif, il y a utilisation du Trusted Platform Module (TPM). Le TPM est un crypto processeur utilisé pour fournir des fonctionnalités de sécurité directement dans le matériel. Parmi les principales fonctions de cette puce, nous trouvons la génération, le stockage et l'utilisation des clés cryptographiques. Les différentes techniques utilisées lors de sa fabrication le rendent robuste contre les attaques physiques. Dans le Chromebook, le TPM est utilisé afin de :

- protéger les clés cryptographiques : les clés ne quittent pas le TPM, tout calcul qui les implique se fait dedans ;
- empêcher la rétrogradation du logiciel : il existe des scénarios où un attaquant voudrait rétrograder le logiciel installé sur la machine afin d'exploiter des failles connues dans les versions antérieures ; pour contrer cette technique, la version du logiciel sera conservée dans le TPM en mode lecture et écriture, et ne pourra être écrasée que si la nouvelle valeur à écrire est plus grande que la valeur existante ;
- le chiffrement de partition : la clé de chiffrement est générée par le générateur de nombres pseudo aléatoires du TPM ; ensuite, la clé est chiffrée par une clé système dans le TPM.

Le TPM n'est utilisé que localement. Ses services ne sont pas disponibles en dehors du Chrome OS.

2.2 Approches pour le *Trusted computing*

Afin d'augmenter la sécurité des dispositifs (surtout les systèmes embarqués), plusieurs solutions existent dans l'industrie, même avant l'apparition de ARM Trustzone (ARM (2009)). Même si ces solutions sont compatibles avec le *trusted computing* fourni par ARM Trustzone, il est rare de retrouver les deux implémentées simultanément. Nous allons décrire brièvement ces différentes techniques, comme expliqué dans ARM (2009).

2.2.1 Module de sécurité matériel externe

Typiquement, ce module prend la forme d'une carte intelligente (*smartcard*, comme FORTEZZA ou KSV-21 par exemple) qui va contenir les informations sensibles à protéger comme les clés privées, et celles capables de faire des opérations cryptographiques. Parmi les points forts de cette solution, on note que ces cartes sont très résistantes aux observations physiques, c'est-à-dire qu'aucune information ne peut être obtenue avec des attaques par canal auxiliaire. Aussi, pour certaines cartes, le processus de vérification est très rigoureux : celles certifiées *type 1* par l'agence nationale de sécurité américaine (National Security Agency (NSA)) peuvent être utilisées pour sécuriser de l'information classifiée en utilisant la cryptographie. Pour être certifié *type 1*, un système doit passer un processus de vérification minutieux qui teste et analyse formellement les fonctionnalités cryptographiques, les fonctionnalités de sécurité, la résistance à l'altération et même les émissions électromagnétiques, vibrations, etc. Cela les rend très utiles dans un cadre où nous avons besoin d'un grand degré de sécurité.
Cependant, l'un des points faibles d'une telle solution est que les cas d'utilisation sont réduits, car leur capacité de calcul est très limitée. Ainsi, comme les opérations sont limitées, la carte doit déléguer quelques fonctionnalités au système d'exploitation (auquel nous ne faisons pas nécessairement confiance) ce qui peut introduire un vecteur d'attaque. On peut citer par exemple le cas d'une opération d'entrée du code personnel (Personal Identification Number (PIN)).
L'inconvénient le plus évident est économique. Le prix de ces cartes est élevé, des centaines de dollars pour les cartes mentionnées. Ceci n'est pas adéquat pour la plupart des cas d'utilisations simples comme celui que nous souhaitons implémenter.

2.2.2 Module de sécurité matériel interne

Dans ce cas de figure, le module est présent sur le System On Chip (SoC) et il en existe principalement deux types. Le premier type fournit en particulier des fonctionnalités cryptographiques. Plus précisément, il permet de stocker des clés cryptographiques, mais aussi

de faire des calculs cryptographiques. Les puces TPM en sont un bon exemple. Elles sont utilisées afin de stocker des clés cryptographiques et d'effectuer des opérations de chiffrement et de déchiffrement. Les clés cryptographiques ne quittent pas la puce. Les spécifications de ces puces et les opérations fournies sont définies par une spécification d'un consortium : le Trusted Computing Group. Les puces TPM n'exécutent que des instructions cryptographiques sans avoir accès aux exécutions des programmes ni au système d'exploitation (par exemple, elles ne peuvent pas bloquer une exécution). Parmi les constructeurs de ces puces, nous trouvons Intel, Broadcom et STMicroelectronics.
Le deuxième type consiste en un processeur général, en plus du processeur principal (Microsemi par exemple). Ce processeur traite les données critiques. Ce schéma ressemble à la sécurité proposée par Trustzone. Bien qu'il soit plus performant que le module de sécurité externe expliqué dans la section 2.2.1, les processeurs utilisés sont toujours moins performants que les processeurs principaux. Leurs cas d'utilisations sont donc restreints. Les ressources doivent être séparées de manière matérielle. Cela se produit au cours du design du SoC et en conséquence nous ne constatons pas de flexibilité pour des modifications (par exemple les données des dispositifs d'entrée/sortie qui peuvent être assignées dynamiquement au monde normal ou au monde sécurisé dans Trustzone). Un autre problème de sécurité se pose : les mécanismes de débogage présents sur le SoC qui constituent un vecteur d'attaque pour le processus sécurisé. La désactivation entraînera une difficulté dans le développement des applications.

2.2.3 Virtualisation

La virtualisation est un mécanisme logiciel (il y a des conditions matérielles à accomplir cependant) qui permet de faire fonctionner plus qu'un système d'exploitation sur la même plate-forme matérielle. Soit un système d'exploitation fonctionne au sein d'un autre (émulation du matériel comme avec Virtualbox), soit les deux fonctionnent en parallèle (para virtualisation comme Xen). Dans ce deuxième cas, l'hyperviseur est exécuté avec les privilèges les plus élevés. Il est responsable de l'allocation de mémoire, de l'exécution et de la traduction des instructions, et de l'isolation des ressources (une machine virtuelle ne doit pas avoir accès aux ressources allouées à une autre machine ou à l'hyperviseur). La virtualisation peut être utilisée en sécurité afin de fournir un environnement d'exécution isolé. Par exemple dans le cas de réingénierie de logiciels malveillants, ou dans celui de l'exécution de données sensibles dans un environnement isolé. L'un des plus grands avantages de la virtualisation est l'absence de modifications matérielles à effectuer - tant qu'il existe une unité de gestion de mémoire (Memory Management Unit (MMU)) dans le processeur pour le cas de para virtualisation -, et ceci même sur les systèmes embarqués. La virtualisation constitue aussi

un avantage économique puisqu'elle ne nécessite pas un matériel dédié et il existe plusieurs logiciels de virtualisation gratuits (Virtualbox et Xen par exemple).
Même si la virtualisation présente plusieurs avantages, elle comporte aussi des inconvénients que nous devons mentionner, tels que l'existence de plusieurs attaques physiques qui visent les hyperviseurs et qui peuvent récupérer des informations sensibles. La majorité de ces attaques sont de types attaque par canal auxiliaire qui exploitent le fait que généralement le cache du processeur au niveau est partagé par toutes les machines virtuelles (comme dans Yarom and Falkner (2014)). Aussi, vu la taille importante (en termes de lignes de code) pour les hyperviseurs, ils contiennent généralement beaucoup de vulnérabilités logicielles qui permettent à un logiciel malveillant de dépasser l'isolation logicielle et d'attaquer les autres machines virtuelles ou l'hyperviseur (il y a des dizaines de vulnérabilités dans Xen comme nous pouvons le constater dans la base de données de vulnérabilités CVE-details, accessible publiquement). Finalement, faire de la virtualisation sur un dispositif embarqué (avec des ressources limitées) entraînera des conséquences sur les performances.

2.2.4 ARM Trustzone

Définition

ARM est la compagnie leader dans la conception de processeurs pour les dispositifs mobiles. Afin d'augmenter la sécurité des appareils mobiles, ils ont présenté une architecture sécurisée nommée Trustzone ARM (2009). Cette technologie a été proposée par ARM afin de fournir un TEE. Elle consiste essentiellement à des extensions matérielles et logicielles au processeur, lui permettant de constituer un support matériel au *trusted computing*. Dans cette architecture, le processeur est divisé en deux processeurs virtuels, gérés par un contrôle d'accès implémenté physiquement. Les deux processeurs donnent accès à deux environnements distincts, référés par la documentation d'ARM comme deux mondes : le monde sécurisé et le monde normal (*secure world* et *normal world*). Le passage du monde normal au monde sécurisé se fait par l'intermédiaire d'un moniteur. Le changement est initialisé par une instruction du type Secure Monitor Call (SMC). Pour assurer la sécurité du monde sécurisé, toutes les ressources du système doivent être assignées à un monde à la fois (particulièrement les dispositifs d'entrée-sortie). La figure 2.2 montre l'architecture simplifiée d'un *core* implémentant les extensions Trustzone. Généralement, la mémoire est divisée au démarrage, et chaque partition assignée à un monde n'est pas visible de l'autre monde. Un $33^{ème}$ bit (dans une architecture 32 bits) a été ajouté. Ce bit, nommé bit non sécurisé (*non-secure bit*), permet d'indiquer, à chaque demande d'accès à la mémoire, de quel monde elle provient, et d'agir en conséquence. Par exemple, si le bit a la valeur 1 et la demande concerne une région de mémoire pour le monde

sécurisé, l'accès sera refusé. Pour combiner bonnes performances et sécurité, le cache L1 est partagé entre les deux mondes, mais il y a ajout d'un tag comprenant de quel monde l'instruction provient.

ARM propose aussi une architecture pour faire du démarrage sécurisé en utilisant Trustzone. Le *trusted boot* se fait en construisant une chaîne de vérification. La vérification se fait par de la cryptographie à clé symétrique. La première portion du code qui s'exécute (premier *bootloader*) devrait être stockée dans la mémoire en lecture seule (ROM) du SoC qui rendra les attaques physiques très difficiles. Si nous avons des composants à qui nous faisons confiance (le processeur et la ROM), nous pouvons vérifier, à partir de là, l'intégrité de tout le reste du système. Le monde sécurisé est toujours chargé en premier.

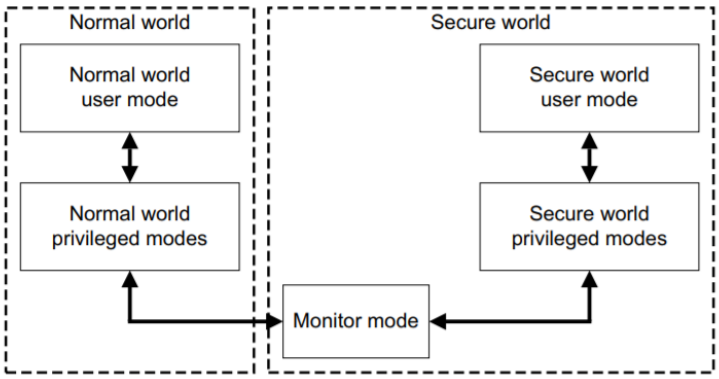


Figure 2.2 Architecture simplifiée de Trustzone, ARM (2009)

Applications pratiques de Trustzone : Les utilisations principales de Trustzone comprennent :

- la protection des données sensibles des utilisateurs : les applications bancaires, par exemple, utilisent le monde sécurisé afin de traiter les mots de passe, et les numéros de carte ; ils ne seront pas accessibles par le monde normal ARM (2009) ;
- le démarrage sécurisé utilisé par les fabricants de téléphones cellulaires. (comme Apple, par exemple sur l'iOS : Apple (2017)) ;
- la protection du monde normal à partir du monde sécurisé ; par exemple, la protection en temps réel du noyau du monde normal, dans le travail de Azab et al. (2014).

Évidemment, nous ne pouvons pas tout exécuter dans le monde sécurisé. Le but du monde sécurisé est de fournir un environnement d'exécution minimaliste afin de diminuer le risque de vulnérabilités et de permettre sa vérification. Il fournit, aussi, des fonctionnalités nécessaires aux applications pour traiter les données critiques telles que la gestion des clés cryptographiques. L'exécution des applications entières dans le monde sécurisé augmente la surface

d'attaque. Un attaquant qui réussit à exploiter une vulnérabilité dans une application tierce pourrait avoir accès aux ressources du monde sécurisé. Cela compromettra la sécurité des autres applications et du monde sécurisé, rendant ainsi l'isolation inutile. Si nous prenons un exemple concret, une application bancaire sur un téléphone cellulaire peut demander à authentifier l'utilisateur à travers ses empreintes digitales. L'application bancaire s'exécute dans le monde normal. Lorsqu'elle veut authentifier l'utilisateur, elle appellera le monde sécurisé, qui scannera l'empreinte digitale de l'utilisateur et la comparera avec celle sauvegardée dans le stockage sécurisé. Même si un attaquant a pu lire les données de l'application bancaire, il n'aura accès ni à la lecture de l'empreinte ni à l'empreinte sauvegardée puisque ces deux opérations se font dans le monde sécurisé en isolation du monde normal. La seule réponse du monde sécurisé à l'application est si l'utilisateur a pu être authentifié ou pas. Si, par exemple, nous exécutons toute l'application bancaire dans le monde sécurisé, toute faille exploitée par un attaquant, lui donnera accès aux données des empreintes digitales de l'utilisateur.

Améliorations de Trustzone

Plusieurs travaux de recherche ont ciblé ARM Trustzone afin de concevoir un système plus sécurisé en utilisant cette technologie, parmi lesquels Raj et al. (2016). Dans ce travail, les auteurs ont remplacé un TPM par une implémentation logicielle en utilisant un processeur ARM implémentant la technologie Trustzone. L'idée générale est d'implémenter le firmware Trusted Platform Module (fTPM) comme un service s'exécutant dans le TEE fourni par le SoC. Le TCB du système se compose du SoC d'ARM (fournissant la technologie Trustzone), le TEE et le service fTPM. Les auteurs ont explicité les points selon lesquels le SoC ARM seul ne peut pas fournir un TPM. Parmi lesquels :

- l'absence de stockage sécurisé ;
- l'absence d'horloge persistante ;
- l'absence d'une source d'entropie.

Pour surmonter ces points manquants, les auteurs ont fait des modifications dans les sémantiques de la logique du TPM, qui consiste à faire des compromis au niveau du design et des exigences au niveau du matériel :

1 : Afin de contourner l'absence de stockage sécurisé, le chiffrement seul ne suffit pas. Les auteurs exigent un contrôleur multimédia (embedded Multi-Media Controller (eMMC)) capable de déjouer les attaques du types *replay* sur la mémoire. La combinaison du eMMC avec le chiffrement permet de fournir un stockage sécurisé.

2 : Des fusibles (*fuses*) accessibles seulement du monde sécurisé : ce sont des fusibles matériels qui permettent d'écrire des données une unique et seule fois. Leurs accès en lecture se faisant à partir du monde sécurisé ; ils seront utilisés pour stocker une clé unique au dispositif, qui

sera employée comme semence pour générer d'autres clés utilisées par le fTPM.
3 : Une source d'entropie accessible par le monde sécurisé seulement générée par un processus physique.
4 : Exécuter une commande pendant une longue période dans le TEE peut compromettre la stabilité du système d'exploitation roulant dans le monde normal. Les auteurs ont décidé de ne pas exécuter des commandes qui prennent beaucoup de temps. La seule commande du TPM qui s'exécute pour une longue période est celle responsable de la génération des clés RSA, et cela pour deux raisons principalement : la recherche de deux nombres premiers assez grands prend du temps, et la recherche exige une semence. La source d'entropie doit donc avoir généré assez d'entropie.
5 : La spécification du TPM requiert une horloge sécurisée. Par exemple, dans le cas d'une attaque du type dictionnaire, le dispositif doit être verrouillé pour une durée déterminée chaque fois que le nombre maximal d'entrées de mots de passe erronés est atteint. L'horloge sécurisée permettra de respecter le délai qu'un attaquant ne pourra modifier. Selon les auteurs, l'absence de l'horloge peut être dépassée en modifiant la logique. En effet, la difficulté d'implémenter l'horloge sécurisée est le fait que le système d'exploitation (auquel on ne fait pas confiance) doit gérer l'horloge. Le fTPM a été implémenté de façon que si la valeur de l'horloge n'est pas stockée, le fTPM n'exécutera pas de commandes.
Les auteurs sont passés après, à l'évaluation des performances par rapport à de vraies puces TPM. Les performances étaient similaires. Certaines commandes étaient même plus rapides à s'exécuter sur le fTPM.
Sur le plan de la sécurité, l'implémentation logicielle du fTPM ouvre la voie à de nouveaux canaux d'attaques, comme les attaques sur la mémoire et celles du type canaux auxiliaires sur le cache. Ces attaques sont impossibles sur une puce TPM puisqu'elle utilise ses propres ressources et n'en partage pas avec le système d'exploitation. Mais les techniques utilisées par Trustzone permettent de minimiser le risque, comme la séparation de la mémoire entre les mondes et l'invalidation du cache en passant d'un monde à l'autre. En contrepartie, la puce TPM est reliée au processeur avec un bus de communication, ouvrant la possibilité à un attaquant ayant un accès physique d'exploiter ce canal et d'intercepter la communication ce qui est impossible dans le cas du fTPM.

Ajout de racine de confiance

Une autre recherche, Zhao et al. (2014), dans laquelle les auteurs bâtissent un prototype de système sécurisé utilisant principalement ARM Trustzone, mais aussi d'autres concepts comme la Physically Unclonable Function (PUF).

Les auteurs ont conçu un système offrant un TEE, aussi fait du *trusted boot*, sans nécessiter de modifications matérielles. En effet, comme le Trustzone n'offre pas de stockage sécurisé, pour résoudre le problème de stockage de clés, les auteurs ont utilisé les PUF. Les PUF, comme décrits dans le travail de Maes and Verbauwhede (2010), sont des objets physiques qui sont soit électroniques (circuit intégré : la forme la plus répandue) ou pas (optique, magnétique, etc.), qui génèrent une réponse lorsqu'ils reçoivent un challenge. Cette réponse dépend du challenge et des caractéristiques physiques uniques du PUF. Parmi ces caractéristiques nous trouvons : robustesse, non-prédictibilité, unicité, reproductibilité avec une petite marge, etc. Les PUF sont utilisés pour reconstituer la clé à chaque démarrage. Au premier les bits du Static Random Access Memory (SRAM) (une petite mémoire située sur le SoC) sont lus. Ensuite, une fonction génératrice en extrait la clé secrète. À chaque démarrage, la clé est reconstituée à l'aide d'une fonction de régénération ce qui permet de rendre difficile à un attaquant de connaître la clé, puisque si l'appareil est éteint, la clé n'existe pas physiquement, et lorsqu'il est alimenté, Trustzone devrait la protéger.
Dans ce système, il y a deux composantes principales :
Building block : c'est un code destiné à s'exécuter juste après le boot ROM, avant même le démarrage du monde sécurisé. Il est chargé dans le SRAM(une petite mémoire dans le SoC donc assez protégée contre les attaques physiques), et son intégrité est vérifiée par le boot ROM. Il est responsable de préparer les primitives pour préparer la racine de confiance, à partir de laquelle tout le système serait vérifié. En particulier, il extrait une semence primaire et une semence aléatoire, et les passe à la racine de confiance. Le *building block* joue aussi un autre rôle, celui de vérifier l'intégrité du code du monde sécurisé.
Racine de confiance : elle consiste principalement à deux primitives qui vont fonctionner dans le monde sécurisé :
1 : le scellage et descellage. Cette primitive permet aux services de sceller et de desceller les données en utilisant le chiffrement qui se fait en deux temps. Premièrement, les données sont chiffrées à l'aide d'une clé déduite du service en question (le *hash* cryptographique du binaire du service). Ensuite, elles sont chiffrées à l'aide de la clé du dispositif. Le fait d'utiliser les deux chiffrements a pour but de s'assurer que seul le service sécurisé concerné peut déchiffrer (premier chiffrement), et que seul le dispositif concerné peut déchiffrer(deuxième chiffrement), ce qui est une forme d'attestation sur l'état du système (seul un monde sécurisé utilisant le bon logiciel peut avoir accès aux clés).
2 : un TPM logiciel. Le TPM physique est remplacé par un service qui le simule en recevant les commandes à travers une interface logicielle. L'implémentation fait en sorte que les logiciels qui ciblaient un TPM physique n'ont pas besoin d'être modifiés afin d'être compatibles avec le TPM logiciel.

Les auteurs estiment que leur prototype est sécurisé contre les différents types d'attaques. En effet, en prenant compte que le SoC est bien protégé contre les attaques physiques, la valeur initiale de la SRAM ne peut pas être lue par un attaquant, vu qu'elle ne quittera pas la puce. Donc, un attaquant ne peut pas obtenir la semence primaire. Pour les attaques logicielles, puisqu'à partir du boot ROM l'intégrité du code est vérifiée, donc seulement un code vérifié peut s'exécuter dans le SRAM. Concernant la clé du dispositif, celle-ci est protégée contre les attaques logicielles vu qu'elle ne quitte pas le monde sécurisé. Par contre, elle n'est pas protégée contre les attaques physiques sur la Random Access Memory (RAM). Les auteurs ont mis en place un mécanisme de changement de clé pour contourner le fait qu'elle soit compromise.

Ajout de virtualisation

Dans un travail récent de Hua et al. (2017), les auteurs proposent de rectifier une faiblesse de ARM Trustzone : l'absence de l'aide à la virtualisation du monde sécurisé. En fait, cette caractéristique n'est pas vraiment nécessaire dans un dispositif mobile, mais l'apparition de serveurs, dont le SoC est basé sur l'architecture ARM, nous interpelle sur l'absence de l'aide à la virtualisation du Trustzone. Actuellement, toutes les machines virtuelles devraient partager un seul monde sécurisé, qui clairement ne constitue pas une solution valide, vu que l'un des objectifs de Trustzone est d'offrir une exécution isolée. Le but des auteurs est aussi d'éviter d'avoir une large portion du code qui s'exécute dans le monde sécurisé afin d'atténuer le nombre de failles potentielles. Par exemple, une solution triviale est d'implémenter un hyperviseur dans le monde sécurisé, mais cette solution nécessite que des milliers de lignes de code s'exécutent dans le monde sécurisé, et ainsi a été rejetée par les auteurs. Ils ont opté pour une autre stratégie : séparer les fonctionnalités et la protection. En effet, ils ont décidé d'implémenter l'hyperviseur dans le monde normal afin de virtualiser les TEE, mais d'utiliser Trustzone afin d'implémenter des fonctionnalités qui vont assurer la sécurité et le bon fonctionnement de l'hyperviseur dans le monde sécurisé. Pour assurer la sécurité de l'exécution, deux modules roulant dans le monde sécurisé sont sollicités :

Secured Memory Mapping (SMM) : ce module permet de contrôler les tables de traduction des adresses mémoire. En particulier, la table de traduction de l'hyperviseur. Ce module s'assure donc qu'il n'y a pas de mémoire du TEE d'une machine virtuelle correspondant à la mémoire d'une autre ou de l'hyperviseur. Aussi, l'hyperviseur ne peut plus allouer de nouvelles pages exécutables après le démarrage.

Secure World Switching (SWS) : chaque changement de contexte entre le monde sécu-

risé et le monde normal d'une machine virtuelle, et aussi entre différentes machines virtuelles, passe par ce module qui s'assure que les états du CPU sont respectés et qu'il n'y a pas de modifications de l'hyperviseur, par exemple. Aussi, ce module vérifie l'intégrité de chaque TEE de machine virtuelle avant d'être chargé dans l'hyperviseur.
En parallèle, il y a les mécanismes souvent utilisés avec Trustzone tels que le *trusted boot* et aussi, il y a la vérification de l'intégrité de l'hyperviseur à chaque démarrage.
Au point de vue sécurité, le système est résistant à la majorité des attaques logicielles, comme l'attaque du type Return Oriented Programming (ROP) (Roemer et al. (2012)) utilisant le code de l'hyperviseur, en s'assurant qu'il n'y a pas d'instructions clés dans la section *text* dans le mode hyperviseur. Les auteurs utilisent des TEE non développés par eux, et qui peuvent contenir des bogues. Même si un attaquant les exploite, il pourrait avoir leurs données ou même les contrôler, mais il ne pourrait pas dépasser l'hyperviseur ou attaquer d'autres machines virtuelles. Cependant, les auteurs disent que leur solution ne prend pas en compte les attaques physiques telles que les attaques par canaux auxiliaires ou les attaques de déni de service (par exemple, l'hyperviseur refuse d'exécuter une machine virtuelle).

Les défauts de Trustzone

Les extensions ARM Trustzone donnent beaucoup de possibilités en termes d'exécution isolée, *trusted boot*, etc. Néanmoins, Trustzone souffre de quelques caractéristiques manquantes afin de pouvoir réaliser un grand degré de sécurité sans nécessiter d'autres modifications matérielles. Ces caractéristiques manquantes sont exposées dans les travaux de recherche, parmi lesquelles nous pouvons citer :

- l'absence de stockage sécurisé : ARM Trustzone fournit un environnement d'exécution isolé, mais ne fournit pas de stockage sécurisé ; ceci implique que même si les données sont chiffrées, le problème de la sauvegarde de la clé reste posé ; la solution pour cette problématique est généralement la programmation de fusible pour un usage unique ;
- l'absence de source d'entropie : une bonne source d'entropie est nécessaire afin de pouvoir générer des clés RSA assez aléatoires ;
- l'absence de compteur persistant : les compteurs persistants sont utilisés pour fournir un compteur qui résiste aux redémarrages, à l'enlèvement de la batterie, etc. Ceci aide à prévenir les attaques visant à installer une vieille version de logiciel, par exemple. Les compteurs sauvegardent la dernière version de logiciel installé, et ensuite, à chaque nouvelle installation, il y a vérification si le logiciel à installer est plus récent que celui déjà installé.

Parmi les problèmes compliquant les travaux de recherche sur Trustzone, il y a celui de l'absence d'implémentation en code source ouvert de cette technologie. En effet, les fabri-

cants de SoC préfèrent garder les codes sources de leurs *firmwares* fermés pour des soucis de concurrence. Le fait de garder les logiciels fermés diminue la sécurité vu que les chercheurs ne peuvent pas trouver des failles facilement. Aussi cela ralentit le développement de noyaux utilisant la technologie Trustzone.
Un autre point important : ARM réalise seulement l'architecture de Trustzone. L'implémentation est cédée aux fabricants de SoC. Malgré la nécessité de bien implémenter le noyau du monde sécurisé, il y a eu plusieurs bogues dans les TEE commercialisés par les vendeurs de SoC. Comme dans la publication de Beaupre (2015), où l'auteur découvre une vulnérabilité dans Trustzone, dans les SoC fabriqués par Qualcomm (en particulier Snapdragon 805). Dans cette faille, un appel système de Trustzone ne filtre pas bien les données en entrée. En conséquence, il peut donner des accès arbitraires aux mémoires du monde sécurisé de Trustzone en lecture et écriture. Une attaque similaire, Huawei (2015), vise une vulnérabilité dans le pilote de Trustzone dans les cellulaires Huawei, qui pourrait permettre de faire crasher le système ou de faire de l'élévation de privilèges et d'exécuter des applications malveillantes dans Trustzone allant jusqu'à donner l'accès au capteur d'empreintes digitales.
Un travail intéressant de Tang et al. (2017), montre que même si nous considérons un système implémentant Trustzone sans faille dans le TEE, un attaquant peut exploiter d'autres vecteurs afin de l'exploiter. Dans ce travail, les chercheurs ont exploité une nouvelle fonctionnalité présente dans les appareils mobiles qui permet de contrôler le voltage et la fréquence du processeur. Ce contrôle est effectué de manière matérielle et logicielle. L'idée générale de l'attaque est de varier le voltage et la fréquence du processeur à partir d'une application malveillante pour le forcer à faire des erreurs de calcul. Tout cela, en ciblant un processus particulièrement utile comme celui de déchiffrement. Le fait que le contrôle de voltage et de fréquence soit indépendant entre les cœurs du même processeur permet d'exécuter l'attaque sur un processus tout en gardant le processus attaquant intact. Les auteurs ont exécuté une application de chiffrement Advanced Encryption Standard (AES) dans le TEE de Trustzone, ensuite ils ont utilisé un processus attaquant afin d'introduire une erreur dans une itération de chiffrement. En conséquence, avoir le bon et le mauvais texte en clair réduira les efforts de l'attaque de force brute pour deviner la clé de 2^{128} à 2^{12} (dans le cas de AES 128). L'attaque peut être considérée comme une attaque physique par canal auxiliaire, mais la nouveauté est qu'elle peut être effectuée sans accès physique, vu que le voltage et la fréquence peuvent être changés avec du logiciel (qui ne se trouve pas dans le monde normal).
Donc, même le fait d'exécuter dans le Trustzone ne donne pas des garanties absolues quant à l'isolation de l'exécution. Ceci est dû au fait que généralement, le TCB contient le noyau sécurisé de Trustzone (le TEE), et ceci, comme vu précédemment peut contenir des bogues et, donc, compromettre la sécurité de tout le système. Et même sans failles dans les logiciels,

les attaquants pourraient faire des attaques physiques sur le matériel directement, bien que ce type d'attaque reste difficile à réaliser et demande du matériel avancé. Les fabricants de SoC utilisent des techniques de protection avancées contre les attaques physiques.

2.2.5 Intel SGX

Intel Software Guard Extensions Anati et al. (2013) est l'ensemble des extensions de sécurité proposé par Intel pour fournir un environnement d'exécution isolé et digne de confiance sans nécessiter beaucoup de modifications matérielles. Intel cible le marché d'ordinateurs et de serveurs, contrairement à ARM qui cible principalement les appareils mobiles. Intel SGX fournit des garanties similaires à celles offertes par ARM Trustzone en ajoutant une nouvelle caractéristique qui est l'attestation à distance (très pratique pour le cas d'exécution dans des serveurs distants), mais son mode de fonctionnement n'est pas similaire.
En effet, l'isolation avec SGX se fait par la création d'enclaves isolées. En fait, le cycle de vie du logiciel dans SGX se déroule principalement en quatre étapes :
1 : Lancement de l'enclave avec le code sensible dedans et un log contenant des informations sur le contenu de l'enclave correspondant à sa mesure.
2 : L'enclave contacte le fournisseur de services afin de recevoir les informations sensibles de sa part. La plate-forme fournit des attestations sur le matériel et l'enclave.
3 : Le fournisseur de service examine l'attestation. S'il conclut que l'enclave est digne de confiance, il transmet les informations.
4 : L'enclave utilise une clé cryptographique matérielle persistante afin de sceller et de desceller les données sensibles lors de leur enregistrement.
Durant l'exécution, la mémoire de l'enclave et son exécution dans le processeur sont isolées par des mécanismes matériels.
Notre étude de Intel SGX n'aura pas la même portée que celle de ARM Trustzone, vu que nous allons viser un dispositif mobile utilisant un SoC sous l'architecture ARM, qui supporte Trustzone et non pas un ordinateur utilisant un processeur Intel.

Applications pratiques

Les applications de SGX sont similaires à celles de Trustzone dans le sens où des données sensibles sont traitées en isolation des autres applications et du système d'exploitation pouvant être infecté (comme, par exemple, le chiffrement de données dans une enclave). Mais le fait que SGX permet aux fournisseurs de services d'attester l'état et d'établir la confiance dans l'enclave ouvre la porte à une nouvelle catégorie d'applications non disponibles dans Trustzone, comme par exemple, dans Signal, une application de messagerie sécurisée qui as-

sure le respect de la vie privée. Pour faire la découverte des contacts qui ont la messagerie Signal installée, cette application propose de téléverser la liste des contacts dans un serveur et de faire les traitements nécessaires dans une enclave en isolation par rapport aux autres services. De plus, le client pourrait faire une attestation à distance du code qui effectuerait les traitements (code ouvert). Ce mode de traitement est l'inverse de l'application typique de l'attestation à distance avec SGX, où un serveur est chargé d'attester l'exécution de son code sur une machine cliente. Le cas le plus évident est le Digital Rights Management (DRM), dans lequel un fournisseur de service voudrait que son contenu soit protégé contre le vol par un tiers.

Défauts

Intel SGX a pris sa part de vulnérabilités, d'autant plus que le système d'exploitation n'est pas inclus dans le TCB. Donc, il n'est pas digne de confiance, mais il demeure tout de même responsable de gérer les ressources, ce qui facilite les attaques par canaux auxiliaires, comme dans Xu et al. (2015), où les auteurs ont réussi à détecter ce type d'attaque sur SGX et ont pu récupérer partiellement des images générées dans une enclave. Dans Weichbrodt et al. (2016), les auteurs ont conduit des attaques de synchronisation, où ils profitent du fait qu'ils peuvent interrompre et reprendre les *threads* de SGX. Ces attaques résultent de la synchronisation erronée d'accès aux données partagées par plusieurs *threads*. En manipulant les *threads*, ils peuvent changer le flux d'exécution du code s'exécutant dans une enclave SGX. Dans Shinde et al. (2015), les auteurs ont pu extraire des parties des clés cryptographiques utilisées dans une enclave. Ceci est fait en provoquant des erreurs dans les pages des processus des enclaves. Cette manipulation donne au système d'exploitation des informations sur les pages dans lesquelles les erreurs sont faites. Cette information permet d'extraire des bits de la clé, vu que le mécanisme de chargement des pages de la librairie de chiffrement (dans ce cas AES) est connu et il y a des valeurs statiques qui sont chargées dans ces pages.
Notons que ce qui constitue un point positif par rapport à ARM Trustzone, c'est que la mémoire allouée à une enclave est chiffrée de manière matérielle.

Conclusion

Dans ce chapitre, nous avons étudié le *trusted computing* et en particulier la technologie Trustzone de ARM. Dans le prochain chapitre, nous allons faire une comparaison entre les techniques possibles en matière d'authentification, et les comparer à notre proposition.

CHAPITRE 3 UN CAS D'USAGE DU TRUSTED COMPUTING : L'AUTHENTIFICATION

Introduction

Après avoir fait une étude de ce qui existe en matière de *trusted computing*, nous allons discuter des différentes approches d'authentification qui existent. Nous allons les comparer avec notre solution et conclure sur ce qui apporte le *trusted computing* par rapport aux autres solutions.

3.1 Méthodes d'authentification

Plusieurs travaux de recherche ont essayé de corriger les lacunes des mots de passe. Dans certains schémas, les auteurs ont proposé des modifications aux mots de passe comme l'authentification par mot de passe à usage unique. Dans d'autres, les schémas proposés remplacent complètement les mots de passe : les méthodes graphiques par exemple. Les principales méthodes proposées, et qui ont été adoptées dans l'industrie seront décrites dans les sections suivantes.

Authentification par mot de passe à usage unique

L'authentification par mot de passe à usage unique (utilisant essentiellement les téléphones cellulaires) commence à être adoptée dans l'industrie comme une solution pour l'authentification à deux facteurs. Ce schéma ne remplace pas les mots de passe, comme ils doivent être utilisés en tandem pour vérifier l'identité de l'utilisateur. En effet, comme expliqué dans Aloul et al. (2009), après l'entrée du mot de passe sur un navigateur de l'ordinateur, l'utilisateur génère un mot de passe pour usage unique (One Time Password (OTP)) à partir d'une application sur son téléphone ou il reçoit ce OTP par message texte. L'utilisateur l'entre sur le site web dans un intervalle de temps relativement court pour terminer l'authentification. Plusieurs fournisseurs de services et réseaux sociaux utilisent cette technique, notamment Google, Dropbox et Facebook.
Une autre technique, proposée par Parno et al. (2006), consiste à utiliser un téléphone cellulaire afin d'établir la session Transport Layer Security (TLS) avec le serveur en parallèle avec l'entrée du mot de passe sur le site. Cela garantit que le site est authentique et qu'il ne s'agit pas d'une tentative d'hameçonnage. Le cellulaire communique avec l'ordinateur via la technologie Bluetooth.

Jeton matériel

Cette technique, comme par exemple dans SecurId, remplace les mots de passe. Dans ce cas, nous passons de quelque chose que nous connaissons à quelque chose que nous avons. En fait, un petit dispositif portable génère, à partir d'une semence connue du serveur, un code chaque n seconde (n peut varier entre 60 et 3600), à entrer en concaténation avec un code statique comme mot de passe, avec l'identifiant de l'utilisateur.

Biométrie

L'utilisation de la biométrie pour l'authentification remplace le concept de mot de passe (nous remplaçons quelque chose que nous connaissons par quelque chose que nous sommes). La forme la plus utilisée est les empreintes digitales comme décrit dans le travail de Maltoni et al. (2009), mais il y a encore la reconnaissance de l'iris, de visage, etc.
La reconnaissance des empreintes digitales est une technologie qui a pris de la maturité ces dernières années. Nous la retrouvons dans les téléphones cellulaires afin de les déverrouiller ou de s'authentifier dans les applications bancaires. Mais la mise en place de mots de passe reste obligatoire pour assurer la rétro compatibilité et l'accès aux services en cas de dysfonctionnement du capteur d'empreintes. En conséquence, ce n'est pas un facteur d'authentification additionnel par rapport aux mots de passe, mais plutôt alternatif.

Graphique

L'authentification graphique se base sur quelque chose que l'utilisateur connaît, tout en profitant du fait que c'est plus facile pour la mémoire de retenir un graphe qu'un mot de passe. Il existe plusieurs schémas et techniques disponibles, comme par exemple dans la recherche de Chiasson et al. (2012), où les auteurs proposent une méthode dans laquelle l'utilisateur clique sur une partie de l'image, et selon l'endroit où il a cliqué, une autre image s'ouvre et la même manipulation se répète un certain nombre de fois ce qui constituera la séquence que l'utilisateur devrait répéter afin de s'authentifier.
Dans la publication de Tao (2005), la méthode propose une grille avec des points d'intersection. L'utilisateur doit créer une forme géométrique en connectant des points. Ensuite, la répéter chaque fois pour s'authentifier. Cette méthode a connu du succès et elle a été utilisée notamment dans le système d'exploitation pour mobile Android afin de déverrouiller l'écran.

Gestionnaire de mots de passe

Les gestionnaires de mots de passe proposent d'aider les utilisateurs à retenir les identifiants entrés sur les différents sites. Tous les mots de passe sont protégés par un mot de passe principal. Ces gestionnaires offrent la synchronisation sur le nuage afin que tous les mots de passe soient accessibles sur tous les appareils de l'utilisateur.
Une proposition de Li and Evans (2017) présente un gestionnaire de mot de passe particulier. En effet, ce gestionnaire, en plus de fournir la fonctionnalité de base qui est de gérer les mots de passe entrés, augmente la sécurité en fournissant de faux identifiants lors du remplissage des formes de connexion. Ensuite, avant le chiffrement des identifiants dans la requête «POST» le gestionnaire les remplace par les vrais. Cette manipulation permet une protection contre plusieurs types d'attaques, comme l'injection de scripts de l'attaquant dans des pages visitées.

3.1.1 Authentification en utilisant le *trusted computing*

Dans la littérature, nous n'avons pas trouvé de travaux majeurs essayant de tirer profit du *trusted computing* pour faire de l'authentification sur plusieurs types de dispositifs (PC, mobile, etc.).
Aussi, comme mentionné dans Florêncio et al. (2014) et Bonneau et al. (2015), les mots de passe, même s'ils ne sont pas la solution idéale d'authentification, sont là pour rester, au moins à court et moyen terme, vu l'absence d'alternatives qui pourraient corriger les problèmes et en même temps être faciles à déployer (surtout en termes de rétrocompatibilité).
Ces deux points nous mènent à penser que plutôt que d'essayer de trouver un nouveau schéma pour remplacer les mots de passe, nous devrions trouver un moyen de mieux les sécuriser. L'une des façons serait le *trusted computing* qui pourrait corriger plusieurs défauts des méthodes d'authentification actuelles. Dans le chapitre 3, nous allons faire une analyse des avantages et des inconvénients entre les différentes méthodes.

3.2 Critères de comparaison des méthodes d'authentification

Pour cette partie, nous nous sommes basés sur une étude comparative de Bonneau et al. (2012) qui non seulement regroupe les méthodes d'authentification dans des catégories bien définies, mais aussi des critères de convivialité, de déploiement et de sécurité concis pour les classer. Nous allons reprendre les définitions des critères. Ils seront utilisés dans le chapitre suivant dans la section 3.3.3 afin de faire une comparaison et une analyse des solutions d'authentifications incluant la nôtre. Ces critères seront expliqués dans les sections suivantes

3.2.1 Critères de sécurité

Ce sont les critères relatifs à différents aspects de la sécurité des solutions comme l'immunité contre certaines attaques ou la gestion des identifiants.

Résistance à l'observation physique : quelqu'un qui observe un utilisateur s'authentifier plusieurs fois (que ce soit directement ou à travers un dispositif d'enregistrement comme une caméra, par exemple) ne peut pas répéter l'authentification.

Résistance à l'imitation ciblée : la méthode résiste à l'imitation de la personne, en connaissant quelques informations comme la date de naissance par exemple.

Résistance à l'attaque par brute force limitée : une attaque par brute force limitée par un maximum de tentatives par le serveur ne peut pas avoir accès à des comptes.

Résistance à l'attaque par brute force non limitée : une attaque par brute force non limitée par un maximum de tentatives ne peut pas avoir accès à des comptes.

Résistance à l'observation interne : la méthode est résistante contre les menaces dans le dispositif de l'utilisateur, comme les virus et les *keyloggers* par exemple.

Résistance aux fuites dans les autres sites : si une fuite se produit dans un site, cela ne compromet pas les comptes de l'utilisateur dans d'autres sites.

Résistance aux attaques de type hameçonnage : un attaquant faisant une attaque de hameçonnage ne peut pas avoir les données d'authentification de l'utilisateur pour les utiliser après.

Résistance au vol : si l'authentification nécessite un objet physique, le vol de cet objet ne doit pas permettre au voleur de s'authentifier.

Pas de tierce partie : la méthode ne doit pas compter sur une tierce partie lors du processus d'authentification.

Demande de consentement explicite : le processus d'authentification doit avoir le consentement explicite de l'utilisateur.

Comptes pas connectables : deux sites qui se partagent les données connexions ne peuvent pas savoir à partir des données de connexions seulement si le même utilisateur se connecte sur les deux sites.

3.2.2 Critères de convivialité

Ces critères servent à comparer la facilité d'utilisation des méthodes par rapport à l'utilisateur.

Sans effort de mémoire : les utilisateurs ne doivent se souvenir d'aucun secret (même pas un mot de passe).

Extensibilité par rapport à l'utilisateur : l'application de la méthode sur plusieurs

comptes ne doit pas augmenter la charge sur l'utilisateur.
Rien à emporter : l'utilisateur n'emporte pas un objet physique pour utiliser la méthode.
Sans effort physique : l'authentification ne requiert pas d'effort physique (taper un mot de passe constitue un effort physique).
Facile à apprendre : l'utilisateur peut apprendre la méthode facilement.
Efficace à utiliser : l'utilisateur ne doit pas perdre beaucoup de temps dans le processus d'authentification.
Erreurs peu fréquentes : le processus d'authentification ne rejette pas régulièrement des utilisateurs légitimes.
Facile à récupérer : facilité à récupérer l'accès en cas d'oubli ou de perte.

3.2.3 Critères de déploiement

La facilité de déployer la méthode avec les infrastructures actuelles (au niveau du client et du serveur) est évaluée avec les critères suivants.
Accessibilité : accessible physiquement par tous les utilisateurs qui peuvent utiliser les mots de passe.
Coût négligeable par utilisateur : le coût de l'utilisation de la méthode est négligeable pour l'utilisateur et le fournisseur.
Compatible avec le serveur : la méthode doit être rétrocompatible avec les serveurs actuels utilisant les mots de passe texte.
Compatible avec le navigateur : l'utilisateur doit pouvoir s'authentifier en utilisant un navigateur commun à jour et sans extensions spécifiques.
Mature : la méthode a déjà été implémentée et déployée à grande échelle au-delà de la recherche.
Pas propriétaire : la méthode est ouverte à tout le monde sans devoir payer des frais.

3.3 Discussion des méthodes d'authentification

Notre méthode d'authentification sera discutée avec beaucoup de détails dans le chapitre 4. Dans cette section, nous analyserons les principales méthodes vues dans la revue de littérature. Celles qui ont eu le plus de succès et d'adoption dans l'industrie. Ensuite, nous discuterons comment la méthode proposée aurait des avantages importants par rapport aux autres solutions.

3.3.1 Analyse des solutions existantes

Authentification par téléphone

La solution la plus adoptée actuellement dans l'industrie, après les mots de passe, est la solution d'authentification à deux facteurs, utilisant les téléphones cellulaires. Cette méthode présente plusieurs avantages sur le plan de la sécurité et permet de contrer la majorité des menaces. Un attaquant doit infecter l'ordinateur et le téléphone cellulaire d'un utilisateur pour pouvoir se connecter à son compte. Bien que cette méthode ait été adoptée par l'industrie, elle n'est pas parrainée par tous les fournisseurs de services. Donc, contrairement à notre solution utilisable par tous les sites, elle requiert des changements côté serveur à tous les sites pour être fonctionnelle partout. L'autre atout à signaler, c'est le respect de la vie privée des utilisateurs. En effet si un site enregistre le numéro de téléphone, cela reste enregistré dans sa base de données et il y aura un risque que ce numéro soit public après une attaque informatique ou que le site l'utilise ou le vende pour des compagnies de publicité et de sondages. Pis encore, certains sites permettent de chercher des utilisateurs à partir de leur numéro de téléphone, ce qui peut dévoiler beaucoup d'informations en connaissant uniquement le numéro de téléphone. Il faut signaler que cette solution ne peut pas être utilisée dans le cas où l'utilisateur est en voyage et n'a pas de réception sur son téléphone, dans la variante où le OTP est envoyé par messagerie texte. Finalement concernant l'aspect financier, même si notre solution requiert un achat, celui-ci sera unique et d'un prix peu élevé. En revanche, pour utiliser la solution d'authentification à deux facteurs utilisant les téléphones cellulaires, il faut non seulement acheter l'appareil mobile (qui coûte plus que le dispositif utilisé), mais de plus cela nécessite un abonnement mensuel à un fournisseur de services de téléphone cellulaire. Cela présente un handicap pour les personnes ne voulant/pouvant pas s'en procurer pour utiliser cette méthode.

Biométrie

La méthode la plus utilisée impliquant la biométrie est celle qui scanne les empreintes digitales. Elle est performante, mais il est rare de la retrouver sur des ordinateurs (en particulier les stations de travail). Bien qu'assez courante sur les téléphones cellulaires, cette technologie doit tout de même être accompagnée d'un mot de passe dans le cas où le scanner poserait problème. De plus, pour être utilisée avec les sites web sur un ordinateur, plusieurs modifications doivent être faites, l'utilisation de l'empreinte se fait localement. Du point de vue de la sécurité, cette technique montre encore des faiblesses. La vérification des empreintes peut être contournée en utilisant les empreintes de l'utilisateur légitime récupérées et mises sur un

tissu par exemple. De plus, dans les implémentations actuelles, elle est utilisée en parallèle des mots de passe. Donc, si un logiciel malveillant (un enregistreur de frappes par exemple) parvient à obtenir le mot de passe de l'utilisateur après un redémarrage, il n'aurait pas besoin de l'empreinte digitale afin de s'authentifier.

Gestionnaire de mots de passe

En ce qui a trait aux gestionnaires de mots de passe, cette solution présente beaucoup d'avantages au niveau de la convivialité et du déploiement. Un seul mot de passe universel permet de chiffrer tous les mots de passe d'autres sites. Il y aussi le remplissage automatique des champs : il suffit d'un clic pour se connecter. Toutefois, cette solution n'est pas exempte de problèmes puisque l'utilisateur doit toujours utiliser le même navigateur. Mais les défauts les plus préoccupants sont liés à la sécurité. Les logiciels malveillants peuvent facilement avoir les mots de passe de l'utilisateur. Il suffit d'intercepter la requête *POST* sortante du navigateur pour obtenir les identifiants. Aussi, un attaquant peut faire une attaque de type force brute ou dictionnaire sur le fichier qui contient les mots de passe, si le mot de passe universel n'est pas bien choisi (une faible entropie). Cette opération peut se faire dans un temps raisonnable. Les logiciels malveillants de type *keylogger* permettent également de récupérer les mots de passe à leur première utilisation, ou encore mieux d'intercepter le mot de passe universel.

3.3.2 Méthode d'authentification proposée

Comme mentionné dans l'introduction, une idée qui a été développée en prototype, de notre collaborateur, Jim McAlear, consistait à mettre un dispositif embarqué entre le réseau internet et un ordinateur. Après, il faut modifier le protocole HyperText Transfer Protocol Secure (HTTPS) afin qu'il détecte la présence du dispositif en communiquant sur un port particulier. Lors de la connexion, la demande de nom d'utilisateur et mot de passe est affichée sur le dispositif, l'entrée de ces informations aussi se fait sur le dispositif. Nous avons repris l'idée d'avoir un dispositif entre l'ordinateur et l'internet, servant à entrer les mots de passe de l'utilisateur. Mais, en termes de rétrocompatibilité, nous avons jugé que la modification des protocoles existants sera très problématique pour un éventuel déploiement. Aussi, dans le but de mieux sécuriser le dispositif, nous avons utilisé le *trusted computing*.
Nous allons maintenant décrire le fonctionnement de la solution proposée : le concept général de notre prototype est simple. Nous aurons un dispositif embarqué qui fonctionne avec un processeur ARM (supportant la technologie Trustzone). Ce dispositif sera lié avec l'ordinateur à utiliser avec un câble Ethernet, et agira comme un proxy : les connexions de l'ordinateur vers l'internet passeront toutes par le dispositif. Le dispositif détectera les champs de connexions

dans les pages web visitées. S'il détecte un champ de connexion, il va demander à l'utilisateur s'il veut utiliser le dispositif ou pas. Si oui, l'utilisateur inscrira ses informations d'identification sur le dispositif. Ces informations seront mises dans la requête du client vers le serveur. La technologie Trustzone sera utilisée afin de sécuriser l'entrée de ces informations. Une autre fonctionnalité implémentée, mais dont l'activation est optionnelle par l'utilisateur, est la sauvegarde des mots de passe entrés en utilisant le stockage sécurisé de Trustzone. Chaque fois qu'un utilisateur se reconnectera au site, les informations d'identification seront entrées automatiquement afin d'ouvrir une session (comme un gestionnaire de mots de passe).

3.3.3 Comparaison avec les autres solutions

Nous allons maintenant faire une comparaison avec les méthodes décrites dans la section 3.1 en utilisant les critères utilisés par l'étude comparative de Bonneau et al. (2012) décrits dans la section 3.2. Le tableau 3.1 résumera la comparaison. À noter que la deuxième solution est la version où nous utilisons le gestionnaire de mots de passe dans le dispositif.

Tableau 3.1 – Comparaison des solutions d'authentification.

○ : Non, ● : Oui, ◔ : Presque

Critères \ Solution	**Authentification par téléphone**	**Jeton matériel**	**Biométrie**	**Graphique**	**Gestionnaire de mots de passe**	**Notre première solution**	**Notre deuxième solution**
Sécurité							
Résistance à l'observation physique	◔	●	●	○	◔	○	◔
Résistance à l'imitation ciblée	◔	●	○	●	◔	○	◔

Tableau 3.1 – Comparaison des solutions d'authentification.

Résistance à l'attaque par brute force limitée	●	●	●	◕	○	◕	◕
Résistance à l'attaque par brute force non limitée	●	●	○	○	○	◕	◕
Résistance à l'observation interne	○	●	○	○	○	●	●
Résistance aux fuites dans les autres sites	●	●	○	●	○	○	○
Résistance aux attaques de type hameçonnage	◕	●	○	●	●	◕	●
Résistance au vol	●	●	○	●	●	●	○
Pas de tierce partie	●	○	●	●	●	●	●
Demande de consentement explicite	●	●	●	●	●	●	●
Comptes pas connectables	●	●	○	●	●	●	●
Convivialité							
Sans effort de mémoire	○	○	●	○	◕	○	◕
Extensibilité par rapport à l'utilisateur	○	○	●	○	●	○	●
Rien à emporter	◕	○	●	●	◕	○	○
Sans effort physique	○	○	◕	○	◕	○	◕
Facile à apprendre	●	●	●	●	●	●	●
Efficace à utiliser	◕	◕	◕	◕	●	●	●
Erreurs pas fréquentes	◕	◕	○	◕	●	◕	●
Facile à récupérer	◕	○	○	●	○	●	○
Déploiement							
Accessibilité	◕	○	◕	○	●	●	●
Coût négligeable par utilisateur	○	○	○	●	●	○	○
Compatible avec le serveur	○	○	○	○	●	●	●
Compatible avec le navigateur	●	●	○	●	○	◕	◕

Tableau 3.1 – Comparaison des solutions d'authentification.

Mature	●	●	◕	○	●	○	○
Pas propriétaire	○	○	○	●	●	●	●

Discussion

Comme le montre le tableau 3.1, aucune solution ne cochera toutes les cases. Nous avons décidé d'avoir un bon compromis entre les différents critères. Pour la sécurité, comme notre solution se basera encore sur les mots de passe, nous avons essayé de contrer les menaces du côté du client, c'est-à-dire principalement les logiciels malveillants, *keyloggers*, etc. Aussi, nous proposons d'offrir un environnement d'exécution sécurisé afin d'avoir une protection dans le cas où un logiciel malveillant réussirait à passer au dispositif embarqué. Nous pouvons nous demander pourquoi ne pas faire une telle application sur téléphone cellulaire intelligent, par exemple, vu que le processeur sera du type ARM et supportera Trustzone. L'inconvénient avec cette solution est le fait que les téléphones sont utilisés pour accomplir plusieurs tâches (communiquer, naviguer sur internet, jouer, etc.). Ceci les rend vulnérables à plusieurs attaques à travers les magasins d'applications. Un logiciel malveillant pourrait donc récupérer les identifiants. En contraste avec notre solution, le dispositif sera utilisé seulement pour des fins d'identification et de sécurité. Donc les probabilités de se faire infecter sont très minces, voire nulles. Autre problème : le téléphone sera utilisé comme proxy, ce qui limitera son utilisation pendant la navigation sur internet, sans oublier l'aspect financier mentionné ci-haut. Notre méthode ne sera pas efficace, cependant, contre les attaques au niveau du serveur. Ceci est une conséquence de notre vision sur le déploiement. En cas d'adoption, nous avons essayé de garder la rétrocompatibilité. Donc, il n'y a pas de nécessité pour des changements majeurs pour le serveur et le client. Toute modification non rétrocompatible au niveau du serveur sera très coûteuse financièrement et diminuera drastiquement les chances d'adoption à l'industrie. Comme mentionné dans Florêncio et al. (2014) et Bonneau et al. (2015) les mots de passe, malgré leurs défauts, feront partie des solutions à court et à moyen terme.
Enfin sur le plan de la convivialité, cette solution ne sera pas difficile à utiliser vu qu'elle se base sur les mots de passe. Donc, les utilisateurs n'auront pas à apprendre une nouvelle méthode. Dans le cas où l'utilisateur déciderait de sauvegarder les mots de passe, elle sera plus conviviale, puisqu'elle sera presque sans effort de mémoire. L'utilisateur devra juste se rappeler du mot de passe d'authentification par rapport au dispositif.

3.4 Cas d'utilisation

Le but principal du dispositif est de fournir la possibilité de s'authentifier d'une manière sécuritaire à un service web. Dans un scénario de vie courante, il sera utilisé si nous ne faisons pas confiance à l'ordinateur que nous utilisons. Par exemple, si nous avons des doutes que notre ordinateur est infecté par un logiciel malveillant. Un autre exemple est son utilisation dans un voyage où nous utilisons un ordinateur de cybercafé et nous voulons accéder à notre compte bancaire en ligne : nous n'avons pas d'idée sur les logiciels installés, particulièrement les logiciels malveillants. De plus, les enregistreurs de frappes peuvent être matériels. Donc, l'utilisation de notre solution offrira une protection.
Sur le plan de la sécurité du dispositif, il y aura certainement des faiblesses. Cependant la protection physique du dispositif réduira grandement les risques de vol des données de l'utilisateur.

Conclusion

Dans ce chapitre nous avons discuté des différentes méthodes d'authentification. Nous avons aussi explicité les avantages et les inconvénients des principales solutions. Nous avons conclu qu'il faudrait toujours faire un compromis entre sécurité, convivialité et déploiement. Et en comparant notre méthode aux celles existantes, nous pouvons constater qu'elle offre des avantages aux niveaux sécurité et déploiement principalement puisque elle contre plusieurs types d'attaques et est rétrocompatible avec ce que existe. Aussi au niveau convivialité, il n'y a pas de nouvelles techniques à apprendre aux utilisateurs par rapport aux mots de passe. Dans le chapitre suivant, nous détaillerons notre implémentation du prototype.

CHAPITRE 4 CONCEPTION D'UN PROTOTYPE POUR L'AUTHENTIFICATION

Introduction

Ce chapitre contient les détails des implémentations que nous avons faites. Nous allons décrire notre système avec différents diagrammes montrant son fonctionnement et son architecture. Nous allons aussi justifier les choix techniques que nous avons pris en termes d'implémentation du *trusted computing* choisie et du matériel à utiliser.

4.1 Objectifs de conception

Notre objectif étant de proposer un dispositif qui nous permettra de nous authentifier de façon sécuritaire aux services web en utilisant le *trusted computing*, nous allons préciser les caractéristiques dont le système doit disposer dans le but de fournir un environnement d'exécution isolé et de protéger les données sensibles des utilisateurs :
Sécurité : pour la sécurité, le niveau envisagé n'est pas très sophistiqué, mais est capable de fournir une protection contre les attaques courantes qui ne nécessitent pas beaucoup d'argent ou du matériel de laboratoire très sophistiqué. Les attaques courantes contre lesquelles nous devons protéger notre système sont : les chevaux de Troie, les *keyloggers*, les attaques par canal auxiliaire pas très sophistiquées, etc.
Prix : le prix du dispositif (incluant tout ajout ou modification matérielle) devrait rester raisonnable. Cela est primordial pour espérer une adoption à grande échelle.
Facilité d'installation et de configuration : l'installation du dispositif ou de modules supplémentaires ne doit pas ajouter beaucoup de complications au niveau de l'installation que ce soit pour les utilisateurs ou les développeurs. Même si c'est uniquement pour cette tâche, cela reste très théorique vu le manque de documentation sur plusieurs aspects.
Nous allons récapituler les informations sur les solutions possibles dans le tableau 4.1 :

Suivant les informations récapitulatives dans le tableau 4.1, nous pouvons facilement écarter la première solution : module de sécurité matériel externe. En fait cette solution est très chère et offre un très haut niveau de sécurité, mais cela est plus adéquat pour des employés gouvernementaux que pour des utilisateurs "ordinaires". Donc, le choix est désormais entre les trois autres solutions. Le module de sécurité matériel interne et Trustzone se ressemblent dans leurs modes de fonctionnement. Tous deux offrent de l'exécution isolée. Cependant des arguments militent en faveur de Trustzone sans même tenir compte du prix, qui est légèrement

Tableau 4.1 Comparaison des solutions d'exécution isolée.

Solution	Sécurité	Prix (en $ canadiens)	Efforts d'installation et de configuration
Module de sécurité matériel externe	Très élevée	Élevé (+350)	Moyens
Module de sécurité matériel interne	Élevée	Moyen (100-150)	Importants
Virtualisation sur le même ordinateur	Très faible	Gratuit	Pas importants
Virtualisation sur un système embarqué	Faible	Moyen (50-100)	Moyens
Trustzone sur un système embarqué	Élevée	Moyen (50-100)	Moyens

plus bas pour Trustzone. En fait, pour Trustzone, la solution proposée est plus complète : gestion d'entrées/sorties pour le monde normal et monde sécurisé, exécution isolée à la même performance que l'exécution normale, facilité de communication et de passage entre le monde normal et le monde sécurisé à travers le moniteur sécurisé (contrairement au module de sécurité interne où la communication entre les deux processeurs se fait en transférant toute donnée à une mémoire externe). Un autre point en faveur de Trustzone, les possibilités offertes en termes d'applications possibles. En effet, les modules de sécurité internes ont des fonctionnalités limitées. Ils offrent principalement des fonctionnalités cryptographiques. En contrepartie, dans le cas d'exécution isolée fournie par Trustzone, il y a des API riches, fournissant des fonctionnalités variées, par exemple l'entrée sécurisée des données à partir du clavier directement dans le monde sécurisé. Pour finir, Trustzone a eu une très bonne adoption dans l'industrie, et il est utilisé pour sécuriser les téléphones cellulaires par exemple.
Cela nous laisse le choix entre deux approches : Trustzone et la virtualisation. Dans le deuxième cas, nous aurons deux possibilités : soit utiliser la virtualisation sur le même ordinateur que l'utilisateur emploiera pour se connecter, soit l'utiliser sur un système embarqué indépendant afin de fournir une isolation de l'exécution. Dans le cas d'exécution sur le même ordinateur, bien que cela représente un avantage économique pour l'utilisateur puisque le coût sera nul, ce sera très faible sur le plan de la sécurité. Par exemple un logiciel malveillant du type *keylogger* qui enregistre ce que se tape sur le clavier, pourrait voler les identifiants d'un utilisateur, même s'il est en train de taper sur une machine virtuelle. Dans l'autre cas d'exécution sur un système embarqué, le coût pour l'utilisateur sera similaire à la solution utilisant Trustzone. Même si la mise en place d'un système utilisant la virtualisation serait plus facile qu'une solution utilisant Trustzone, au point de vue sécurité (comme mentionné

dans la section 2.2.3) une telle solution serait beaucoup moins sécuritaire que Trustzone. Sans oublier le côté performance, vu qu'une machine virtuelle dans un environnement embarqué avec des ressources limitées sera moins performante qu'une solution native, utilisant les mécanismes d'isolation matériels et logiciels fournis par Trustzone.

4.2 Choix du matériel

La prochaine étape d'implémentation est de choisir le bon matériel. Nous allons expliciter les critères indispensables ou très importants afin de pouvoir implémenter le modèle. En nous basant sur ces critères, nous choisirons le système embarqué adéquat.

4.2.1 Critères

Processeur
Le premier critère est d'avoir un processeur avec une architecture ARM (presque valide pour tous les systèmes embarqués). Mais en plus, il doit implémenter les extensions de sécurité Trustzone. En effet, comme nous l'avons vu dans la section 4.1, le meilleur moyen de fournir de l'exécution isolée est d'utiliser les extensions de sécurité des processeurs. Cela offre un bon compromis entre sécurité et coûts d'achat et d'installation.

Interfaces réseau
Afin de fonctionner comme proxy, entre l'ordinateur de l'utilisateur et l'internet, le dispositif doit comporter au minimum deux interfaces réseaux, ou à défaut, la possibilité d'avoir une interface réseau à partir d'une autre interface comme Universal Serial Bus (USB), par exemple.

Prix
Afin d'espérer une adoption du modèle à grande échelle, son prix ne doit pas être très élevé pour un utilisateur moyen (avoisinant le prix d'accessoires informatiques et beaucoup moins cher que les téléphones cellulaires, moins de 100 $ canadiens). Donc, le prix sera un facteur à considérer, surtout si des produits satisfont tous les autres critères.

Documentation
Une documentation disponible sur internet et une large communauté d'utilisateurs faciliteront l'installation, la résolution d'éventuels problèmes. En effet, pour un matériel largement adopté, il est généralement bien maintenu avec toutes les mises à jour et les sources de logiciels. Aussi, c'est plus facile d'obtenir de l'aide sur un problème particulier.

Autres considérations
Parmi les autres considérations dans le choix du matériel : la taille du système embarqué. En effet, un système très grand peut ne pas être convenable au transport et à l'installation.

Aussi, une sortie vidéo standard comme Video Graphics Array (VGA) ou High-Definition Multimedia Interface (HDMI) sera nécessaire, au moins pour les premières utilisations. Finalement, un autre point important, la possibilité de faire du débogage, surtout pour les applications dans Trustzone où les informations et messages ne sont pas dirigés vers la sortie standard.

4.2.2 Solutions possibles

Il existe un grand choix pour les cartes de développement. Sur le marché, nous trouvons une centaine de cartes avec différentes configurations. Pour faire notre choix, nous allons commencer par voir, lesquelles ont une implémentation de Trustzone disponible et publique. Les systèmes restants sont résumés dans le tableau 4.2

Tableau 4.2 Comparaison des cartes électroniques incluant la technologie Trustzone

Carte	Processeur	Interfaces	Prix	Documentation	Disponibilité
HiKey	ARM Cortex A53	WIFI/Blutooth/ Ethernet/USB	95$ Canadien	Publique et assez riche	Disponible
Juno	ARM Cortex A72	WIFI/Blutooth/ Ethernet/USB	Non disponible	Publique mais pas riche	Non disponible directement
Mediatek MT8173	ARM Cortex A72/A53	WIFI/Blutooth/ Ethernet/USB	Non diponible	Non publique	Non disponible directement
Raspberry Pi 3	ARM Cortex A53	WIFI/Blutooth/ Ethernet/USB	45$ Canadien	Publique et très riche	Disponible

Vu que les systèmes Juno et Mediatek MT8173 ne sont pas disponibles aux marchés grand public et nécessitent de contacter les entreprises (généralement, ce n'est pas simple de s'en procurer) le choix reste limité principalement entre HiKey et le Raspberry Pi 3 puisqu'ils sont facilement accessibles sur la plupart des magasins d'électronique en ligne. Ces deux systèmes ont des configurations très similaires. En effet, ils ont le même processeur, la même taille de mémoire vive, les mêmes interfaces (HDMI, Ethernet, Wifi, Bluetooth, USB). Donc, le choix ne va pas se décider sur les détails techniques des deux cartes, mais plutôt sur les autres aspects. Le prix du Raspberry Pi 3 est moins que la moitié du HiKey. Donc, pour la même configuration, c'est évident que le Raspberry Pi 3 représente la meilleure affaire. Surtout que le prix du système joue certainement un rôle important dans une éventuelle adoption du prototype.
L'autre aspect est la documentation très riche du Raspberry Pi 3 par rapport au HiKey, étant donné le succès de vente impressionnant. Il existe énormément de forums de discussion,

de blogues d'information et de tutoriels sur internet.

4.2.3 Notre choix

Comme mentionné dans le paragraphe précédent, nous avons choisi le Raspberry Pi 3 afin d'implémenter notre prototype. Nous allons présenter ses caractéristiques.

Processeur

C'est un Quad Core cadencé à 1.2GHz, 64bit. Son architecture est ARM Cortex A53. Elle est caractérisée par l'ensemble d'instructions ARMv8-A et les extensions de sécurité ARM (2009). Le processeur est fabriqué par Broadcom.

Mémoire

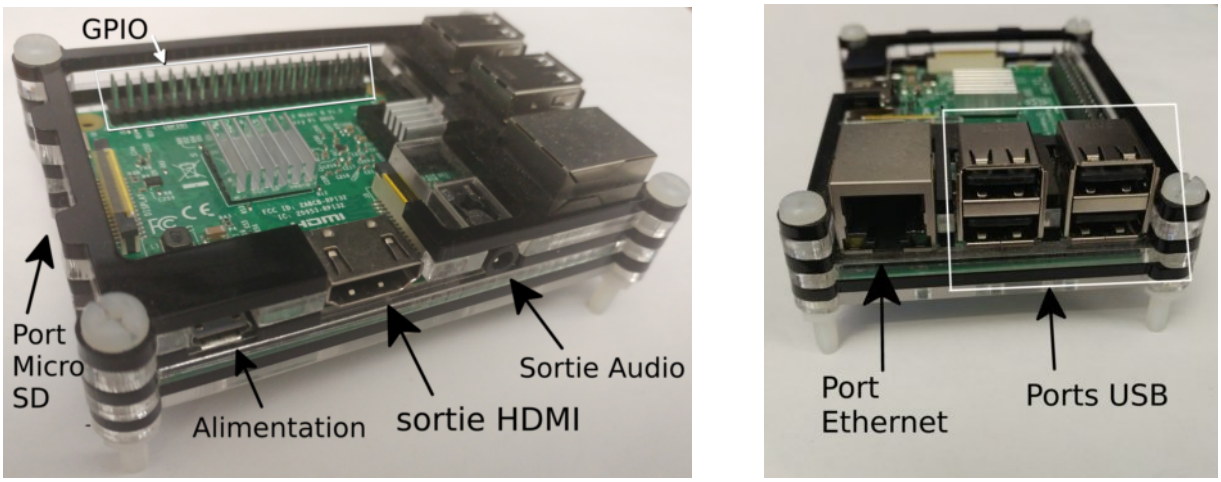
La mémoire vive est de 1 Gigabit. Pour la mémoire non volatile où le stockage permanent des données de l'utilisateur et du système d'exploitation se font, il y a un port de carte mémoire de type micro SD (Secure Digital (SD)). Donc, l'utilisateur peut utiliser une carte pouvant même aller jusqu'à 256 Gigabit. Concernant notre configuration, nous avons choisi une carte mémoire de 32 Gigabit, que nous voyons largement suffisante pour le système d'exploitation et les programmes à installer. Le fait que la mémoire non volatile soit sur une carte mémoire amovible facilite le développement, vu que le transfert de fichier est très simple à partir d'un ordinateur.

Interfaces

Les interfaces du Raspberry Pi 3 sont : un port Ethernet, 4 ports USB 2.0, Wifi, Blutooth, HDMI, prise audio analogique, 40 broches de General Purpose Input/Output (GPIO) (particulièrement utile dans le débogage où nous pouvons lire les sorties analogiques) et, bien sûr, un port d'alimentation électrique.

Dimensions

Les dimensions sont de 85,60 mm * 53,98 mm * 17 mm, le poids est de 45 g. La figure 4.1 montre le Raspberry Pi 3 avec les interfaces annotées.



(a) (b)

Figure 4.1 Les interfaces de la Raspberry Pi 3

4.3 Installation de Trustzone

L'étape suivante, après avoir choisi le système embarqué que nous allons utiliser, est l'installation du logiciel adéquat nous permettant de tirer profit de ARM (2009). Pour ça, nous avons cherché les implémentations de ARM (2009). La majorité de ces implémentations sont propriétaires et ne sont pas disponibles publiquement. En effet pour des raisons de compétitivité, chaque fabricant de systèmes préfère garder ses implémentations et secrets industriels fermés. La seule implémentation disponible, à code source ouvert, est : Open Portable Trusted Execution Environment (OP-TEE) accessible via le site OP-TEE.

4.3.1 Aperçu du OP-TEE

Le projet de OP-TEE était une solution propriétaire créée par STMicroelectronics et Ericsson. Ensuite, après avoir été maintenue et détenue par STMicroelectronics, ce projet a été passé à une firme de sécurité informatique Linaro qui l'a transformé en une solution de TEE open source. Ce projet constitue une implémentation complète d'un environnement d'exécution isolé. L'implémentation est disponible pour plusieurs cartes ainsi que pour les systèmes d'exploitation pour mobile Android, ainsi que Linux. Le principal objectif de conception est l'utilisation des mécanismes de ARM (2009) afin de fournir de l'exécution isolée du système d'exploitation riche et des autres applications en exécution isolée aussi. Mais il existe d'autres objectifs, tels qu'une taille réduite du TEE ou encore celui de faciliter la portabilité. Le code source est facilement accessible sur leur répertoire sur GitHub (Répertoire OP-TEE).
Le port du code du OP-TEE sur le Raspberry Pi 3 a été fait par Sequitur Labs, une compagnie

de sécurité qui se spécialise dans la sécurité des systèmes embarqués.

4.3.2 Fonctionnement du OP-TEE

L'architecture du OP-TEE et l'ensemble des API utilisées pour les différents types d'opérations sont implémentés sous les spécifications de GlobalPlatform. C'est une standardisation rigoureuse et complète qui vise à définir les spécifications du TEE et des opérations sécurisées.
L'architecture du OP-TEE est décrite dans la figure 4.2.

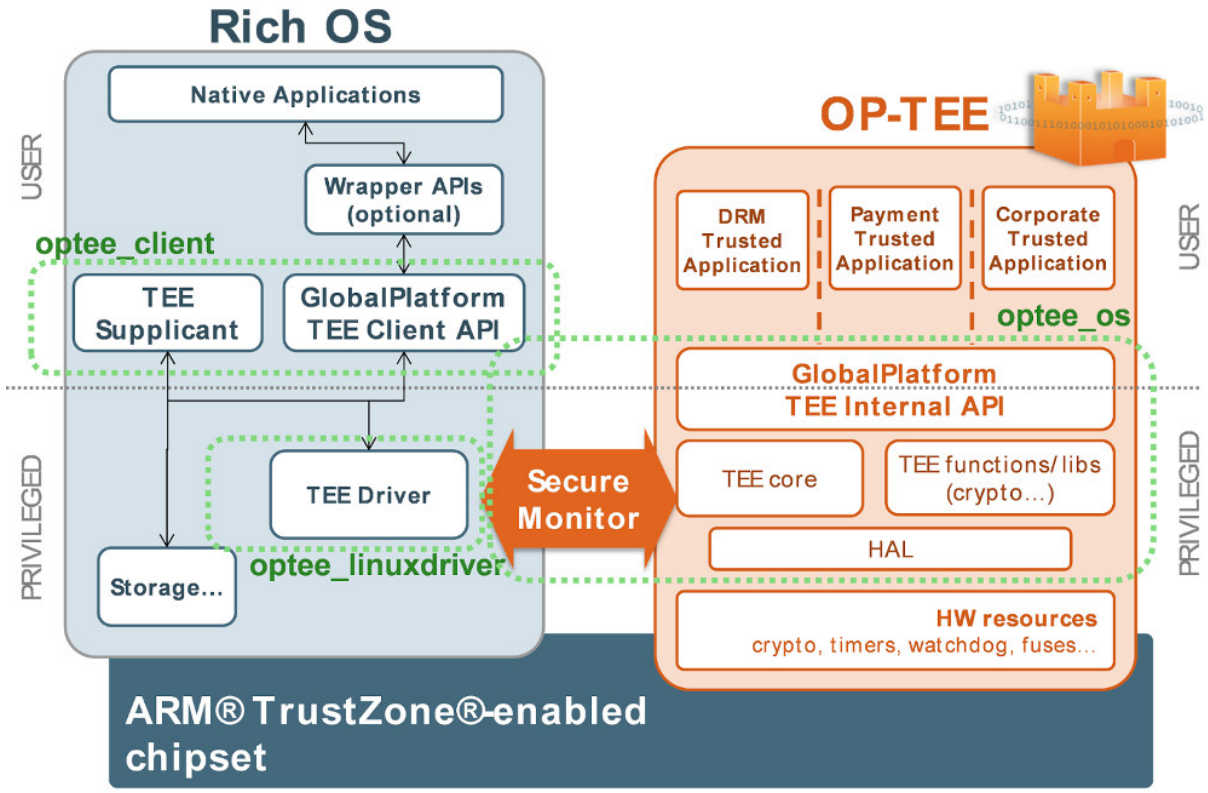


Figure 4.2 Architecture de OP-TEE, OP-TEE

Les principales composantes de ce TEE sont :

TEE Client API

C'est une API qui roule dans l'espace utilisateur du monde normal. Elle fournit le point d'accès de l'application client au TEE. C'est à travers cette interface que l'application du monde normal peut ouvrir une session pour une application de monde sécurisé.

TEE Driver

Le pilote du TEE est responsable de la communication entre le monde normal et le monde sécurisé. La communication se fait principalement par Remote Procedure Call (RPC).

OP-TEE OS

C'est le système d'exploitation sécurisé qui tourne dans le monde sécurisé. Il est constitué principalement de deux composantes : le noyau (Core) et un ensemble de librairies principalement de cryptographie destinées à être utilisées par les applications du monde sécurisé. Ce système d'exploitation est très minimaliste, sa taille ne dépassant pas le mégaoctet. Le système d'exploitation sécurisé doit avoir une petite taille pour principalement deux raisons, la première étant qu'il a plus de chance de pouvoir s'installer dans une petite mémoire sur la SoC, comme la SRAM, par exemple ; cela est plus sécuritaire qu'une installation en mémoire *flash*. L'autre raison est de réduire les probabilités d'avoir des bogues vu la petite taille. Un bogue dans le système d'exploitation sécurisé pourrait avoir des conséquences très dommageables.

4.3.3 Installation de OP-TEE

L'installation de OP-TEE sur le Raspberry Pi 3 se fait en quatre grandes étapes :

1 : La première étape consiste à installer les dépendances nécessaires sur un ordinateur afin de le préparer pour construire et compiler le logiciel de OP-TEE. Parmi ces dépendances, des libraires de compression, de cryptographie et de construction (*build*).

2 : Ensuite, nous devons télécharger le code de OP-TEE relatif au système choisi depuis leur répertoire. Dans notre cas, le système est Raspberry Pi 3. Nous devons aussi télécharger la chaîne de compilation (*toolchain*). Elle est indispensable, puisque nous allons faire de la compilation croisée. C'est à dire, nous allons compiler sur un ordinateur (architecture x64) du code destiné à s'exécuter sur une autre architecture (dans notre cas arm64).

3 : La prochaine étape est simplement de compiler le code de l'OP-TEE. Nous utiliserons la commande «Make». Cette commande générera un *bootloader*, un noyau Linux (pour l'environnement riche) qui fournira les mécanismes de communication entre le REE et le TEE et enfin, le système sécurisé OP-TEE.

4 : Finalement, il ne nous reste qu'à formater la carte micro SD en deux partitions, une de type File Allocation Table (FAT), et une Extended File System (EXT). Ensuite, mettre les fichiers générés dans ces deux partitions. Les fichiers de démarrage iront dans la petite partition FAT. Les fichiers du système d'exploitation iront dans la partition EXT. Il y a un ensemble de tests à exécuter afin de vérifier si le TEE est bien en place et qu'il n'y a pas de problèmes de matériel.

4.4 Prototype

Le développement de notre prototype s'est déroulé sur deux étapes. La première est le développement du proxy, sans utiliser le *trusted computing*. La deuxième est l'introduction du *trusted computing* dans le fonctionnement du proxy.

4.4.1 Sans *trusted computing*

La première étape consiste à faire fonctionner le Raspberry Pi 3 comme proxy entre l'ordinateur de l'utilisateur et l'internet. La figure 4.3 montre la mise en place des différents dispositifs. Le Raspberry Pi 3 a son propre affichage et son propre clavier indépendant de celui de l'ordinateur.

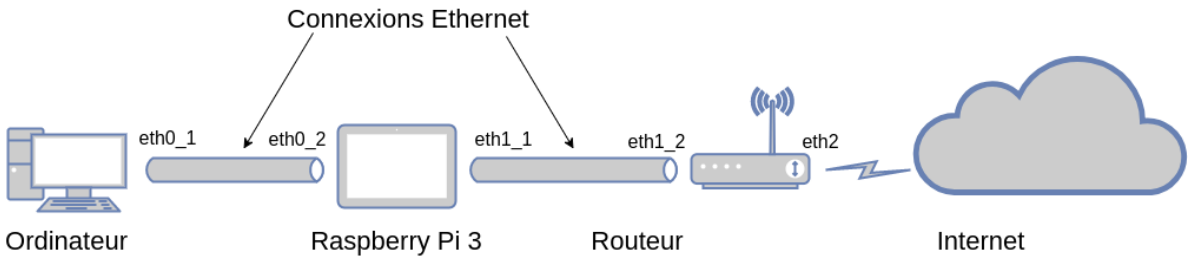


Figure 4.3 Mise en place des dispositifs

Configuration

Dans cette mise en place, tout le trafic réseau de l'ordinateur passera par le Raspberry Pi 3. Tout d'abord, nous avons configuré ces deux dispositifs à communiquer entre eux. Nous avons assigné des adresses Internet Protocol (IP) statiques, pour l'interface de l'ordinateur eth0_1 et celle du Raspberry Pi 3 liée à l'ordinateur eth0_2. Ce faisant, nous n'aurons pas à configurer un proxy dans les paramètres du système d'exploitation ou du navigateur que nous allons utiliser.
La prochaine étape est de rediriger le trafic entre les deux interfaces du Raspberry Pi 3, afin que l'ordinateur puisse se connecter à internet. Nous avons utilisé les commandes réseau de Linux. Nous avons commencé par activer le transit IP (IP *forwarding*). Ensuite, nous avons écrit des règles pour la IP table de Linux. Premièrement, accepter le transit (*Forward*) entre eth0_2 et eth1_1. Puis, accepter le transit pour les connexions déjà établies (*state established*). Après, changer l'adresse source des paquets sortants avec l'option «*masquerade*».

Finalement, comme notre proxy va écouter les connexions sur le port 8080, nous avons redirigé le trafic entrant sur les ports 80 (HyperText Transfer Protocol (HTTP)) et 443 (HTTPS) vers le port 8080. Ces règles vont nous assurer que le trafic réseau circulera entre les deux interfaces. Évidemment, la deuxième interface du Raspberry Pi 3 est connectée au routeur, et permet donc de fournir l'accès à l'internet. L'adresse IP de l'interface eth1_1 est obtenue via Dynamic Host Configuration Protocol (DHCP).
Ainsi notre ordinateur a accès à l'internet, et cela, seulement à travers notre Raspberry Pi 3. Pour l'instant, il n'y a aucune modification du trafic réseau. Notre dispositif embarqué ne fait que transférer les paquets réseau.

Mitmproxy

En vue de faciliter l'interception et la modification du trafic (surtout le trafic HTTPS), nous avons utilisé une librairie codée en Python, qui s'appelle mitmproxy. Elle offre beaucoup de fonctions dans l'optique d'intercepter, d'inspecter ou encore de modifier le trafic réseau. L'outil en ligne de commande permettant de faire ces tâches s'appelle mitmdump. Intercepter le trafic HTTP est très simple même dans un mode transparent par rapport à l'utilisateur, étant donné que les paquets ne sont pas chiffrés. Mais maintenant, ce mode n'est quasiment plus utilisé pour effectuer une connexion puisqu'il n'est pas sécurisé. En effet, un attaquant peut facilement s'emparer des identifiants juste en se connectant sur le même réseau sans fil et en sauvegardant les paquets après les avoir détournés à travers sa machine. En revanche, pour le trafic HTTPS, c'est plus compliqué. Avant d'expliquer le fonctionnement du proxy, nous allons rappeler le fonctionnement général de l'établissement de la connexion du protocole HTTPS.
L'établissement de la connexion est l'étape préliminaire où le client et le serveur se mettent d'accord sur les méthodes de chiffrement à utiliser. Aussi, le client s'assure qu'il communique avec le bon serveur en vérifiant son identité (parfois l'inverse aussi, c'est-à-dire que le serveur vérifie l'identité du client). Le déroulement de l'établissement de la connexion («*handshake*») se fait comme suit :
1 : Le client envoie un premier message «*ClientHello*» contenant la version du protocole à utiliser (Secure Sockets Layer (SSL)/TLS), les méthodes de chiffrement possibles, les méthodes de compression et une valeur aléatoire.
2 : Le serveur répond avec un «*ServerHello*» qui contient les méthodes choisies selon les préférences du client (les méthodes de chiffrement et de compression envoyées dans le «*ClientHello*» étant triées par ordre de préférences) et une valeur aléatoire.
3 : L'étape suivante sert à vérifier l'identité du serveur auprès du client. Il envoie son certificat SSL. Le client vérifie bien que le certificat a été émis par un tiers de confiance, tout en

vérifiant aussi la date de validité. Dans des cas très particuliers, un serveur peut demander un certificat auprès du client pour l'identifier.
4 : Le client crée une clé de chiffrement pré-maître, selon la méthode choisie, la chiffre avec la clé publique du serveur, et l'envoie. Seul le serveur peut déchiffrer ce message avec sa clé privée.
5 : Le client et le serveur génèrent la clé de session maître à partir de la clé pré-maître.
6 : Les deux parties s'échangent des messages indiquant que les prochains échanges seront chiffrés avec la clé maître.
En conséquence, le mode évident du proxy serait juste de transférer les paquets entre le client et le serveur. Mais ceci ne va pas nous donner la possibilité d'inspecter ou de modifier le trafic puisqu'il sera chiffré. Le seul moyen possible de faire ainsi est de réaliser une attaque de la personne au milieu (*Man In The Middle*). Pour ce faire, le proxy fait deux «*handshakes*». Une avec le serveur et une avec le client. Ensuite, il transmet les paquets entre les deux.
Vu que le protocole TLS n'est pas encore cassé, le client sera conscient du mitmproxy et devra installer ses certificats sur l'ordinateur à utiliser, ce qui constitue une limitation. Cette technique permettra d'inspecter et de modifier le trafic HTTPS sur le Raspberry Pi 3 avec l'aide de mitmproxy. À chaque installation de mitmproxy, ce dernier va générer un certificat racine dans son répertoire. Le certificat sera utilisé pour signer des certificats pour chaque connexion TLS. Évidemment, pour chaque installation différente, le certificat sera différent. De plus, l'utilisateur pourra générer son propre certificat et l'utiliser à la place de celui généré automatiquement par mitmproxy.

Fonctionnement

Nous allons maintenant décrire le fonctionnement de notre programme.
Notre prototype, agira comme proxy entre l'ordinateur et le Raspberry Pi 3. Nous allons, à chaque page web demandée, détecter s'il y a des champs de connexion de l'utilisateur. Le cas échéant, le chargement s'arrête. Un message sera affiché à l'utilisateur sur l'écran lié au Raspberry Pi 3, lui demandant s'il veut utiliser son dispositif pour entrer ses identifiants ou continuer le processus normal d'authentification sur l'ordinateur. Dans ce dernier cas, il n'y aura plus d'actions à faire, à part transmettre les paquets entre l'ordinateur de l'utilisateur et le serveur (pour le site visité seulement). Dans l'autre cas, le chargement de la page va continuer. L'utilisateur devrait, lors de la connexion au site, mettre des identifiants factices dans les champs de connexion. Ceci est dû au fait que la majorité des sites effectuent des contrôles pour voir si les champs ne sont pas vides avant d'envoyer la requête de connexion. La requête POST qui contient les identifiants sera interceptée. L'utilisateur sera invité à taper ses identifiants sur le Raspberry Pi 3. Ensuite les valeurs factices seront remplacées par

les vrais identifiants dans la requête.
Les figures 4.4 et 4.5 montrent le diagramme de séquences et le logigramme.

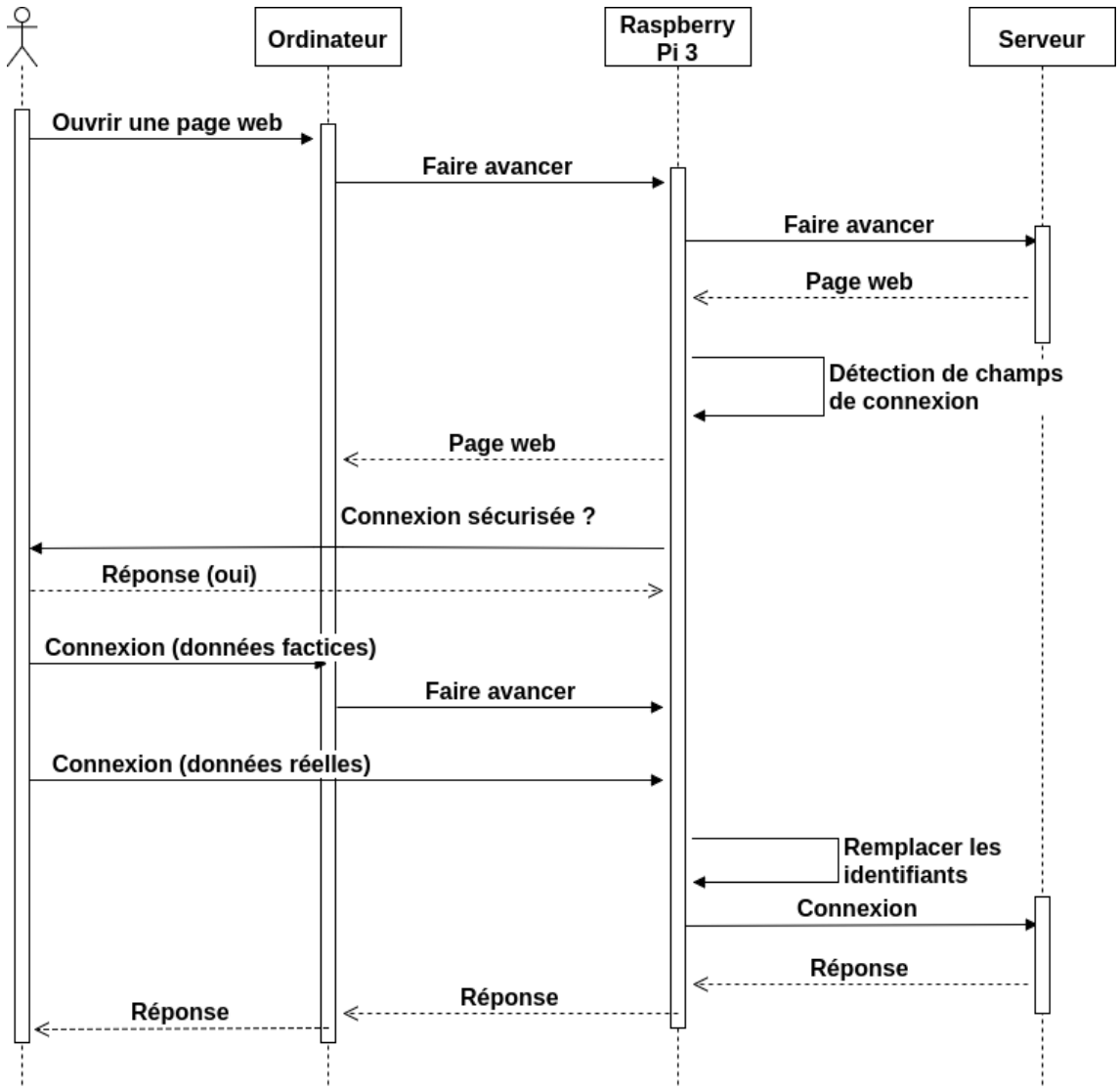


Figure 4.4 Diagramme de séquences modélisant le déroulement d'une connexion sécurisée

Dans la figure 4.4, nous voyons le cas où un utilisateur choisit notre méthode afin de se connecter à travers un formulaire de connexion sur un site web. Il répondra donc oui à la question du système s'il veut utiliser la connexion sécurisée. À noter que l'utilisation du Raspberry Pi 3 pour entrer les identifiants de connexion est indépendante entre les sites visités. C'est-à-dire, l'utilisateur peut choisir, pour chaque site visité, s'il veut ou pas utiliser la connexion sécurisée. Ceci est particulièrement utile dans le cas où il visite plusieurs sites, mais il ne leur accorde pas le même degré de sécurité. Par exemple, se connecter sur le site de la banque est plus critique que d'ouvrir une session dans un site de divertissement, ainsi

il prendra la décision en conséquence.
La figure 4.5 montre le logigramme de notre solution.

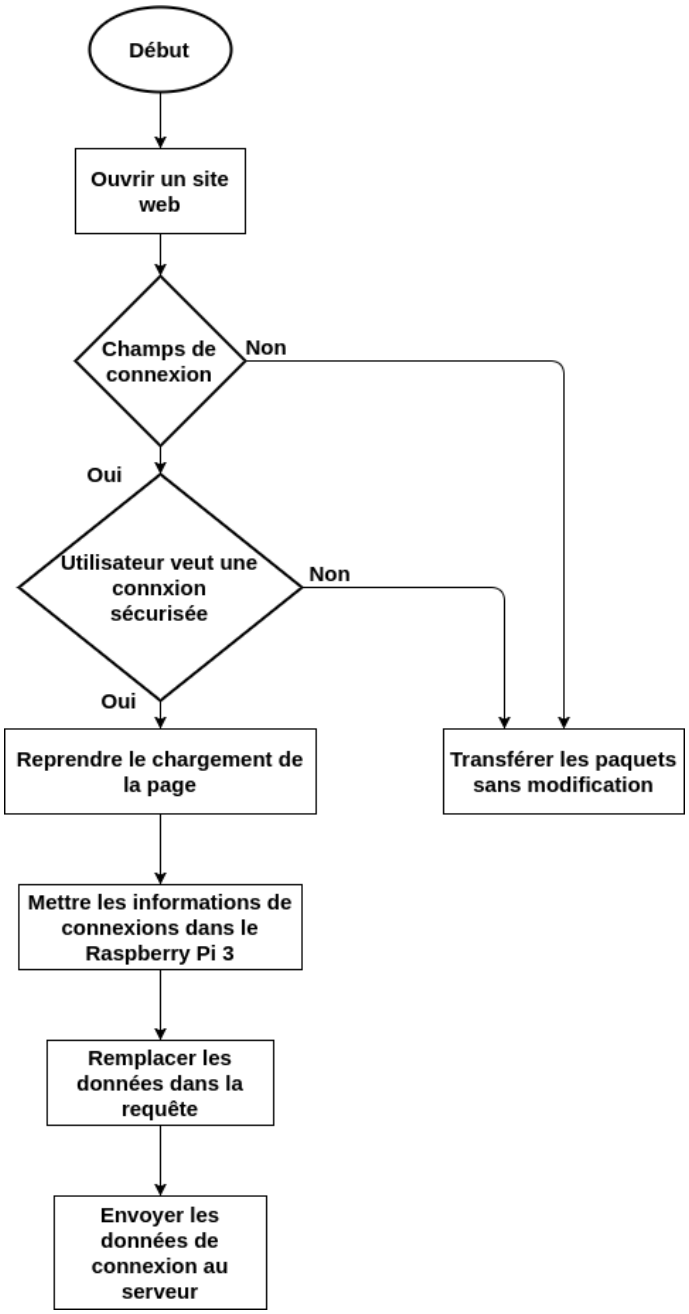


Figure 4.5 Logigramme du programme

Nous notons qu'à chaque demande de réponse de la part de l'utilisateur, le flux de connexion sera stoppé en attente. Aussi pour l'instant le *multithreading* n'est pas supporté. Par conséquent, les visites des pages web, pour se connecter, se font d'une manière séquentielle. La

recherche des champs de connexion et le remplacement des données de connexion dans la requête se font en utilisant les expressions régulières. Dans cette configuration, les cookies sont encore gérés par le navigateur du client sur l'ordinateur. Les cookies sont des fichiers stockés, qui jouent le rôle d'un jeton permettant de garder des informations sur la navigation. Ils sont utilisés avec les protocoles HTTP/HTTPS puisqu'ils sont sans état («*stateless*»), c'est-à-dire que les requêtes sont indépendantes et isolées les unes des autres. Parmi les utilisations des cookies, notons celle de garder active la connexion à un site, de façon à ce qu'un utilisateur n'ait plus à entrer ses identifiants à chaque fois qu'il visite le site, car la valeur du cookie, qui est stockée aussi dans le serveur, permettra de l'authentifier.

4.4.2 Avec *trusted computing*

L'idée générale étant implémentée et testée, nous allons passer à introduire le *trusted computing* dans notre programme. L'un des objectifs visés était l'utilisation de l'entrée sécurisée à partir du clavier. Cette fonctionnalité n'a pas été implémentée pour des raisons techniques hors de notre portée. Les détails seront discutés dans la section 5.3. Nous allons donc utiliser une autre caractéristique du *trusted computing* : le stockage sécurisé pour offrir des aspects fonctionnels et sécuritaires. Tout d'abord, nous détaillerons le fonctionnement des applications sécurisées avec OP-TEE.

Applications sécurisées

Les applications sécurisées sont composées de deux parties : une partie client à l'espace utilisateur de Linux et une partie sécurisée qui va s'exécuter dans le monde sécurisé. Les deux applications sont écrites avec le langage «C» en utilisant l'API de GlobalPlatform.

Partie client :

Cette partie sert comme interface par laquelle nous allons exécuter des fonctions sécurisées sur nos données. À partir de cette application, nous allons appeler l'autre partie, qui sera chargée dans le TEE. Dans le listage 4.1, nous montrons une portion de code situé dans l'application côté monde normal, dans laquelle nous allons :

1 : initialiser un contexte : permet de communiquer avec le TEE avec le pilote présent dans le monde normal (le TEE_driver dans la figure 4.2) (ligne 10).

2 : ouvrir une session : ouvrir une session avec l'autre partie de l'application (partie sécurisée) dans le TEE. Il faut, évidemment, entre autres fournir l'identificateur unique de l'application sécurisée (ligne 18).

3 : invoquer une commande : cette étape appelle l'instance de l'application sécurisée avec le nom de la méthode à utiliser et les paramètres correspondants (ligne 31).

À la ligne 26, nous spécifions les types des paramètres (paramètre en entrée au monde sécurisé, en sortie du monde sécurisé ou en entrée sortie).
À la fin, après l'exécution des fonctions, il faut détruire le contexte et la session créés.

Listing 4.1 Portion de code pour appeler une application sécurisée

```
1 #include <tee_client_api.h>
2 /* OP-TEE TEE client API*/
3
4 #include <hash_verif_ta.h>
5 /* Contient l'identifiant unique et les identifiants des fonctions */
6
7 int main()
8  {
9       /* Initialiser un contexte pour se connecter au TEE */
10       res = TEEC_InitializeContext(NULL, &contexte);
11       if (res != TEEC_SUCCESS)
12               errx(1, "TEEC_InitializeContext failed with code 0x%x",
   res);
13
14       /*
15        * Ouvrir une session a l'application securisee,
16        * l'identifiant de l'application est le troisieme parametre.
17        */
18       res = TEEC_OpenSession(&contexte, &session, &uuid,
   TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);
19       if (res != TEEC_SUCCESS)
20          errx(1, "TEEC_Opensession failed with code 0x%x origin 0x%x",
   res, err_origin);
21
22        /*
23        * Preparer les parametres, et indiquer le type de chaque
24        * parametre
25        */
26       operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT,
   TEEC_VALUE_INPUT, TEEC_VALUE_OUTPUT, TEEC_NONE);
27
28    /*
29     * Appeler la fonction cible
30    */
31    res = TEEC_InvokeCommand(&session, fonction_id, &operation, &
   err_origin);
32 }
```

Partie sécurisée :
Cette partie est la plus critique. Elle va se charger et s'exécuter dans le TEE. Impérativement, pour chaque application il faudrait implémenter cinq fonctions :
1 : création d'un point d'entrée à l'application : pour faire des initialisations et allouer des ressources.
2 : point d'entrée de session : vérification du client et des paramètres.
3 : point d'entrée de l'invocation de commande : déterminer la fonction à exécuter et l'appeler.
4 : fermeture du point d'entrée de session.
5 : destruction du point d'entrée à l'application.
L'application sécurisée est signée par une clé privée de chiffrement asymétrique. OP-TEE vérifiera ensuite avec une clé publique. Une autre alternative serait de mettre le contenu de l'application sécurisée directement dans le système d'exploitation sécurisé de OP-TEE. Ainsi invoquée, elle serait chargée directement du monde sécurisé au lieu d'être authentifiée à partir du monde normal. Nous avons opté pour la première solution, soit la signature numérique.
Compilation :
Nous avons compilé ces programmes sur notre ordinateur de laboratoire. Nous avons fait de la compilation croisée (*cross-compilation*) : nous avons compilé du code pour une architecture ARM-v8 à partir d'un ordinateur où l'architecture est x64. Dans cette compilation, il faut aussi inclure les API de OP-TEE. Au moment de la compilation du TEE sécurisé OP-TEE, il y a installation de la chaîne d'installation avec un kit de compilation pour le OP-TEE.
Un autre bon aspect de développement, est l'obligation de respecter un standard d'écriture de code sinon il n'y aura pas de compilation. Comme exemple à ces règles l'obligation de séparer les déclarations de variables des instructions. Aussi tous les avertissements sont traités comme des erreurs. C'est-à-dire que s'il y a des avertissements, il n'y aura pas de compilation réussie.
Débogage :
Le débogage des applications sécurisées n'est pas évident. En effet, il n'y a pas de possibilité d'afficher sur la sortie standard à partir du monde sécurisé. Ainsi, pour pouvoir déboguer les applications sécurisées, nous devons passer par l'interface GPIO. Cette interface offre la possibilité de se connecter à un ordinateur via l'interface Universal Asynchronous Receiver Transmitter (UART). Cela nous permet, à l'aide d'un émulateur de terminal, de lire la sortie en série du Raspberry Pi 3. Le mode débogage du OP-TEE étant configuré pour utiliser cette interface comme sortie pour le monde sécurisé. La pièce matérielle utilisée pour réaliser cette opération est un câble avec un bout USB et un bout qui se connecte sur deux broches des connecteurs GPIO.
Évidemment, pour des raisons de sécurité, le débogage du monde sécurisé doit être utilisé

seulement dans un environnement de développement et de test. Dans un environnement de production, il doit être désactivé en changeant des paramètres de compilation, sinon un attaquant pourrait voir ce qui se passe dans le monde sécurisé.

Stockage sécurisé

Le stockage sécurisé est un aspect très important du *trusted computing*. En effet, c'est une des bases du *trusted computing*, parce que c'est quasi-impossible de le réaliser sans pouvoir stocker de manière sécuritaire les données (en particulier les clés de chiffrement). OP-TEE offre du stockage sécurisé qui garantit trois propriétés de sécurité : la confidentialité des données, leur intégrité et l'atomicité des opérations. La confidentialité des données c'est-à-dire que les données ne sont pas visibles et accessibles, sauf aux utilisateurs qui ont en le droit. Quant à l'intégrité, cela signifie que nous pouvons nous assurer que les données ne seront pas modifiées de manière non autorisée sans que nous le sachions. Finalement, l'atomicité de l'opération implique que si l'opération n'est pas terminée avec l'état de succès en retour, il n'y aura pas d'écriture.
Le stockage sécurisé utilise le système de fichiers de Linux pour stocker les données. Il est effectué par une série d'opérations.

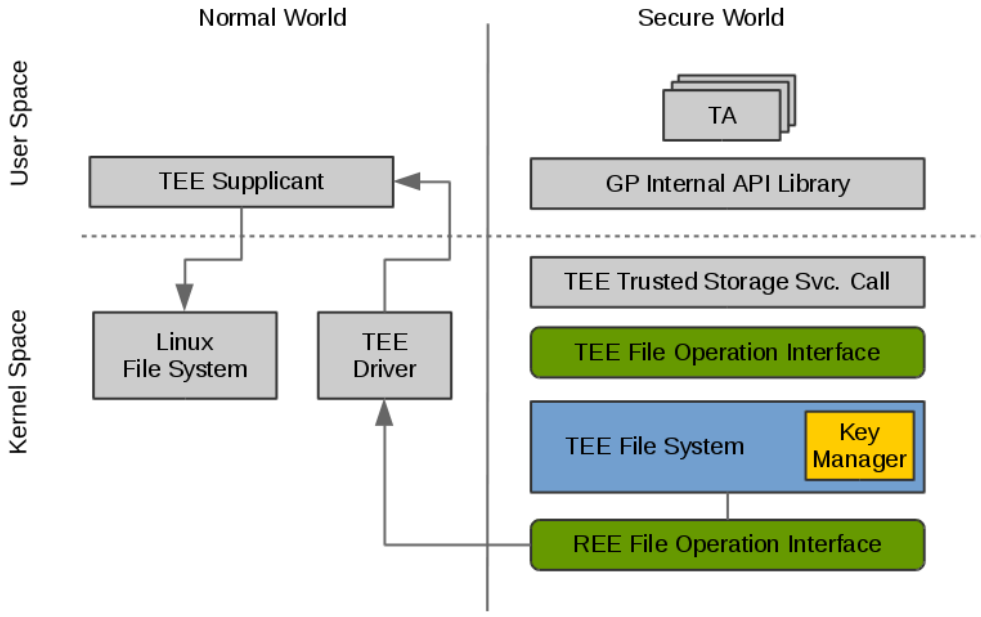


Figure 4.6 Architecture du stockage sécurisé, Répertoire OP-TEE

Comme nous voyons dans la figure 4.6, Il y a une chaîne d'opérations effectuée à chaque sauvegarde sécurisée. Il y a appel aux services de stockage sécurisé chaque fois qu'une application appelle les fonctions d'écriture à des objets persistants. Ensuite, le système de fichiers du TEE (TEE file system) va chiffrer les données. Il va les faire parvenir au pilote du TEE à travers des messages RPC. Finalement, les données vont être stockées sur le système de fichiers Linux.
La gestion des clés est effectuée par un gestionnaire de clés (*key manager*). Il est responsable, entre autres, de gérer les clés de chiffrement. Il y a usage de trois clés principalement :
Clé de stockage sécurisé : c'est une clé unique par dispositif. Elle est générée à partir des identifiants matériels du SoC, ensuite stockée dans le OP-TEE.
Clé de stockage de l'application sécurisée : c'est une clé relative à une application sécurisée. Elle est générée à partir de la clé de stockage sécurisé et de l'identifiant de l'application sécurisée.
Clé de chiffrement de fichier : cette clé est utilisée pour chiffrer et déchiffrer les fichiers créés à partir de l'application sécurisée. Elle est générée par une fonction de génération de nombres pseudo aléatoires et elle est chiffrée par la clé de stockage de l'application sécurisée.
À noter que l'implémentation des fonctions permettant l'extraction des clés matérielles est laissée au développeur, vu que chaque SoC a ses propres caractéristiques.

Fonctionnement

Ayant expliqué le fonctionnement de OP-TEE et du stockage sécurisé, nous allons maintenant expliquer comment nous les avons intégrés dans notre prototype. La logique générale est expliquée dans le nouveau logigramme 4.7. Nous avons utilisé ces fonctionnalités pour principalement deux tâches : la vérification d'intégrité de notre programme et la sauvegarde des données de connexions de l'utilisateur si désiré.
La vérification d'intégrité se fait au lancement du programme. Nous avons utilisé les fonctions de hachage cryptographique pour la faire. La fonction de hachage que nous avons choisie est : SHA-3 (Keccak) avec la taille du haché 512 bits. En fait, même si SHA-2 n'a pas d'attaque significative connue contre elle, SHA-3 est plus récente et en contraste elle est une solution dont le fonctionnement est ouvert. Pour notre solution, nous avons calculé le haché de notre programme et nous l'avons stocké de manière sécuritaire avec OP-TEE. Ensuite, dans chaque démarrage de l'application, nous calculons le *hash* et le comparons avec la valeur stockée. La comparaison se fait dans le monde sécurisé. Celui-là retournera, soit que les deux hachés sont égaux, soit qu'ils ne le sont pas. Évidemment, si les deux hachés ne sont pas égaux, ça veut dire qu'il y a eu modification du script.

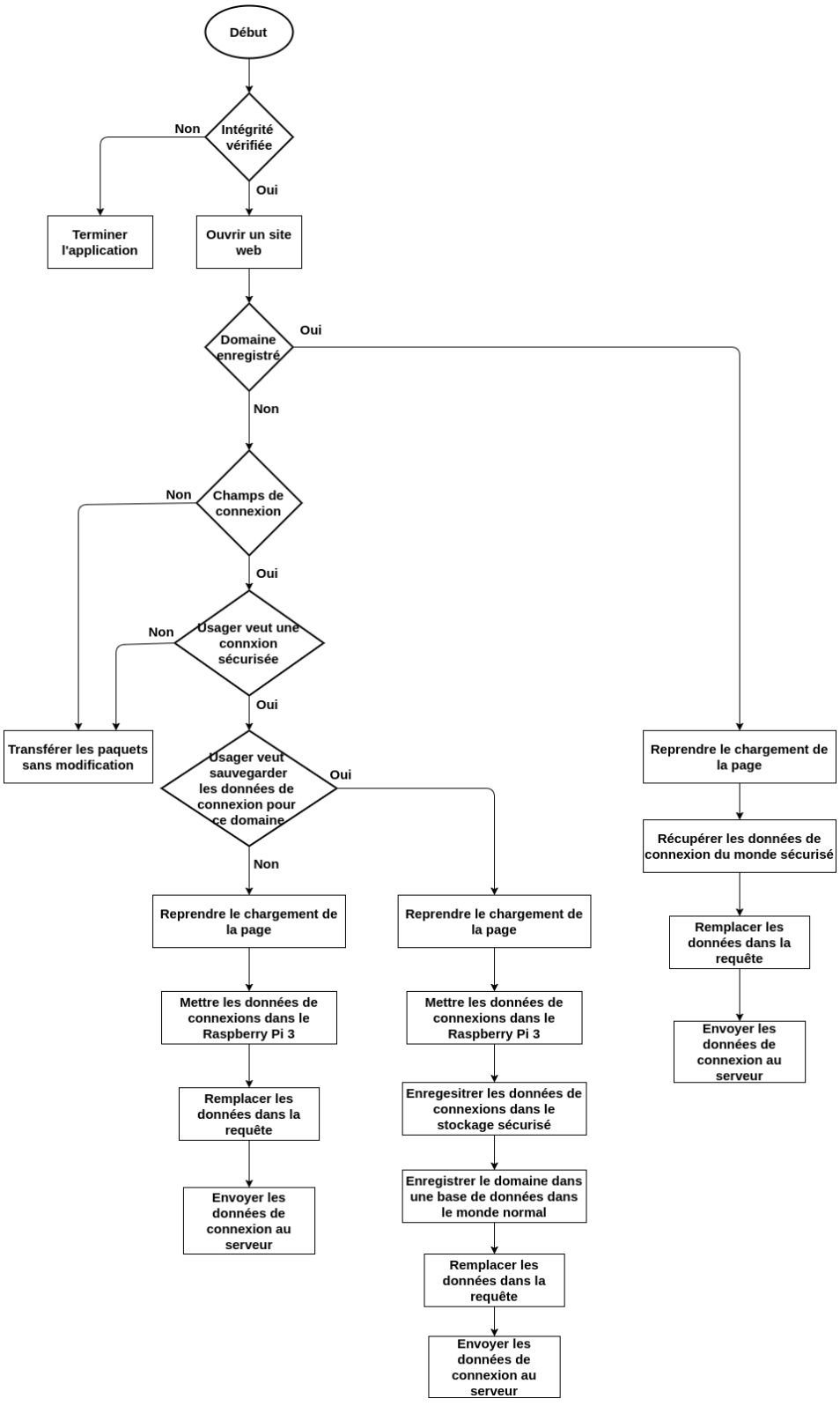


Figure 4.7 Logigramme du programme en utilisant le *trusted computing*

Nous avons aussi crée un exécutable léger dont la fonction est de vérifier l'intégrité du script. Un utilisateur pourrait simplement le copier et l'utiliser s'il y a soupçon de modification du script qui pourrait toucher la partie vérification d'intégrité. Cet exécutable est sur le dispositif, mais devrait être disponible à être téléchargé à partir d'internet.
Ensuite, après que la vérification d'intégrité est effectuée, le proxy va s'exécuter. S'il y a détection de champs de connexion, le programme demandera à l'utilisateur s'il veut sauvegarder ses données de connexion dans le dispositif à l'aide du stockage sécurisé, après avoir demandé s'il veut utiliser la connexion sécurisée (à la manière des gestionnaires de mots de passe dans les navigateurs). Si l'utilisateur accepte les deux demandes, le chargement de la page continuera comme décrit dans la première étape. Au moment de remplacement des données de connexion dans la requête que nous allons envoyer au serveur, après que l'utilisateur les a entrées dans le dispositif, nous faisons appel à la fonction dans le monde sécurisé pour sauvegarder les données. En parallèle, nous sauvegardons le nom de domaine dans une base de données non chiffrée dans le monde normal. Ainsi, pour chaque site visité, nous allons vérifier si les données de connexion existent pour ce site. Si c'est le cas, l'utilisateur n'aura qu'à mettre de données factices dans les champs de connexion de la page web visitée sur son ordinateur. Notre programme va récupérer les vraies données à partir du stockage sécurisé et les remplacer dans la requête. Il aurait été possible, comme dans les gestionnaires de mots de passe des navigateurs, de mettre les vraies données de connexion dans les champs de connexion. Ainsi, l'utilisateur n'a plus qu'à appuyer sur le bouton connexion. Mais ça posera un problème de sécurité, vu que nous avons supposé que nous n'avons pas confiance en l'ordinateur. Donc, nous éviterons que les données de connexions aillent sur l'ordinateur.

Conclusion

Dans ce chapitre, nous avons détaillé notre implémentation du prototype et du *secure computing*. Dans le chapitre suivant, nous discuterons des différents modèles d'attaque, ainsi que les limitations de notre solution.

CHAPITRE 5 ÉTUDE DE LA SÉCURITÉ DE NOTRE PROTOTYPE

Dans ce chapitre, nous allons discuter de la solution proposée au chapitre 4. Nous commencerons par présenter les attaques existantes et les défenses de notre système. Nous allons aussi décrire les limites de notre solution.

5.1 Niveaux de protection

Nous avons discuté tout au long de ce mémoire des attaques informatiques qui ont pour objectif, entre autres, de voler les données de connexion des utilisateurs. Nous allons commencer par les logiciels malveillants qui visent ou qui sont installés sur l'ordinateur de l'utilisateur.

5.1.1 Attaques visant l'ordinateur

L'absence de stockage des identifiants de connexion en tout temps sur l'ordinateur diminue énormément la surface d'attaque des logiciels malveillants. En effet, cet aspect permet d'éviter la menace présentée par les logiciels malveillants présents sur l'ordinateur qui ont pour objectif de voler les identifiants des utilisateurs. En particulier, ceci permet de contrer les attaques suivantes :

Les enregistreurs de frappes (*keyloggers*)

Ils peuvent être logiciels (qui se propagent et s'installent à travers les courriels indésirables, téléchargements furtifs, etc.), mais peuvent aussi être matériels (une pièce qui s'attache au bout du fil USB du clavier, qui enregistre toutes les frappes sur une petite mémoire interne). Ces attaques sont généralement les plus utilisées pour voler les identifiants de connexion des comptes bancaires en ligne. Dans notre système, l'utilisateur ne va pas risquer cette attaque, puisqu'il ne va pas taper son mot de passe. Au contraire, il va taper un mot de passe factice.

Hameçonnage

Les attaques de type hameçonnage (*phishing*) visent à tromper l'utilisateur et à l'inciter à se connecter sur un faux site qui ressemble au site authentique. Il y a plusieurs méthodes que les pirates utilisent pour faire du hameçonnage : le spam, principalement, mais aussi les redirections dans les publicités, l'empoisonnement des moteurs de recherche, l'achat de noms de domaine qui ressemblent au nom de domaine du site ciblé (*typo squatting*), etc.

Notre solution ne protégera pas complètement l'utilisateur de cette attaque, puisque si un utilisateur va sur un site de hameçonnage et se connecte, notre modèle va faire parvenir les données de connexions au serveur. Par contre, si l'utilisateur sauvegarde ses données de connexion pour un site (légitime) donné sur le Raspberry Pi 3, elles ne vont pas être utilisées pour un site de hameçonnage, car les noms de domaine ne concorderont pas. Ainsi, il pourrait se rendre compte qu'il est sur un site de hameçonnage (similairement à ce qui arrive avec un gestionnaire de mots de passe). Par ailleurs, cette attaque reste réalisable même sur une solution d'authentification à deux facteurs. Un attaquant peut prendre les identifiants d'un utilisateur via le site de hameçonnage, ensuite se connecter sur le site légitime et en même temps afficher un message à l'utilisateur (sur le site illégitime) d'entrer un code qu'il va recevoir par texto. Aussi, l'attaquant pourrait se connecter au site légitime. Par conséquent, l'utilisateur est toujours appelé à vérifier le site sur lequel il va se connecter à l'aide de l'icône de cadenas pour la connexion sécurisée, et il est tenu aussi de vérifier le certificat du site et son domaine. Cependant, notre solution facilite cette tâche pour l'usager lorsque les identifiants sont sauvegardés sur le dispositif à l'aide du stockage sécurisé.

Vol de cookies

Certains logiciels malveillants visent à voler les cookies des utilisateurs. Comme expliqué dans 4.4.1, ils servent entre autres à garder une session active pour un site donné. Le but de voler un cookie est, évidemment, d'ouvrir une session avec le compte de l'utilisateur à qui le cookie a été volé. Pour l'instant, dans notre prototype, les cookies sont gérés par le navigateur du client. C'est-à-dire que si un attaquant parvient à en voler, il pourrait se connecter. Mais l'effet de vol des cookies n'est pas énorme autant que l'utilisateur ferme la session. En fait, les sites avec des informations sensibles (les sites de banques en ligne par exemple) ne permettent pas de garder une session ouverte après une déconnexion ou une fermeture de l'instance du navigateur dans laquelle le site de la banque est ouvert. Donc, le seul moyen d'en tirer avantage est de voler le cookie et l'utiliser en même temps que l'utilisateur légitime utilise le service. Même pour les sites à utilisation générale, si un attaquant a réussi à ouvrir une session, il ne pourrait pas changer le mot de passe. En outre, plusieurs sites notifient l'utilisateur par rapport à l'ouverture d'une session avec des paramètres non usuels (localisation ou appareil utilisé par exemple). Malgré l'impact limité, ceci reste une limitation à adresser dans des travaux futurs.

Attaques par force brute

Certains attaquants vont essayer d'obtenir les données de connexion d'un utilisateur en faisant une attaque par force brute sur le site. Une attaque par dictionnaire ou force brute consiste à essayer plusieurs mots de passe jusqu'à trouver le bon. Dans le cas d'attaque par dictionnaire, l'attaquant essaie une liste de mots de passe les plus communs ou probables. Tandis que dans le cas de force brute, un attaquant essaie toutes les combinaisons possibles pour un nombre donné de caractères. Comme ces attaques ne vont pas passer par l'ordinateur (ni par le dispositif, bien sûr) alors l'utilisation du dispositif n'aura aucun effet. La protection contre ce type d'attaque doit se faire au niveau du serveur. Beaucoup de serveurs limitent les tentatives de connexion des utilisateurs. Par exemple, si un utilisateur entre son mot de passe de façon erronée cinq fois de suite, son compte sera verrouillé pendant une période de temps. Même si cette mesure pourrait empêcher des utilisateurs légitimes de se connecter parce qu'ils ont une difficulté à se rappeler ou à entrer leur mot de passe, elle est indispensable, sinon tous les comptes d'utilisateurs seront vulnérables aux attaques par force brute ou dictionnaire.

5.1.2 Attaques visant le dispositif

Un attaquant pourrait avoir comme cible notre dispositif utilisé afin de se connecter de manière sécuritaire. Étant donné que le Raspberry Pi 3 est destinée à une seule fin, les possibilités d'infection seront amoindries, contrairement à l'ordinateur qui est, généralement, utilisé pour accomplir plusieurs tâches (travail, loisirs, navigation sur internet, etc.), ce qui augmente le risque d'infection par un logiciel malveillant.

Attaques physiques

Un attaquant pourrait avoir un accès physique au Raspberry Pi 3. Cela donne naturellement un grand avantage à l'attaquant qui pourrait avoir plusieurs possibilités d'attaques. La plus simple est de faire un déni de service pour la connexion sécurisée : il peut facilement formater la carte micro SD ou supprimer des fichiers critiques. En conséquence, l'utilisateur ne peut plus se connecter à un service à travers le dispositif et il doit passer obligatoirement par une solution moins sécurisée. Un attaquant pourrait vouloir accéder aux identifiants de connexion de l'utilisateur. Dans ce cas, nous avons deux possibilités : soit l'utilisateur utilise le dispositif juste pour se connecter sans utiliser la fonctionnalité de connexion automatique avec les données de connexions sauvegardées au préalable, soit il utilise cette fonctionnalité, ce qui implique que les données de connexion seront stockées sur la carte micro SD. Dans le premier cas, l'attaquant peut conduire des attaques afin de voler les identifiants de connexion. Les

attaques logicielles seront décrites dans la section suivante. Pour le deuxième cas, même si les identifiants sont stockés sur le système de fichiers Linux, ils sont chiffrés avec le chiffrement AES. Donc, une attaque de force brute prendra assez de temps (milliers d'années pour une clé de taille 128 bits) pour la considérer irréalisable. Par contre, l'attaquant pourrait faire fonctionner le dispositif. Il y a un écran de connexion au lancement du Raspberry Pi 3 demandant un nom d'utilisateur et un mot de passe. Mais ce n'est pas compliqué de contourner cela en modifiant le fichier «*shadow*» qui contient les hachés des mots de passe des utilisateurs. En conséquence, l'attaquant pourrait se connecter aux domaines déjà enregistrés de l'utilisateur, puisque le remplacement des données se fait automatiquement. Selon ce scénario, l'attaquant aura accès à une session d'un compte de l'utilisateur. Pour pouvoir récupérer ces données, il faut implémenter des attaques logicielles.
Concernant les attaques à canal auxiliaire, elles sont réalisées dans le cas où l'accès physique ne donne pas d'avantage à l'attaquant. Elles demandent du matériel de laboratoire sophistiqué afin de les réaliser. Ainsi nous jugeons qu'elles sont hors de la portée du travail, étant donné que l'accès physique est suffisant.

Attaques logicielles

Un attaquant peut conduire des attaques logicielles avec ou sans un accès physique au dispositif. Même si avoir accès au Raspberry Pi 3 à distance n'est pas évident, ça reste possible dans quelques scénarios. En effet, puisque notre dispositif est connecté à internet d'une part et à un ordinateur que nous supposons infecté d'autre part, il sera exposé à des menaces. Par exemple, si un attaquant réussit à partir de l'ordinateur infecté à exploiter une potentielle vulnérabilité de dépassement de tampon dans le programme écrit ou dans l'une des librairies utilisées dans le dispositif en utilisant les requêtes HTTP par exemple. Une autre possibilité, l'infection à partir du serveur web. Cela reste peu probable, car si un attaquant réussit à compromettre un serveur, il aura accès aux bases de données qui contiennent les mots de passe ou leurs hachés, donc n'aura pas d'intérêt à attaquer le dispositif. Dans notre mise en place, seulement deux ports réseaux seront ouverts, le port 80 (HTTP) et le port 443 (HTTPS). Si un utilisateur change des paramètres facultatifs au fonctionnement du prototype, par exemple, lancer le service Secure Shell (SSH), cela ouvre un nouveau vecteur d'attaque.
Les attaques logicielles à partir du dispositif sont similaires à celles pouvant être réalisées dans un scénario classique où un ordinateur est utilisé pour se connecter puisque le système d'exploitation utilisé, même s'il a l'architecture ARM-v8 pour fonctionner sur un dispositif embarqué, reste très semblable à celui utilisé sur un ordinateur. En effet, un attaquant pourrait implémenter un enregistreur de frappes directement sur le Raspberry Pi 3. Ainsi, il

pourrait se procurer les vraies données de connexion pendant que l'utilisateur essaie de les entrer. Dès que l'attaquant a un accès privilégié au dispositif, quelques scénarios sont alors possibles, comme décharger la mémoire vive, écouter la communication entre les processus, etc.
Néanmoins, l'utilisation du *trusted computing* réduit la surface d'attaque. En effet, partant du fait qu'il y a séparation physique de la mémoire entre le monde normal et le monde sécurisé, même si un attaquant parvient à avoir un accès distant au dispositif, il ne pourrait pas changer les exécutions dans le monde sécurisé. Ainsi, dans un scénario où un utilisateur se ferait infecter son ordinateur sans *trusted computing*, l'exécution du code critique ne se fera pas en isolation du système d'exploitation auquel l'attaquant a accès. Au contraire, dans notre dispositif, il y a des tâches qui s'effectueront en isolation, par exemple la vérification d'intégrité. Cela signifie qu'une modification du fonctionnement du programme sera détectée à partir du monde sécurisé. Une autre fonctionnalité qui n'a pas pu être implémentée est l'entrée sécurisée des identifiants. Dans cette fonctionnalité, l'utilisateur tape ses identifiants directement dans la partie sécurisée de l'application. Une telle fonctionnalité éliminera la menace présentée par les enregistreurs de frappes présents sur le dispositif par exemple. Ceci est dû au fait qu'un logiciel malveillant s'exécutera dans le monde normal, mais l'entrée des identifiants se fait dans le monde sécurisé en isolation du monde normal et en conséquence en isolation du logiciel malveillant. Dans ce cas, un des moyens restant pour l'attaquant est d'intercepter la préparation et l'envoi de la requête de connexion à partir du dispositif puisque l'envoi de la requête ne se fait pas depuis le monde sécurisé. L'entrée sécurisée des données ainsi que le démarrage sécurisé seront discutés dans la section 5.3.

5.2 Difficultés d'implémentation

Un des indicateurs que nous allons utiliser afin d'évaluer la faisabilité et la viabilité de développement d'applications utilisant le *trusted computing* est les difficultés d'implémentation rencontrées.

Manque de documentation pour le système d'exploitation sécurisé

Pendant le processus de développement, nous avons remarqué le manque de documentation et de ressources pour le TEE OP-TEE. En fait, le seul moyen d'avoir une réponse à nos questions était de les poser sur leur répertoire GitHub. Cependant, nous n'avons pas réussi à obtenir de réponses satisfaisantes pour toutes nos questions, ce qui a ralenti le développement.

Installation du pilote Ethernet

Une des difficultés pour l'implémentation et le fonctionnement du prototype est la mise en place d'une seconde interface Ethernet. Nous utilisons l'interface native du Raspberry Pi 3 et en plus, pour la deuxième interface, nous utilisons un adaptateur USB vers Ethernet. Le premier problème est que le pilote pour l'adaptateur doit être chargé par le noyau Linux. De plus, la version du noyau Linux qui compile le pilote doit correspondre exactement avec la version du noyau qui le charge. Le système d'exploitation mis à disposition par les développeurs de OP-TEE ne contient pas les modules système nécessaires pour compiler le code du pilote. De plus, les versions du noyau pour le système d'exploitation sont très particulières ce qui nous empêche de faire de la compilation croisée. En conséquence, nous avons été obligés de recompiler le noyau Linux utilisé par les développeurs de OP-TEE pour générer les modules et ainsi pouvoir compiler le pilote de l'adaptateur directement sur le Raspberry Pi 3, puis le charger avec la commande «modprobe».

En conclusion, même si le développement de l'application utilisant le *trusted computing* a présenté quelques défis relatifs au fait que l'utilisation de ce concept n'est pas très démocratisée, ils sont surmontables. En plus, plus son utilisation s'élargira, plus il y aura de facilités à trouver des solutions aux problèmes à travers les communautés d'utilisateurs sur internet.

5.3 Limites de la solution

Comme nous l'avons vu plus haut dans ce mémoire, il n'existe pas de solution qui cocherait toutes les cases de sécurité, de déploiement et de convivialité. Néanmoins, nous faisons des efforts pour offrir le meilleur compromis. L'autre point à signaler est que toute solution proposée dépend forcément du cas d'utilisation. Un utilisateur peut combiner plusieurs méthodes d'authentification pour différents cas d'utilisation. Par exemple, il pourrait utiliser un gestionnaire de mots de passe sur son ordinateur à la maison dont il est sûr qu'il n'est pas infecté. En parallèle, il pourrait également utiliser notre solution lorsqu'il est forcé d'utiliser un autre ordinateur dans lequel il n'a pas confiance.
Tout en sachant que toute solution n'est pas totalement à l'abri d'éventuels points faibles, celle que nous proposons est également vulnérable sur certains points. Nous essaierons de développer ses limites dans les passages qui suivent.

Absence de démarrage sécurisé

Le démarrage sécurisé, comme expliqué en détail au chapitre 2, consiste à effectuer une chaîne de vérification qui commence au lancement du premier micrologiciel jusqu'à la vérification du système d'exploitation. Dans notre cas, cette vérification n'est pas faite. Son implémentation aurait donné plus de garanties quant à l'exécution du programme et la manipulation des données de l'utilisateur. En fait, comme le dispositif a un seul rôle à effectuer, la vérification d'intégrité du système à chaque démarrage sera suffisante pour être sûr de l'absence de logiciels malveillants sur le dispositif. L'intégrité du programme pourrait aussi être vérifiée à la fin de la chaîne de vérification.
Actuellement, ce qui empêche l'implémentation d'un tel concept est le fait que tout le code du démarrage du Raspberry Pi 3 est fermé et non modifiable. Mais le code utilisé pour notre application n'est pas dépendant du Raspberry Pi 3, juste de OP-TEE. C'est-à-dire que si une carte peut implémenter le démarrage sécurisé (en utilisant OP-TEE), il n'y aura pas de gros changements à faire pour exécuter l'application. Ainsi, nous aurons une solution mieux sécurisée.

Absence d'entrée de données sécurisée

Comme autre point faible de notre solution, nous pouvons noter l'absence d'une manière d'entrer les données de connexion de l'utilisateur d'une manière sécurisée : l'entrée des données se fait directement dans le monde sécurisé. Actuellement, dans notre prototype, l'entrée des données de connexions se fait dans l'application s'exécutant dans le monde normal. GlobalPlatform a défini les spécifications d'entrée sécurisée des données, mais ce n'est pas implémenté dans OP-TEE. Ces spécifications définissent la gestion des dispositifs d'entrée/sortie entre le monde normal et le monde sécurisé. Donc le clavier, qui représente le moyen d'entrer les données de l'utilisateur, sera virtuellement déconnecté du monde normal et connecté au monde sécurisé chaque fois que les méthodes d'entrée de données sécurisées seront appelées. Ce mécanisme renforcera la sécurité du modèle vu que les données seront entrées directement dans le monde sécurisé, ce qui évitera par exemple qu'un enregistreur de frappes enregistre les identifiants de l'utilisateur.

Absence d'une interface graphique

Bien que notre solution soit fonctionnelle, comme décrit au chapitre 4, l'interaction avec l'utilisateur se fait avec une interface en ligne de commande. Comme la communication est simple dans ce cas, cette méthode convient pour la plupart des utilisateurs. Mais pour une

adoption à grande échelle, une interface graphique simplifiée aidera à rendre notre solution plus facile à utiliser pour tous les usagers.

Utilisation du Wifi

Dans cette version du prototype, nous avons opté pour deux interfaces Ethernet pour la simplicité. Mais l'utilisation du Wifi au lieu d'une interface Ethernet (celle qui relie l'ordinateur et le Raspberry Pi 3) présentera des avantages : plus de mobilité puisque la communication entre le dispositif et l'ordinateur sera sans fil et encore, la possibilité d'utiliser la solution avec un téléphone cellulaire. L'utilisation du Wifi entre l'ordinateur ou le cellulaire et le dispositif n'affectera pas la sécurité puisque même si un attaquant arrive à intercepter et à déchiffrer les communications, les identifiants transmis sont factices et les vrais identifiants seront entrés dans le dispositif.

Conclusion

Dans ce chapitre, nous avons présenté des scénarios d'attaques possibles et leurs conséquences. Nous pouvons conclure que notre prototype nous protège de la majorité des attaques visant les ordinateurs comme les enregistreurs de frappes. Pour les attaques visant le dispositif, le stockage sécurisé permet de contrer certaines attaques physiques. L'exécution isolée diminue la surface d'attaque des attaques logicielles. Ceci démontre les avantages offerts par le *trusted computing*. Nous avons aussi donné les limites de notre prototype avec l'implémentation actuelle. Leur correction augmentera significativement la sécurité et la convivialité du prototype.

CHAPITRE 6 CONCLUSION

Dans ce mémoire, nous nous sommes donné comme objectifs d'étudier et d'implémenter le *trusted computing* et de l'utiliser dans une solution pour le transfert de données de connexion d'une manière sécuritaire. Pour les réaliser, nous nous sommes posé les questions de recherche suivantes :

Question 1 :
Quelles sont les différentes approches pour fournir du *trusted computing* sur du matériel dédié ?

Question 2 :
Parmi ces approches, laquelle offre le meilleur compromis entre sécurité, coût, rétrocompatibilité et facilité de développement dans le domaine de l'authentification ?

Question 3 :
L'approche choisie du *trusted computing* est-elle avantageuse par rapport aux méthodes d'authentification actuelles selon ces mêmes critères ?

Question 4 :
Peut-on en pratique concevoir un prototype d'authentification utilisant cette approche du *trusted computing* qui soit rétrocompatible, facile à développer et à coût réduit ?

Question 5 :
Quel est le risque résiduel de compromission d'une telle solution d'authentification basée sur le *trusted computing* ?

Nous allons synthétiser nos travaux en montrant comment nous avons répondu aux questions dans la section 6.1. Enfin, nous allons proposer dans la section 6.2 des solutions pour améliorer le système dans le futur et suggérer des pistes de recherche afin de mieux exploiter le *trusted computing* dans d'autres problématiques de sécurité.

6.1 Synthèse des travaux

Question 1

Quelles sont les différentes approches pour fournir du *trusted computing* sur du matériel dédié ?

Le but du *trusted computing* est principalement d'avoir une exécution isolée des processus

et données critiques. Ainsi, nous sommes assurés de la sécurité des données. Nous sommes également assurés qu'il n'y a pas eu de modification de la logique d'exécution, à condition que les mécanismes logiciel et matériel réalisant le *trusted computing* ne contiennent pas de bogues ou de défauts d'implémentation. Nous avons commencé par décrire ses origines et ses implémentations dans la recherche dans le chapitre 2. Ensuite, nous avons présenté les différentes alternatives possibles dans l'industrie afin de tirer profit de ses mécanismes de sécurité. Ces techniques sont principalement les modules de sécurité externes, les modules de sécurité internes, la virtualisation et enfin les extensions de sécurité aux processeurs.

Question 2

Parmi ces approches, laquelle offre le meilleur compromis entre sécurité, coût, rétrocompatibilité et facilité de développement dans le domaine de l'authentification ?

Après avoir exploré les différentes techniques de *trusted computing*, nous avons décidé laquelle nous allons utiliser. Ce choix était basé sur nos objectifs de conception. Nous voulions trouver le meilleur compromis entre sécurité, coût, rétrocompatibilité et facilité de développement. Les modules de sécurité externes sont chers et difficiles à mettre en place pour les considérer dans notre cas d'utilisation. De même, les modules de sécurité internes semblent présenter une difficulté à l'installation et au développement. En fait, les modules de sécurité internes, en particulier les TPM, reposent principalement sur des mécanismes d'isolation matériels afin de fournir du *trusted computing*. Du point de vue de la sécurité, théoriquement, les modules de sécurité internes et les extensions de sécurité du processeur fournissent les mêmes garanties quant à l'exécution isolée. Mais, le fait que les extensions de sécurité contiennent plusieurs mécanismes logiciels augmente les probabilités d'avoir des failles de sécurité. La taille du système d'exploitation sécurisé étant très réduite, cela nous encourage à faire confiance dans la sécurité des extensions de sécurité du processeur. Un autre point important concerne le fait que les extensions de sécurité du processeur offrent plus de flexibilité quant aux tâches pouvant être exécutées en isolation. En effet, les modules de sécurité internes fournissent principalement des fonctionnalités cryptographiques. En contraste, les extensions de sécurité permettent de développer une plus grande variété d'applications dépendant uniquement de l'API du monde sécurisé.

Finalement, pour le choix entre la virtualisation et les extensions de sécurité, nous avons penché pour la solution qui offre une sécurité améliorée sans être trop difficile à installer. Un de nos objectifs est de garantir un grand degré de sécurité, ce que la virtualisation ne peut pas l'offrir. En conséquence, nous avons choisi les extensions de sécurité du processeur

à l'aide de la technologie Trustzone de ARM.

Question 3

L'approche choisie du *trusted computing* est-elle avantageuse par rapport aux méthodes d'authentification actuelles selon ces mêmes critères ?

Dans le chapitre 3, nous avons discuté des avantages et des inconvénients des principales méthodes d'authentification. Nous avons montré que l'utilisation du *trusted computing* dans une solution d'authentification avec un dispositif dédié permettrait d'avoir plusieurs avantages. Le fait d'avoir un dispositif dédié, indépendant d'un ordinateur que nous supposons infecté, permet de contrer plusieurs menaces présentes sur les ordinateurs actuellement, principalement les enregistreurs de frappes. Nous avons ensuite utilisé le *trusted computing* dans le but de sécuriser le dispositif. Nous avons utilisé principalement le stockage sécurisé afin de vérifier l'intégrité du programme utilisé ainsi que pour stocker les identifiants des utilisateurs. Du point de vue de la sécurité, cela permet la détection des modifications apportées à ce programme, et également la protection des données stockées face aux attaques physiques. Nous aurons ainsi une convivialité similaire à celle offerte par un gestionnaire de mots de passe, tout en ayant une meilleure sécurité. Des améliorations possibles comme le démarrage sécurisé ou l'entrée sécurisée des données rendront ce schéma encore plus sécuritaire. Enfin, concernant le coût, le prix étant aux alentours de 35 dollars canadiens, cela n'est pas cher par rapport aux autres accessoires informatiques, et est ainsi à la portée de la majorité de la population.

Question 4

Peut-on en pratique concevoir un prototype d'authentification utilisant cette approche du *trusted computing* qui soit rétrocompatible, facile à développer et à coût réduit ?

À ce propos, nous avons conçu notre solution afin de transmettre les données de connexion de l'utilisateur en toute sécurité en utilisant le concept du *trusted computing*. Nous avons détaillé l'implémentation et la conception dans le chapitre 4. Nous avons cherché en parallèle à minimiser au maximum les modifications nécessaires pour le client et le serveur. Cela est primordial pour espérer une adoption de la solution. En effet, il y a un seul changement à faire du côté du client : l'installation du certificat de chiffrement asymétrique afin de pouvoir gérer le trafic réseau HTTPS. Le fonctionnement du serveur reste donc inchangé. Le

développement du prototype, même s'il a présenté quelques défis, n'était pas insurmontable pour une personne possédant de bonnes compétences en développement. De plus, plus les personnes expérimenteront avec cette technologie, plus il sera facile de développer et de trouver des réponses aux questions. Concernant le coût, comme expliqué dans la section précédente, le prix du matériel est très abordable, même s'il y a des améliorations possibles en termes de convivialité qui nécessiteraient quelques dépenses supplémentaires. Une solution idéale serait d'avoir un dispositif produit juste pour ce cas d'utilisation. Une autre possibilité serait d'avoir un dispositif utilisant le *trusted computing* et regroupant toutes les applications sensibles de point de vue de la sécurité.

Question 5

Quel est le risque résiduel de compromission d'une telle solution d'authentification basée sur le *trusted computing* ?

Pour évaluer notre système, nous avons discuté des attaques possibles. Nous avons détaillé les types d'attaques dont nous sommes protégés grâce à notre système, et celles qui constitueraient toujours une éventuelle menace pour la sécurité des données de l'utilisateur. L'étude était théorique, sous forme de modèles de menace, en se basant sur les logiciels malveillants existants et leur fonctionnement. Nous avons démontré que notre solution résiste à la plupart des menaces présentes sur un ordinateur infecté. Concernant les menaces sur le dispositif, le *trusted computing* nous permet de réduire la surface des attaques possibles. Nous avons aussi décrit, dans les limites de ce travail, les implémentations à faire afin de mieux sécuriser le dispositif et les données qu'il protège.

6.2 Travaux futurs

Afin d'améliorer le travail et le rendre plus sécuritaire, des améliorations futures doivent se faire. En effet, dans la section5.3, nous avons exposé des limites du modèle conçu et implémenté.
Le premier objectif serait d'implémenter le démarrage sécurisé. Cela améliorera grandement la sécurité du dispositif et donnera à l'utilisateur la garantie qu'il n'y a pas eu de modification de logiciels. Pour y parvenir, le matériel choisi devra permettre de le faire. Ainsi, le seul vecteur d'attaque restant sera le déni de service si un attaquant décide de modifier le système d'exploitation ou le programme de connexion sécurisée. Un autre objectif serait d'ajouter d'autres vérifications. Par exemple, pour prévenir le hameçonnage, en vérifiant les adresses

des sites web visités et les protocoles utilisés, et au besoin avertir l'utilisateur.
Une autre direction de recherche sera d'exploiter le potentiel du *trusted computing* pour régler d'autres problèmes de sécurité. L'une des pistes pouvant être explorées est la sécurisation des objets connectés en utilisant ce concept. Une autre possibilité est de l'utiliser davantage dans les applications actuelles sur les ordinateurs, afin d'améliorer la sécurité et de protéger les données sensibles des utilisateurs, surtout pour les processus critiques.

RÉFÉRENCES

"Xen project software overview", https://wiki.xen.org/wiki/Xen_Project_Software_Overview, en ligne ; accédé 19-Septembre-2017.

F. Aloul, S. Zahidi, et W. El-Hajj, "Two factor authentication using mobile phones", dans *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on.* IEEE, 2009, pp. 641–644.

I. Anati, S. Gueron, S. Johnson, et V. Scarlata, "Innovative technology for cpu based attestation and sealing", dans *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, 2013.

Apple, "iOS Security", https://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2017, en ligne ; accédé 15-Novembre-2017.

W. Arbaugh, D. Farber, et J. Smith, "A secure and reliable bootstrap architecture", dans *Proceedings of the 1997 IEEE Symposium on Security and Privacy.* IEEE, 1997, pp. 65 – 71.

ARM, "ARM TRUSTZONE White paper", http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009, en ligne ; accédé 09-Novembre-2017.

N. Asokan, J.-E. Ekberg, K. Kostiainen, A. Rajan, C. Rozas, A.-R. Sadeghi, S. Schulz, et C. Wachsmann, "Mobile trusted computing", *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1189–1206, 2014.

A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, et W. Shen, "Hypervision across worlds : Real-time kernel protection from the arm trustzone secure world", dans *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2014, pp. 90–102.

S. Beaupre, "TRUSTNONE - Signed comparison on unsigned user input", http://theroot.ninja/disclosures/TRUSTNONE_1.0-11282015.pdf, 2015, en ligne ; accédé 09-Novembre-2017.

J. Bonneau, C. Herley, P. Van Oorschot, et F. Stajano, "Passwords and the evolution of imperfect authentication", *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, 2015.

J. Bonneau, C. Herley, P. C. Van Oorschot, et F. Stajano, “The quest to replace passwords : A framework for comparative evaluation of web authentication schemes”, dans *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE, 2012, pp. 553–567.

S. Chiasson, E. Stobert, A. Forget, R. Biddle, et P. C. Van Oorschot, “Persuasive cued click-points : Design, implementation, and evaluation of a knowledge-based authentication mechanism”, *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 2, pp. 222–235, 2012.

V. Costan, I. A. Lebedev, et S. Devadas, “Sanctum : Minimal hardware extensions for strong software isolation.” dans *USENIX Security Symposium*, 2016, pp. 857–874.

CVE-details, “XEN : Security Vulnerabilities”, `https://www.cvedetails.com/vulnerability-list/vendor_id-6276/XEN.html`, en ligne ; accédé 16-Novembre-2017.

D. Florêncio, C. Herley, et P. C. van Oorschot, “An administrator’s guide to internet password research”, dans *28th Large Installation System Administration Conference (LISA14).* Seattle, WA : USENIX Association, 2014, pp. 44–61. En ligne : `https://www.usenix.org/conference/lisa14/conference-program/presentation/florencio`

FORTEZZA, “FIPS-140-1 Security and FORTEZZA Crypto Cards”, `https://technet.microsoft.com/en-us/library/cc962054.aspx`, en ligne ; accédé 11-Janvier-2018.

GlobalPlatform, “GlobalPlatform”, `https://www.globalplatform.org/`, en ligne ; accédé 29-Janvier-2018.

Google Chromebook, “Chromebook TPM usage”, `https://www.chromium.org/developers/design-documents/tpm-usage`, en ligne ; accédé 09-Octobre-2017.

HiKey, “HiKey specifications”, `https://www.96boards.org/product/hikey/`, en ligne ; accédé 26-Janvier-2018.

Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, et H. Guan, “vtz : Virtualizing arm trustzone”, 2017.

Huawei, “Security Advisory - Two Privilege Escalation Vulnerabilities in Huawei Mate 7 Smartphones”, `http://www.huawei.com/en/psirt/security-advisories/hw-432799`, 2015, en ligne ; accédé 09-Novembre-2017.

Juno, “Juno Arm Development Platform specifications”, `https://developer.arm.com/products/system-design/development-boards/juno-development-board`, en ligne ; ac-

cédé 26-Janvier-2018.

Keccak, “Keccak”, https://keccak.team/keccak_specs_summary.html, en ligne ; accédé 5-Février-2018.

KSV-21, “KSV-21 Enhanced Crypto card”, https://www.raytheon.com/capabilities/rtnwcm/groups/public/documents/content/ksv_21.pdf, en ligne ; accédé 11-Janvier-2018.

H. Li et D. Evans, “Horcrux : A password manager for paranoids”, *arXiv preprint arXiv :1706.05085*, 2017.

R. Maes et I. Verbauwhede, “Physically unclonable functions : A study on the state of the art and future research directions”, dans *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 3–37.

D. Maltoni, D. Maio, A. K. Jain, et S. Prabhakar, *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

Mediatek MT8173, “Mediatek MT8173 specifications”, https://www.mediatek.com/products/tablets/mt8173, en ligne ; accédé 25-Janvier-2018.

Microsemi, “Microsemi FPGA security”, https://www.microsemi.com/products/fpga-soc/security#secure-hardware, en ligne ; accédé 11-Janvier-2018.

mitmproxy, “mitmproxy”, https://mitmproxy.org/, en ligne ; accédé 31-Janvier-2018.

OP-TEE, “OP-TEE”, https://www.op-tee.org/, en ligne ; accédé 29-Janvier-2018.

B. Parno, C. Kuo, et A. Perrig, “Phoolproof phishing prevention”, dans *Financial Cryptography*, vol. 4107. Springer, 2006, pp. 1–19.

H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon *et al.*, “ftpm : A software-only implementation of a tpm chip.” dans *USENIX Security Symposium*, 2016, pp. 841–856.

Raspberry Pi 3, “Raspberry Pi 3 specifications”, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, en ligne ; accédé 25-Janvier-2018.

R. Roemer, E. Buchanan, H. Shacham, et S. Savage, “Return-oriented programming : Systems, languages, and applications”, *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 1, p. 2, 2012.

Répertoire OP-TEE, “OP-TEE repo”, https://github.com/OP-TEE, en ligne ; accédé 29-Janvier-2018.

R. SecurId, “RSA SecurID overview”, https://www.rsa.com/content/dam/pdfs/5-2017/rsa-securid-suite-solution-brief-0417.pdf, en ligne ; accédé 27-Novembre-2017.

Sequitur Labs, “Sequitur Labs”, https://www.sequiturlabs.com/, en ligne ; accédé 29-Janvier-2018.

A. Seshadri, A. Perrig, L. Van Doorn, et P. Khosla, “Swatt : Software-based attestation for embedded devices”, dans *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 272–282.

S. Shinde, Z. L. Chua, V. Narayanan, et P. Saxena, “Preventing your faults from telling your secrets : Defenses against pigeonhole attacks”, *arXiv preprint arXiv :1506.04832*, 2015.

Signal, “Private contact discovery for Signal”, https://signal.org/blog/private-contact-discovery/, en ligne ; accédé 29-Novembre-2017.

R. Strackx, F. Piessens, et B. Preneel, “Efficient isolation of trusted subsystems in embedded systems”, *Security and Privacy in Communication Networks*, pp. 344–361, 2010.

G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, et S. Devadas, “Aegis : architecture for tamper-evident and tamper-resistant processing”, dans *Proceedings of the 17th annual international conference on Supercomputing*. ACM, 2003, pp. 160–171.

A. Tang, S. Sethumadhavan, et S. Stolfo, “Clkscrew : exposing the perils of security-oblivious energy management”, dans *26th USENIX Security Symposium*, 2017.

H. Tao, “An arrangement and method of graphical password authentication”, Oct. 5 2005, US Patent App. 11/163,115.

Trusted Computing Group, “Trusted Computing Group”, https://trustedcomputinggroup.org/, en ligne ; accédé 05-Décembre-2017.

Virtualbox, “Virtualbox”, https://www.virtualbox.org/, en ligne ; accédé 16-Novembre-2017.

N. Weichbrodt, A. Kurmus, P. Pietzuch, et R. Kapitza, “Asyncshock : Exploiting synchronisation bugs in intel sgx enclaves”, dans *European Symposium on Research in Computer Security*. Springer, 2016, pp. 440–457.

Y. Xu, W. Cui, et M. Peinado, “Controlled-channel attacks : Deterministic side channels for untrusted operating systems”, dans *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 640–656.

Y. Yarom et K. Falkner, “Flush+ reload : A high resolution, low noise, l3 cache side-channel attack.” dans *USENIX Security Symposium*, 2014, pp. 719–732.

S. Zhao, Q. Zhang, G. Hu, Y. Qin, et D. Feng, “Providing root of trust for arm trustzone using on-chip sram”, dans *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*. ACM, 2014, pp. 25–36.