INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Conception et implémentation d'un mécanisme d'authentification par carte à puce pour le chiffrement de disques durs par le logiciel VeraCrypt

Rapport de pré-étude et spécification fonctionnelle

Mathis Caisson, Andrei Cocan, Guilhem Grac, Dorian Humeau, François Le Roux, Mathis Mauvisseau, Brice Namy
Encadrants : *Gildas Avoine, Jules Dupas*

Département informatique
Projet de 4ème année, 2022-2023

Table des matières

Introduction **2**

1 Cahier des charges **3**

2 Fonctionnement de VeraCrypt **4**
2.1 Structure d'un volume . . . 4
2.2 Authentification à double facteur . . . 5
2.3 Montage d'un volume . . . 5
2.4 Modification du mot de passe . . . 6
2.5 Volumes cachés . . . 7

3 Cryptographie dans VeraCrypt **8**
3.1 Algorithmes . . . 8
3.2 Random Number Generator . . . 8
3.3 Application des keyfiles . . . 9
3.4 Dérivation des clés . . . 9
3.5 Mode de chiffrement . . . 10

4 Exploitation des cartes EMV pour VeraCrypt **11**
4.1 Alternative EMV . . . 11
4.2 Standard EMV . . . 11
4.3 Déploiement . . . 11
4.4 Commandes . . . 12
4.5 Sélection d'applications . . . 12
4.6 Lecture des données . . . 13
4.7 Interfacage . . . 13
4.8 Données utilisables . . . 14

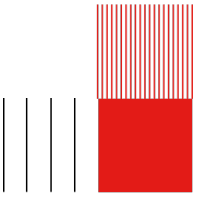
5 Code de VeraCrypt **16**
5.1 Architecture logicielle . . . 16
5.2 Attribution des chemins . . . 16
5.3 Application des keyfiles . . . 16
5.4 Adaptation aux cartes EMV . . . 17
5.5 Maquette de l'interface graphique . . . 18

6 Proofs Of Concept **19**
6.1 Premier POC : Extraction des données EMV . . . 19
6.2 Deuxième POC : Intégration basique dans VeraCrypt . . . 20

7 Gestion de projet **21**
7.1 Réunions et outils . . . 21
7.2 Répartition des rôles et tâches . . . 21
7.3 Entrevue avec Mounir Idrassi . . . 22
7.4 Analyse et Gestion du risque . . . 23
7.5 Faisabilité du projet . . . 23
7.6 Avancement du projet . . . 24

Conclusion **25**

Bibliographie **26**



Introduction

Objectif Ce projet s'articule autour d'un logiciel dont le code est open source : VeraCrypt. Cet outil est utilisé pour le chiffrement de disques ou de volumes. L'objectif est d'ajouter à ce logiciel la possibilité d'utiliser une carte EMV plutôt qu'une carte PKCS#11 pour renforcer la sécurité du mot de passe ainsi que la propriété de déni plausible.

Contexte VeraCrypt succède au logiciel TrueCrypt et est développé depuis fin 2014 par IDRIX, une société de conseil en systèmes et logiciels informatiques. Mounir Idrassi, seul et unique membre, en assure la gérance et est le principal développeur de VeraCrypt. Des ingénieurs volontaires contribuent aussi au développement du logiciel. La version publique la plus récente (1.25.9) a été publiée en février 2022. Ce logiciel est mis à jour deux à trois fois par an en moyenne. Disponible sous Windows, Linux et macOS, VeraCrypt permet à ses utilisateurs de créer des volumes de stockage virtuels chiffrés [1] dissimulables voire même de chiffrer directement des partitions du système d'exploitation sur lequel il est installé.

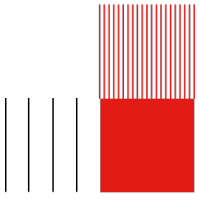
Veracrypt est utilisé dans le monde entier, par des milliers de personnes et sur des systèmes d'exploitations divers. Ce logiciel est principalement téléchargé sur deux hébergeurs d'applications : `Launchpad` [17] et `Source Forge` [18]. Le site officiel de VeraCrypt utilise l'hébergeur Launchpad ne fournissant aucune statistique de téléchargements, rendant difficile toute étude. En revanche, l'hébergeur Source Forge fournit des statistiques et des commentaires sur les différents téléchargements du logiciel. Veracrypt y est téléchargé plusieurs centaines de fois par jour pour un total de plus de 1,2 million de téléchargements depuis juin 2013. 72% de ces téléchargements sont effectués depuis la plateforme Windows, le reste étant partagé entre Linux, macOS et d'autres systèmes d'exploitation. D'après Source Forge, les États-Unis sont le pays où VeraCrypt est le plus téléchargé, représentant environ 23% des téléchargements dans le monde. Selon les statistiques, Veracrypt est majoritairement téléchargé depuis des pays du nord du globe, tels que les États-Unis, la France, l'Allemagne, la Pologne ou encore la Russie. Les commentaires laissés par les utilisateurs ayant téléchargé ce logiciel présentent des profils variés, dont l'expérience et les compétences sont plus ou moins complètes. Un utilisateur de Veracrypt peut être un employé d'une PME, un journaliste sur le terrain tout comme un particulier sur son ordinateur personnel.

Le public visé est notamment composé de dissidents politiques, de lanceurs d'alertes et de reporters en milieu hostile dont la sécurité des informations est essentielle. L'association Reporters Sans Frontières recommande cet outil de chiffrement de disques dans son Guide de Protection des Sources de 2021 [24]. VeraCrypt est intégré au SILL [2] [23] depuis le 1er janvier 2018, la version minimale préconisée étant la 1.24.

La sécurité du code source de VeraCrypt (v1.18) a été auditée par deux ingénieurs de Quarkslab [3] entre le 16 août et le 14 septembre 2016 grâce au financement de l'OSTIF [4]. Le rapport [22] montrait une vulnérabilité critique au niveau de la cryptographie, corrigée depuis la version 1.19 de VeraCrypt. Plus récemment, un autre audit a été mené par le Fraunhofer Institute for Secure Information Technology (SIT) au nom du Federal Office for Information Security (BSI) d'Allemagne. Le rapport [21], publié en anglais le 10 décembre 2020, est disponible sur le site de la BSI. De cet audit a été conclu que VeraCrypt (v1.23) ne présente pas de failles de sécurité substantielles et assure bien la confidentialité des données chiffrées. Cependant, l'authentification et l'intégrité ne sont pas assurées dans certains cas.

Les grandes étapes de ce projet seront tout d'abord d'étudier le standard EMV afin de pouvoir manipuler les cartes et d'analyser le code source de VeraCrypt pour déterminer quelles modifications apporter, puis de démontrer la faisabilité technique du projet et enfin de développer une version modifiée de VeraCrypt fonctionnelle et incorporable au logiciel officiel.

1. Dénommés tout simplement «volumes» dans ce rapport.
2. Le Socle Interministériel de Logiciels Libres est un ensemble de logiciels libres de droits préconisés par l'État français dans le cadre de la modernisation globale de ses systèmes d'informations, dont le Premier Ministre est chargé de la gestion.
3. Entreprise française spécialisée en cybersécurité et plus spécifiquement dans la protection et l'analyse de logiciels.
4. Open Source Technology Improvement Fund, association apportant une aide financière et logistique aux projets open-source en lien avec la sécurité.



1 Cahier des charges

Objectifs du projet

L'objectif du projet est de pouvoir utiliser VeraCrypt avec des cartes EMV. Les fonctionnalités ajoutées consisteront à extraire d'une carte à puce de type EMV des informations propres à celle-ci, éventuellement les traiter, et les utiliser en tant que keyfiles (voir section 2.2). Ce principe reprend exactement celui existant pour les cartes PKCS#11. Pour décrire un cas d'usage, un utilisateur pourra connecter sa carte EMV à un lecteur branché à son ordinateur et ainsi l'utiliser en tant que source de keyfile. Au moment de créer un volume, il pourra cocher l'option «Utiliser keyfile» et sélectionner la carte EMV connectée. Pour déchiffrer ce volume, il devra une nouvelle fois saisir le mot de passe, brancher sa carte EMV et préciser qu'il l'utilise pour déchiffrer le volume. L'une des données suggérées par le client est l'ICC Public Key Certificate, un certificat propre à la carte pouvant être utilisé comme keyfile.

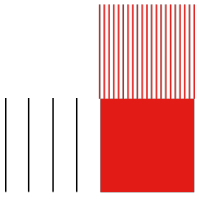
Spécifications techniques

- Récupérer le projet Github VeraCryt en le copiant ou en effectuant un fork, afin d'ajouter en définitive les modifications au sein de l'application existante.
- Le code sera développé en C/C++ et devra être multiplateforme au même titre que le projet existant. Il devra également être suffisamment documenté pour permettre la relecture par des membres extérieurs à l'équipe. Une documentation utilisateur extérieure au code doit de surcroît être fournie.
- La partie portant sur la manipulation de cartes EMV (extraction de l'ICC Public Key Certificate) a préalablement été grandement développée par M. Dupas et fournie sous la forme de scripts Python et C, le client laissant le choix du traitement de ce certificat et même la possibilité de sélectionner d'autres éléments à extraire de la carte si le choix est argumenté.
- Les licences de librairies tierces utilisées devront être compatibles avec leur utilisation au sein de VeraCrypt et leurs fichiers devront être ajoutés.
- L'intégralité des tests effectués sera à présenter avec leurs détails techniques (processus de test, OS de la machine, modèle, pays, année d'émission des cartes utilisées).
- Le fonctionnement avec les cartes EMV doit être similaire à celui des cartes PKCS#11 depuis le point de vue des utilisateurs.
- Toutes les cartes EMV émises après 2014 doivent être acceptées si elles contiennent un Icc Public Key Certificate. De plus, trois applications EMV doivent être supportées pour le moment : Visa, Mastercard et American Express.

Extensions possibles

M. Dupas, jouant le rôle du client, propose deux extensions possibles à réaliser en fin de projet si les délais le permettent :

- Développer la même fonctionnalité avec le «sans contact» des cartes EMV
- Réfléchir à la possibilité d'un keyfile stocké sur un téléphone (en utilisant NFC, Bluetooth) dans le but de toucher un potentiel de personnes encore plus important.



2 Fonctionnement de VeraCrypt

VeraCrypt est un outil de chiffrement de disques durs permettant de créer et de manipuler des volumes chiffrés. Ce logiciel inclut parmi ses fonctionnalités les suivantes : créer un volume, y ajouter des fichiers, modifier ou effacer ces fichiers, changer le mot de passe propriétaire du volume. Les volumes VeraCrypt ne sont pas de simples volumes chiffrés : ils ne peuvent être accédés que lorsqu'ils sont « montés » dans le système de fichiers en utilisant VeraCrypt (voir section 2.3).

2.1 Structure d'un volume

Techniquement, un volume VeraCrypt est un fichier dont la taille est fixée à la création et constante tout au long de la vie du volume, quel que soit le nombre de fichiers présents. Évidemment, lorsque la taille limite est atteinte, il n'est plus possible d'ajouter de fichiers. Toute donnée chiffrée dans le volume est indistinguable d'une donnée générée aléatoirement.

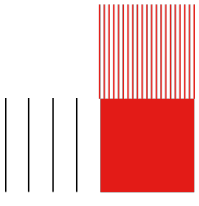
Table 1 – Structure d'un volume VeraCrypt [19]

Décalage (octets)	Taille (octets)	Description
0	512	Entête
512	65024	Réservé
65536	65536	Entête du volume caché
131072	x	Zone des données
131072 + x	65536	Sauvegarde de l'entête
196608 + x	65536	Sauvegarde de l'entête du volume caché

Au début d'un volume VeraCrypt se trouve l'entête, nécessaire au déchiffrement des données. Suit une zone réservée où sont stockées les informations utilisées par le bootloader en cas de chiffrement d'une partition du système d'exploitation. 65536 octets sont dédiés au stockage de l'entête du volume caché (voir section 2.5). Dans le cas où l'entête est endommagé, il est alors impossible de monter le volume VeraCrypt dans le système de fichiers. Pour y remédier, une sauvegarde est stockée à la fin du volume. Ce pied de page est une copie du contenu de l'entête, chiffrée avec une autre paire de clés, dérivées à l'aide d'un autre sel (voir section 3.4). De même, l'entête du volume caché est sauvegardée dans le dernier espace du volume. Au milieu de ces zones se trouve celle de stockage des données chiffrées.

Table 2 – Structure d'une entête VeraCrypt [19]

Décalage (octets)	Taille (octets)	Description
0	64	Sel
64	4	Chaîne ASCII «VERA»
68	2	Version du format de l'entête
70	2	Version minimale du programme pour ouvrir le volume
72	4	Somme de contrôle CRC-32 des octets 256 à 511 déchiffrés
76	16	Réservé
92	8	Taille du volume caché en octets
100	8	Taille du volume en octets
108	8	Décalage en octets du début de la zone des données
116	8	Volume des données chiffrées en octets
124	4	Bits de drapeaux / indicateurs
128	4	Taille du secteur en octets
132	120	Réservé
252	4	Somme de contrôle CRC-32 des octets 64 à 251 déchiffrés
256	256	Clés concaténées chiffrant la zone des données



L'entête contient les informations nécessaires au chiffrement et déchiffrement des données. La taille (512 octets) et le contenu de l'entête sont constants de la création à la destruction du volume. L'entête contient tout d'abord le sel en clair sur 64 octets, puis la chaîne de caractères «VERA» en ASCII sur 4 octets. La version du format de l'entête permet à VeraCrypt de savoir quels champs récupérer et où les trouver, l'actuelle étant la deuxième. Les sommes de contrôle calculées par la fonction CRC-32 sont stockées sur 4 octets chacune. Elles permettent de vérifier l'intégrité des clés pour la première et des éléments de l'entête pour la seconde. L'entête contient à mimina deux clés (dans le cadre du mode de chiffrement XTS [5] - voir section 5) dans ses 256 derniers octets : la clé primaire et la clé secondaire, utilisées pour chiffrer et déchiffrer les données [6]. Si le volume ne contient pas de volume caché (voir section 2.5), l'espace de l'entête dédié à la taille du volume caché est rempli de zéros. La quasi-totalité de l'entête est conservée chiffrée dans la mémoire de la machine de l'utilisateur. Seul le sel, nécessaire au déchiffrement de l'entête qui le contient, y est stocké en clair.

2.2 Authentification à double facteur

Actuellement, la sécurité du chiffrement de VeraCrypt est basée sur l'utilisation d'un mot de passe connu uniquement du créateur du volume. Ce dernier a également la possibilité d'utiliser un ou plusieurs fichiers additionnels afin de renforcer l'entropie du mot de passe : les keyfiles. Ces fichiers n'ont pas l'obligation d'être spécialement créés dans cet objectif, n'importe quel type de fichier peut être choisi par l'utilisateur. VeraCrypt propose d'en générer aléatoirement. Ces keyfiles peuvent être stockés n'importe où, le disque où le volume associé est localisé étant l'endroit le plus récurrent. Il est possible d'utiliser une mémoire amovible, une clé USB par exemple, comme espace de stockage de keyfile. Cependant, VeraCrypt permet aussi à ses utilisateurs de stocker leurs keyfiles sur une carte à puce de type PKCS#11 [7]. Ce type de carte est dédié à des applications sécurisées, la protection des données sur la carte étant assurée par un code PIN [8].

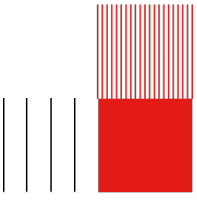
Quels que soient les fichiers utilisés comme keyfiles ou l'endroit où ils sont stockés, ils sont ensuite combinés avec le mot de passe (voir section 3.3) puis dérivés à l'aide d'un sel (voir section 3.4) afin de générer les clés cryptographiques protégeant les données. L'utilisation de keyfiles en plus du mot de passe est assimilable à une authentification à double facteur de l'utilisateur voulant accéder au volume.

2.3 Montage d'un volume

Afin d'accéder à son volume, l'utilisateur doit d'abord le monter dans le système de fichiers [11] (Figure 1). Plusieurs étapes de déchiffrement ont alors lieu, et cela uniquement à la volée dans la RAM. Aucune donnée n'est jamais stockée en clair sur le disque de la machine de l'utilisateur, elles sont cependant forcément présentes à un moment en clair dans le cache du processeur. Tout d'abord, 512 octets du volume sont chargés dans la RAM, correspondant à l'entièreté de l'entête chiffré. Pour le déchiffrer, l'utilisateur fournit donc à VeraCrypt les secrets que sont le mot de passe et les keyfiles, qui sont directement combinés par VeraCrypt (voir section 3.3). Cette combinaison et le sel (extrait de l'entête) sont dérivés à l'aide de la fonction de dérivation de clés PBKDF2 [9] (voir section 3.4), utilisant l'algorithme de hachage. Cette dernière génère les deux clés permettant de déchiffrer l'entête avec l'algorithme de chiffrement.

Cependant, les deux algorithmes (choisis par l'utilisateur à la création du volume) ne sont ni retenus par VeraCrypt, ni stockés dans l'entête. Garder secrets les algorithmes utilisés impose à quiconque souhaite déchiffrer l'entête d'effectuer une recherche exhaustive sur toutes les combinaisons d'algorithmes possibles. La sécurité contre une attaque par recherche exhaustive du mot de passe en est considérablement renforcée. Effectivement, elle se transforme en une double recherche exhaustive et est exponentiellement rallongée.

5. Mode de chiffrement par blocs basé sur XEX utilisé pour le chiffrement complet des disques.
6. Le chiffrement est à clés symmétriques : les mêmes clés sont utilisées pour le chiffrement et le déchiffrement.
7. Public-Key Cryptography Standard #11 est une API définissant une interface générique pour les périphériques cryptographiques.
8. Personal Identification Number
9. Password-Based Key Derivation Function 2 est une fonction de dérivation de clé, appartenant à la famille des normes Public Key Cryptographic Standards, plus précisément PKCS #5 v2.0.



Dans le cadre d'une utilisation normale, VeraCrypt dispose directement des secrets fournis par l'utilisateur. De ce fait, sa recherche exhaustive est plutôt assimilable à une détection automatique des algorithmes. Les combinaisons possibles sont testées une à une en générant une paire de clés et en essayant de déchiffrer l'entête avec. Deux vérifications sont alors réalisées sur les octets obtenus par la tentative de déchiffrement. La première est que ces octets contiennent les caractères «VERA». La seconde porte sur les valeurs des sommes de contrôle CRC-32 présentes dans l'entête, comparées avec les valeurs calculées par VeraCrypt sur le clair obtenu. Une fois ces deux vérifications validées, alors l'intégrité de l'entête est assurée et VeraCrypt considère correcte la combinaison des algorithmes de chiffrement et de hachage utilisée. Tant que les deux vérifications ne sont pas validées, VeraCrypt teste une nouvelle combinaison d'algorithmes.

VeraCrypt propose aussi à l'utilisateur de renseigner l'algorithme de hachage, ce qui accélère la recherche exhaustive. L'entête une fois déchiffré, VeraCrypt peut en extraire les clés permettant à leur tour de déchiffrer les données du volume à la volée avec le mode XTS (voir section 3.5) utilisant l'algorithme de chiffrement.

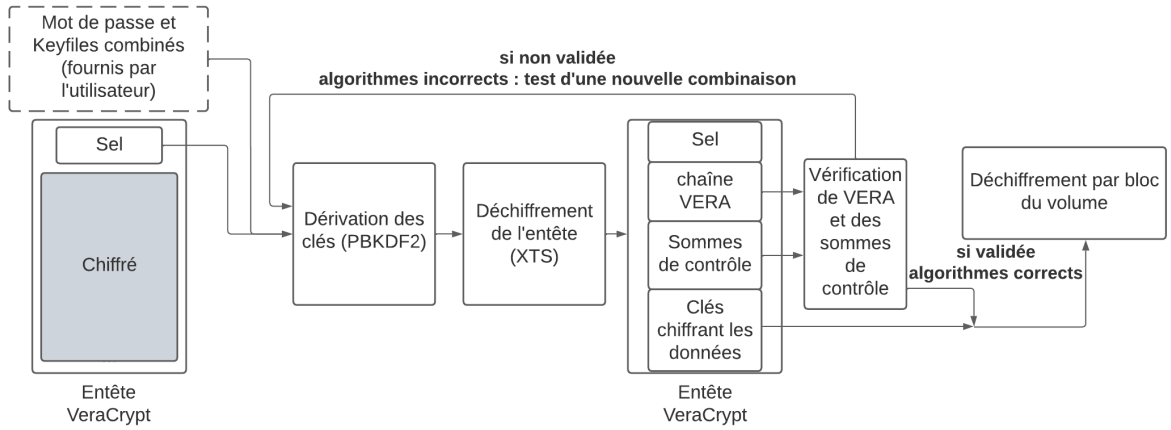


Figure 1 – Montage d'un volume

2.4 Modification du mot de passe

Le contenu de l'entête est constant tout au long de la vie du volume VeraCrypt. Par conséquent, le chiffré de l'entête reste le même tant que le propriétaire du volume ne change pas son mot de passe. Dans le cas où l'utilisateur choisit de modifier son mot de passe [9], de nouvelles clés sont dérivées à partir du nouveau mot de passe et l'entête est entièrement rechiffré avec. Le changement du mot de passe n'ayant pas d'impact sur l'intégrité des clés chiffrant les données du volume, ces dernières ne nécessitent pas de rechiffrement et restent intactes. Cependant, cela impacte fortement la sécurité du volume. En effet, un attaquant possédant l'ancien l'entête chiffré peut remplacer l'entête chiffré actuel par l'ancien et monter le volume en utilisant l'ancien mot de passe. Créer un nouveau volume chiffré avec le nouveau mot de passe et y transférer les fichiers est une meilleure alternative pour pallier à cette faille de sécurité.

2.5 Volumes cachés

Dans le but de renforcer le déni plausible, VeraCrypt a ajouté la possibilité de cacher un volume à sa création. Le principe est le suivant : un volume est créé à l'intérieur d'un autre volume standard (Figure 2) dit «externe». Ce volume caché est réparti chiffré dans l'espace libre du volume externe. Le chiffrement le fait s'apparenter à des données aléatoires. Lorsqu'un volume standard ne contient pas de volume caché, ses zones réservées pour l'entête et les données d'un éventuel volume caché sont remplies avec de la mémoire libre correspondant à des valeurs générées aléatoirement. Ainsi, lorsque le volume externe est monté, il est impossible de conclure quant à la présence d'un volume caché en son sein : dans les deux cas, les contenus des zones dédiées à un volume caché sont indistinguables de données aléatoires.

Figure 2 – Un volume caché dans un volume VeraCrypt standard [14]

Le propriétaire possède alors deux mots de passe : un pour le volume externe et un autre pour le volume caché. Le mot de passe saisi détermine quel volume (caché ou externe) monter. VeraCrypt tente d'abord de déchiffrer l'entête du volume externe en utilisant le mot de passe saisi. S'il réussit, le volume externe est monté. S'il échoue, VeraCrypt charge dans la RAM la zone du volume externe où un entête de volume caché peut être stocké (Table 2, octets 65536-131071) et tente de déchiffrer ces octets en utilisant le mot de passe saisi. Si le déchiffrement est un succès, alors le volume externe en contient un caché que VeraCrypt est en mesure de monter. Le décalage du début des données cachées est déterminé en soustrayant la taille du volume caché à la taille totale des deux zones de données (standarde et cachée). Si le déchiffrement est un échec, c'est que le mot de passe renseigné est incorrect.

De cette manière, si le propriétaire est forcé de donner son mot de passe, il peut donner celui du volume externe, dans lequel il aura placé des documents non sensibles. Le volume caché, indistinguable de données aléatoires, demeurera dissimulé et protégé. Le déni plausible est donc assuré.

Cette fonctionnalité fait écho à la loi en vigueur depuis le 5 juin 2016 et à l'actualité avec un cas récent. Selon l'article 434-15-2 du code pénal [4], refuser de donner aux autorités judiciaires la « convention secrète de déchiffrement d'un moyen de cryptologie » susceptible d'avoir été utilisée pour préparer, faciliter ou commettre un crime ou un délit est puni de 3 ans d'emprisonnement et de 270 000 € d'amende. Un cas récent implique une personne arrêtée pour stupéfiant ayant refusé de donner le code de déverouillage de deux téléphones portables. Le 7 novembre 2022 [6], la Cour de Cassation s'est exprimée sur ce cas et a considéré que ces téléphones portables étaient équipés d'un moyen de cryptologie. Ainsi, toute personne faisant l'objet de suspections est obligée de de divulguer le mot de passe de son matériel informatique. En se plaçant dans le cas d'une personne n'ayant aucune activité illégale, celle-ci ne consentirait pas forcément à donner son code de déverouillage. Son appareil personnel pourrait contenir des données confidentielles et personnelles. Un volume caché serait par conséquent pertinent car il permettrait dans ce cas de pouvoir donner aux forces de l'ordre un mot de passe protégeant des données non sensibles tout en respectant la loi.

3 Cryptographie dans VeraCrypt

3.1 Algorithmes

Lors de la création d'un volume, l'utilisateur doit choisir deux algorithmes cryptographiques : celui de hachage [12] et celui de chiffrement [10]. L'algorithme de hachage est choisi parmi une liste de quatre algorithmes : SHA-256, SHA-512, Whirlpool et Streebog. De même, VeraCrypt propose plusieurs algorithmes de chiffrement dans le but de compliquer toute recherche exhaustive du mot de passe : un attaquant devra tester chaque combinaison de paires d'algorithmes de chiffrement, allongeant exponentiellement son attaque.

Cinq algorithmes de chiffrement symétriques sont proposés : AES, Serpent, Twofish, Camellia et Kuznyechik. Ils peuvent être utilisés tels quels, ou combinés en cascade selon une sélection de dix combinaisons prédéfinies. Par exemple, l'utilisateur peut choisir la combinaison suivante : AES(Twofish(Serpent)). Chaque bloc de données en clair est d'abord chiffré avec Serpent, puis avec Twofish et enfin avec AES. Dans ce cas, chaque algorithme de chiffrement utilise ses propres clés, toutes les clés étant évidemment mutuellement indépendantes entre les différents algorithmes de la combinaison. Les clés utilisées pour chiffrer les données sont stockées concaténées dans les 256 derniers octets de l'entête.

3.2 Random Number Generator

Lors de la création d'un volume sont générées : les clés servant à chiffrer et déchiffrer les données, en taille et en nombre correspondant à l'algorithme de chiffrement choisi ; le sel servant à dériver les clés de l'entête ; et éventuellement des keyfiles aléatoires à combiner avec le mot de passe. Pour cela, VeraCrypt utilise un générateur de nombres aléatoire [16] : le *Random Number Generator* (RNG). Ce RNG est composé d'un True RNG et d'un Pseudo RNG.

Le TRNG est un Hardware RNG [10] capturant des données dès le début de la création d'un volume par l'utilisateur. Ces données correspondent à tous les mouvements de souris, entrés clavier et variables système générées pour en emmagasiner l'aléa. Le PRNG [11] traite ensuite ces données. Il dispose d'un ensemble de 320 octets de données nommé *pool* par la suite. Ce pool est préalablement rempli avec des valeurs aléatoires générées par VeraCrypt à l'aide des différentes sources d'entropies, logicielles et matérielles. Toutes les données capturées par le TRNG sont sommées octet par octet avec les valeurs déjà présentes dans le pool du PRNG, modulo 2^8. Tous les 16 octets ajoutés, une fonction de dispersion appelée *pool mixing function* est appliquée sur le pool.

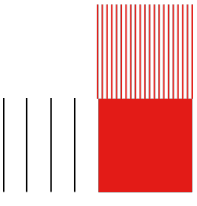
Cette pool mixing function utilise la fonction de hachage (H) choisie par l'utilisateur. L'ensemble du pool est divisé en blocs (Bi) de tailles égales correspondant à la taille de la sortie de la fonction de hachage. Chaque bloc est remplacé par un XOR entre lui-même et le hachage de l'ensemble du pool (Equation 1).

$$\forall i \in \{0..n\}, B_i := B_i \oplus H(B_0||...||B_n) \tag{1}$$

Au final, le pool est rempli de valeurs aléatoires présentant une grande dispersion. De ce pool sont extraites les clés chiffrant les données stockées dans le volume, le sel et les keyfiles si demandés par l'utilisateur.

10. Générateur aléatoire non déterministe dont les bits produits proviennent de l'observation de phénomènes physiques imprédictibles, générés par le matériel hardware et software tels que le temps, l'identifiant du processus, les bruits mécaniques et électriques etc.

11. Générateur aléatoire déterministe dont les bits produits sont générés par des algorithmes déterministes initialisés avec un ensemble de données : la *seed*.



3.3 Application des keyfiles

Un keyfile est un fichier pouvant être combiné au mot de passe pour en accroître la sécurité [15]. Il est choisi par l'utilisateur ou peut être généré aléatoirement (section 3.2). VeraCrypt ne limite ni le nombre de keyfiles (qui peut être nul), ni la taille de ceux-ci. Toutefois, seul le premier méga octet de chaque keyfile est utilisé. Le mot de passe et les keyfiles éventuels sont combinés suivant l'algorithme décrit sur la figure 3 ci-dessous. Celui-ci utilise la fonction de somme de contrôle CRC-32 comme fonction de hachage, avec pour vecteur d'initialisation 0xFFFF FFFF. Cette fonction prend un octet en entrée et en génère quatre en sortie. Cet algorithme dispose aussi d'un vecteur d'octets nommé *keyfile pool* où la combinaison est stockée, qui sera envoyé à la fonction de dérivation PBKDF2 une fois le dernier keyfile combiné (voir section 3.4).

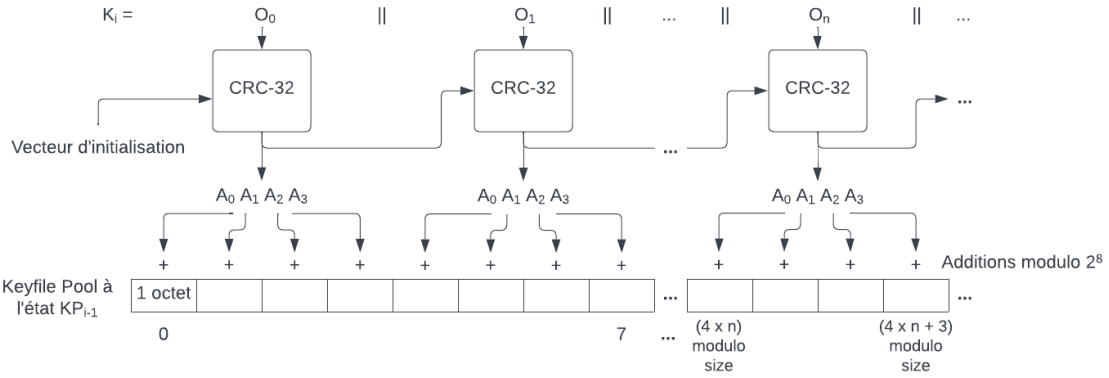


Figure 3 – Combinaison du mot de passe et des keyfiles

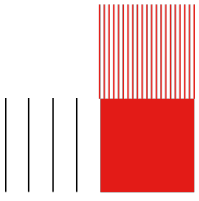
Le i[ème] keyfile K_i est décomposé en octets : O_0, O_1 etc. Chaque octet est haché par CRC-32 générant 4 octets ajoutés aux octets du keyfile pool modulo 2^8, le stockage étant sur 8 bits. Le keyfile pool KP_i correspond à l'application du i[ème] keyfile au keyfile pool KP_{i-1}. Avant l'application du premier keyfile, le keyfile pool KP_{-1} est initialisé avec le mot de passe de l'utilisateur codé en utf-8. Sa taille détermine celle du keyfile pool : si elle est inférieure à 64 octets alors le keyfile pool a une taille de 64 octets. Sinon, la taille du keyfile pool est de 128 octets, celle-ci étant la taille maximale d'un mot de passe. Le mot de passe est complété de zéros s'il est plus court que le keyfile pool.

3.4 Dérivation des clés

Les clés utilisées pour chiffrer et déchiffrer l'entête sont obtenues à l'aide de la fonction de dérivation PBKDF2 [13]. Celle-ci applique un HMAC [12] (basé sur la fonction de hachage choisie par l'utilisateur) sur le sel et la combinaison mot de passe - keyfiles (voir section 3.3), et itère cette opération des milliers de fois. Le nombre d'itérations peut être impacté par l'utilisateur à travers le choix d'un *Personal Iterations Multiplier* (PIM). Par défaut, le PIM est initialisé par VeraCrypt à 485, ce qui assure un haut niveau de sécurité. La fonction appliquée au PIM diffère suivant l'algorithme de hachage utilisé. Par exemple, si l'utilisateur choisit SHA-512 le nombre d'itérations vaut alors `15000+(PIM*1000)`.

Plusieurs chaînes d'itérations de la fonction HMAC sont calculées (Figure 4). Au début d'une chaîne, le sel est concaténé avec un compteur en entrée i, différent selon la chaîne, avant de subir les itérations de HMAC. Chaque HMAC prend en entrée le mot de passe et les keyfiles combinés. La sortie est directement envoyée au prochain HMAC et xorée avec le résultat du précédent.

12. HMAC est un protocole cryptographique permettant l'authentification de messages. Cette authentification se fait par le calcul d'une opération de hachage combinée avec une clé secrète et sa force dépend de la fonction de hachage sous-jacente et du choix de la clé secrète.



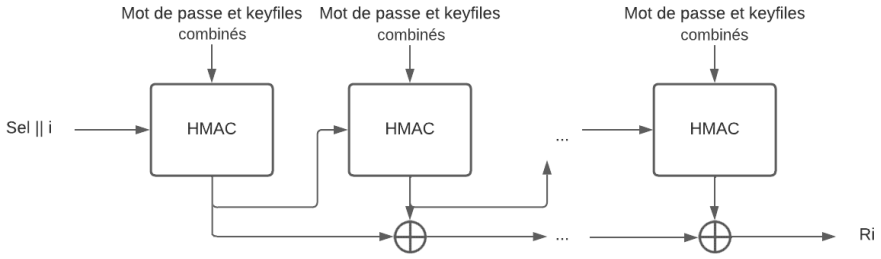


Figure 4 – Une chaine de dérivation de PBKDF2 [7]

Une fois toutes les itérations de HMAC effectuées pour chaque chaîne de dérivation, les sorties de chaque chaîne R_i sont concaténées pour donner les clés finales de l'entête. Le nombre de chaînes à calculer dépend donc directement du nombre de clés et de leur longueur, et par conséquent de l'algorithme de chiffrement utilisé.

3.5 Mode de chiffrement

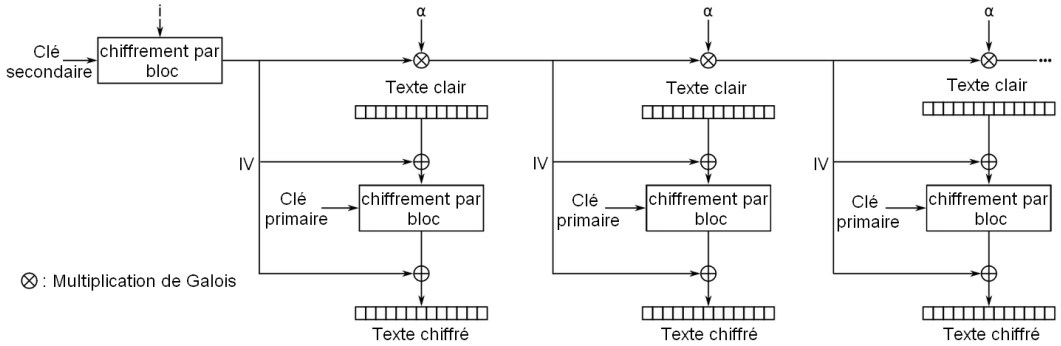


Figure 5 – Le mode de chiffrement XTS [1]

L'entête et le corps d'un volume VeraCrypt sont chiffrés avec le même mode : XTS (Figure 5). Ce mode est dédié au chiffrement de disques durs et utilise deux clés : la clé primaire et la clé secondaire. Les données en clair sont divisées en blocs dont la taille correspond à la taille d'entrée de l'algorithme de chiffrement choisi. À chaque bloc des données en clair est tout d'abord appliqué un XOR bit à bit avec un vecteur d'initialisation (IV). Ce XOR est ensuite chiffré avec la clé primaire. À ce chiffré est enfin appliqué une seconde fois un XOR avec un vecteur d'initialisation. Le premier bloc utilise comme vecteur d'initialisation le numéro du secteur (où l'on stocke les données, basé sur l'adresse physique) chiffré avec la clé secondaire. Cela permet de diversifier le chiffrement. Les blocs suivants altèrent le vecteur d'initialisation du bloc précédent en lui appliquant une multiplication de Galois [13] avec un entier α [14]. Chaque bloc d'un même secteur possède donc un vecteur initial différent. XTS assure l'unicité de chaque couple (clé secondaire | numéro de secteur).

Grâce au mode de chiffrement XTS, chaque bloc peut être chiffré et déchiffré indépendamment des autres car l'accès y est direct. À cette parallélisation s'ajoute la possibilité de pré-calculer les vecteurs d'initialisation, ce qui fait gagner un temps non négligeable à la lecture d'une partie spécifique du volume.

13. Multiplication modulaire, permettant de retomber dans l'espace de départ.
14. Cet alpha est généré par XTS, il correspond à un élément primitif a appartenant au champ de Galois GF(2) modulo 128 (soit 2^{128}), élevé à la puissance l'indice du bloc à chiffrer i, ie. $\alpha = a^i$.

4 Exploitation des cartes EMV pour VeraCrypt

4.1 Alternative EMV

Actuellement, l'approche de stockage du ou des keyfiles sur une carte PKCS#11 est la plus sécurisée, mais est dédiée à des utilisateurs appartenant au monde de la cybersécurité, qu'ils soient des professionnels ou des amateurs compétents. De plus, elle réduit drastiquement le déni plausible de l'utilisateur. En effet, la présence d'une carte PCKS#11 dans le portefeuille d'un reporter peut attirer l'attention et la suspicion des autorités lors de son passage par les douanes d'un état hostile.

L'utilisation de cartes suivant la norme EMV (voir section 4.2) est une alternative à l'approche mentionnée ci-dessus. Les cartes bleues utilisées en France sont de type EMV. Au-delà de la France, ces cartes sont extrêmement répandues dans le monde entier pour une utilisation sans aucun lien avec le chiffrement de volumes virtuels. Avec le passeport, il s'agit certainement des deux dispositifs les plus répandus sur le globe et normés par des standards internationaux. La quasi-totalité des utilisateurs de VeraCrypt possèdent donc au moins une carte EMV.

Utiliser des données internes à la carte EMV du créateur du volume comme keyfiles est donc une façon de mettre en place sa double authentification. Ces données doivent être suffisamment aléatoires pour qu'un adversaire ne puisse pas les déduire, tout en étant propres à la carte. Cette approche est toutefois moins sécurisée que celle par carte PKCS#11. En effet, sur une carte EMV l'accès aux données ne requiert pas d'authentification contrairement aux PKCS#11 qui exigent un code PIN. L'avantage réside dans le déploiement à grande échelle permettant de conserver la propriété de déni plausible de l'utilisateur. De plus la facilité d'utilisation des cartes EMV et le fait que ses données utilisées comme keyfiles ne soient pas stockées sur le disque dur font de cette approche la meilleure alternative possible.

4.2 Standard EMV

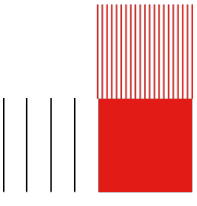
EMV [26] est un standard technique pour les cartes de paiement et les terminaux pouvant les accepter. EMV est l'abréviation de "Europay, Mastercard et Visa", les trois entreprises à l'origine de ce standard. Ce dernier est actuellement managé par EMVCo, un consortium dans lequel le contrôle est réparti équitablement entre Visa, Mastercard, JCB, American Express, China UnionPay, et Discorver.

Les cartes de paiement sous le standard EMV sont des *cartes à puces*. Elles stockent leurs informations non seulement sur un circuit intégré, mais aussi sur une bande magnétique par souci de rétrocompatibilité. Ces bandes magnétiques sont cependant vouées à disparaitre dès 2024. Les cartes EMV doivent être insérées dans un lecteur, ou en être à proximité dans le cas des cartes sans contact utilisant la technologie NFC [15], pour permettre d'interagir avec les données à partir du terminal. Les standards de communication sont basés sur l'ISO/IEC 7816 [3] pour les cartes avec contact et l'ISO/IEC 14443 [2] pour celles sans contact.

4.3 Déploiement

Avant l'utilisation des cartes à puces pour les transactions physiques, les cartes de crédit ou à débit utilisaient des pistes magnétiques ou des empreintes mécaniques pour lire et enregistrer les données du compte, et une signature sur la carte pour la vérification de l'identité du payeur. L'arrivée des premières cartes à puces dans les années 70 puis du standard EMV en 1995 a tout d'abord permis une nette amélioration de la sécurité dans le but d'une forte réduction de la fraude. Elle a aussi permis un contrôle plus précis des transactions par carte bancaire « hors ligne » ne nécessitant pas de demande d'autorisation bancaire. L'EMV permet l'utilisation d'un code PIN et d'algorithmes cryptographiques tels que DES, Triple-DES, RSA et AES [26] afin de permettre à la carte et au terminal de s'authentifier mutuellement. Le standard garantit ainsi l'interopérabilité entre les cartes EMV et les terminaux de paiement EMV à travers le monde.

15. Near-Field Communication



4.4 Commandes

Selon la norme ISO/IEC 7816 sur laquelle s'appuie le standard EMV, le lecteur est maître du dialogue : il émet les commandes auxquelles la puce répond après traitement. En particulier, la partie 3 de la norme définit les protocoles de transmission pouvant être utilisés pour les échanges entre le lecteur et la carte à puce en mode contact. La partie 4 quant à elle définit le format des échanges au niveau applicatif, basé sur les APDU [16]. Les APDU permettent de structurer les commandes échangées entre la carte et le lecteur.

Une commande APDU [3] envoyée depuis un lecteur contient un entête de 4 octets obligatoire (CLA, INS, P1, P2) et entre 0 et 65 535 octets de données, tandis que la réponse de la carte contient entre 0 et 65 535 octets de données puis deux octets obligatoires pour les status (SW1, SW2).

Table 3 – APDU

Commande APDU		
Nom	Taille (octets)	Description
CLA	1	Classe d'instruction : type de la commande (ex : interindustry, proprietary)
INS	1	Code d'instruction : code de la commande (ex : read data)
P1	1	Paramètres de la commande (ex : l'offset du curseur)
P2	1	
L_c	0,1 ou 3	**Nombre (N_c) d'octets de données qui suivent.** 0 octet : N_c prend la valeur 0. 1 octet : de valeur entre 1 et 255 qui exprime N_c. 3 octets, avec le premier égal à 0 : N_c prend une valeur entre 1 et 65 535.
Command data	N_c	N_c octets de données
L_e	0,1,2 ou 3	**Le nombre maximal (N_e) d'octets de réponse attendu.** 0 octet : N_e prend la valeur 0 1 octet : de valeur entre 1 et 255 qui exprime N_c ou 0 exprime $N_e = 256$. 2 octets (si L_c étendu était présent dans la commande) : de valeur entre 1 et 65 535 qui exprime N_e ou deux 0 qui exprime $N_e = 65536$. 3 octets avec le premier égal à 0 (si L_c étendu était présent dans la commande) : exprime N_e de la même façon qu'avec deux octets.

Réponse APDU		
Nom	Taille (octets)	Description
Response data	N_r au maximum N_e	N_c octets de données
SW1	1	Status de la commande (ex : 90 00 en hexadécimal indique un succès)
SW2	1	

4.5 Sélection d'applications

Un des buts principaux de EMV est de fournir des cartes contenant plusieurs applications à la fois (cartes multi-applicatives). Ces applications peuvent être par exemple des applications pour débiter le compte courant, pour débiter la réserve de crédit associée distincte du compte courant, un porte-monnaie électronique (ex : Moneo). EMV décrit dans le livre 1 du standard l'implémentation de la sélection d'application de façon à pouvoir identifier le type de produit. Ainsi, chaque émetteur (Mastercard, Visa, etc.) dispose de sa propre application et est identifié grâce à un RID [17] unique de cinq octets. L'AID [18] (voir l'exemple en Figure 6) qui permet d'adresser les différentes applications est donc composé du RID et du PIX [19] qui permet aux émetteurs de différencier l'application des différentes applications qu'ils proposent. Pour sélectionner une application en particulier, il faut donc envoyer la commande `SELECT FILE` + AID sous la forme d'une APDU à la carte.

16. Application Protocol Data Unit
17. Registered application provider IDentifier
18. Application IDentifier
19. Proprietary application Identifier eXtension

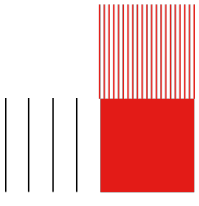


Figure 6 – Structure de l'AID de l'application Visa Electron

4.6 Lecture des données

Les données d'une carte à puce sont stockées dans des fichiers qui peuvent être lus grâce à la commande `read record`. EMV ne spécifie pas quels fichiers contiennent des données, c'est pourquoi ils doivent tous être lus pour le savoir. Le standard définit cependant des `tags` pour chaque type de données. Les données sont stockées dans les fichiers en BER-TLV [20], un format d'encodage défini par le standard ASN.1. Il s'agit d'un format qui se décrit et se délimite lui-même : chaque élément est encodé avec un type, une longueur, les valeurs et si nécessaire un marqueur de fin. Cela permet ainsi au récepteur de décoder le message sans avoir de connaissances a priori de la taille, du contenu et de la sémantique des données, et ce même si le flux est incomplet.

4.7 Interfacage

L'un des objectifs de ce projet est le développement d'un module extrayant les données de la carte EMV. Ce module, baptisé EMVExtractor, doit assurer la communication entre VeraCrypt et la carte EMV à travers le lecteur de cartes de l'utilisateur. Le marché des lecteurs de cartes étant conséquent, l'un des défis de ce projet est de pouvoir communiquer avec un nombre non exhaustif de lecteurs différents, pouvant évoluer avec le temps. Pour faire face à cette diversité de lecteurs, Microsoft a développé le standard de communication *Personal Computer/Smart Card*. Toute librairie suivant le standard PC/SC permet de communiquer avec la carte à travers le driver par le biais d'une gamme de fonctions standardisées transmettant des APDU.

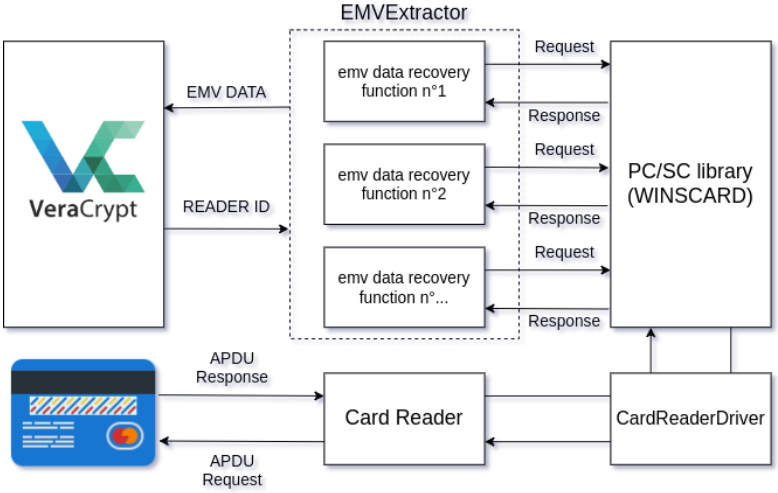


Figure 7 – Schéma de l'interaction entre VeraCrypt et une carte EMV

20. Basic Encoding Rules - Type-Length-Value

La librairie Winscard suit le standard PC/SC et est disponible à la fois nativement sous Windows, mais aussi sous Linux ou macOS via le paquet PCSCLite. Par conséquent, elle correspond parfaitement à la contrainte de développement multiplateforme imposé dans le cahier des charges. Windscard propose directement des fonctions compatibles avec les trois systèmes d'exploitation requis, qui seront utilisées par le module de communication EMVExtractor. Son rôle sera de parcourir à l'aide d'APDU les différentes applications de la carte bancaire afin de vérifier la présence d'au moins l'une des trois applications requises (Visa, American Express ou MasterCard) et en récupérer les données le cas échéant.

4.8 Données utilisables

4.8.1 Certificats

Lors d'une transaction bancaire avec une carte EMV, l'application utilisée s'authentifie auprès de la banque afin que cette dernière vérifie l'intégrité de certaines données. Le protocole EMV définit trois types d'authentification : l'authentification statique des données (SDA [21]), l'authentification dynamique des données (DDA [22]) et l'authentification combinée des données (CDA [23]). Ces données sont propres à l'instance de l'application déployée sur la carte EMV. De plus, elles sont statiques et identifiées par des tags uniques. Chaque application disponible sur une carte EMV contient donc ses propres données authentifiant la carte.

La SDA a été la première méthode d'authentification à naître au début de l'ère EMV. Afin de réaliser l'authentification statique des données, la carte EMV fournit les données présentes dans le tableau ci-dessous (Figure 4) extrait du livre 3 de la norme EMV [8].

Table 4 – Table des données requises pour la SDA

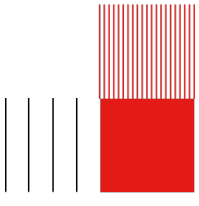
Tag	Value
'8F'	Certification Authority Public Key Index
'90'	Issuer Public Key Certificate
'93'	Signed Static Application Data
'92'	Issuer Public Key Remainder
'9F32'	Issuer Public Key Exponent

Cette authentification présente cependant des défauts. Les informations utilisées étant lisibles et statiques par définition, un attaquant peut facilement les copier sur une carte factice et de ce fait cloner la carte EMV. Elle présente aussi un défaut de sécurité en cas de compromission de la clé privée de l'émetteur. La SDA a donc été remplacée au fur et à mesure par de la DDA et de la CDA, palliant ses défauts. Pour effectuer ces deux types d'authentification, de nouvelles données sont requises (voir Table 5).

Table 5 – Table des données requises pour la DDA ou CDA

Tag	Value
'8F'	Certification Authority Public Key Index
'90'	**Issuer Public Key Certificate**
'92'	Issuer Public Key Remainder
'9F32'	Issuer Public Key Exponent
'9F46'	**ICC Public Key Certificate**
'9F47'	ICC Public Key Exponent
'9F48'	ICC Public Key Remainder
'9F49'	Dynamic Data Authentication Data Object List

21. Static Data Authentication
22. Dynamic Data Authentication
23. Combined Data Authentication



L'**ICC Public Key Certificate**, évoqué dans le cahier des charges par M. Dupas, est présent parmi ces données et est identifié par le tag '9F46'. S'y trouve aussi un autre certificat : l'**Issuer Public Key Certificate**, identifié par le tag '90'. Servant pour l'authentification dynamique des données dans le cadre de la norme EMV, ces deux certificats sont statiques et non déductibles. Leur taille est décrite comme variable dans le standard, mais une étude par l'équipe du projet a permis d'estimer leur taille entre 176 et 200 octets. La taille du keyfile pool étant de 64 octets, un seul de ces certificats permet de faire environ trois tours de combinaison avec le mot de passe. Ainsi, ce sont deux très bons candidats dans le cadre de l'utilisation des données d'une carte EMV pour les utiliser tels quels comme keyfiles.

4.8.2 CPLC

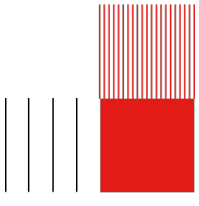
Les données présentes dans les applications ne sont pas les seules disponibles sur une carte EMV. En effet, sur toute carte à puce suivant le standard EMV se trouvent à la racine des données appelées **Card Production Life Cycle** identifiées par le tag '9F7F'. Ces CPLC ne sont propres qu'à la carte et n'ont aucun lien avec les applications qui y sont déployées. Elles concernent le processus de production de la carte à puce. Une commande APDU spéciale, différente de celle pour les applications, permet la lecture de ces données : `get data`. En analysant les données CPLC de nombreuses cartes à l'aide du logiciel CardPeek [24] [20], elles regroupent les informations présentes dans le tableau ci-dessous (Figure 6) où IC réfère à la carte, ICC à la puce de la carte et OS à son système d'exploitation.

Table 6 – Ensemble des données CPLC

IC Facbricator	IC Type	OS Provider ID
OS Release Date	OS Release Level	IC Fabrication Date
IC Serial Number	IC Batch ID	IC Module Fabricator
IC Module Packaging Date	ICC Manufacturer	IC Embedding Data
Prepersonalizer ID	Prepersonalization Date	Prepersonalisation Equipment
Personalizer ID	Personalization Date	Personalization Equipment

Chaque information est codée sur deux octets, sauf trois qui le sont sur quatre, pour un total de 42 octets. Leur signification n'est pas expliquée dans le livre 3 d'EMVco, et toute recherche sur internet ne ressort que des explications floues sur le processus de production des cartes à puces. Chaque fabricant de cartes possède un identifiant, mais il est impossible de trouver une liste fiable et à jour regroupant tous les identifiants et le nom du fabricant associé. Toutefois, ces données CPLC sont directement utilisables comme keyfiles de par leur taille et leur non déductabilité.

24. Logiciel open-source, cet outil permet la lecture de carte à puce avec une interface graphique basée sur GTK, fonctionnant sous GNU Linux/Windows/MacOS X.



5 Code de VeraCrypt

5.1 Architecture logicielle

Le code à développer devant être multiplateforme comme défini dans le cahier des charges du projet, les codes des versions Windows et Linux/macOS de VeraCrypt ont été étudiés. Ces deux versions utilisent du code en commun pour certaines fonctionnalités mais disposent chacune de leur propre version de la majorité du code source. Cette duplication est tout particulièrement visible lors de l'utilisation des keyfiles. Elle est réalisée en langage C pour Windows et en C++ pour Unix. Cette duplication est aussi vraie pour l'interface graphique qui utilise l'`API Windows` pour Windows et la librairie `WxWidget` pour Unix. L'intégration de la fonctionnalité EMV devra potentiellement être faite deux fois pour ces éléments. Mounir Idrassi affirme que l'approche C++ de Linux va dans le sens d'une amélioration de la qualité du code, et confirme le développement du code de ce projet dans ce langage.

Les environnements de développement sont eux aussi dupliqués, Windows utilisant `Microsoft Visual Studio 2010` pour les versions 32 bits et 64 bits et Linux utilisant `CMAKE`. Il est donc nécessaire de prendre en main ces deux environnements pour compiler et tester les versions pour ces deux OS.

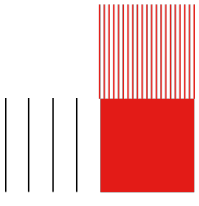
5.2 Attribution des chemins

Lors de la dérivation des clés de l'entête d'un volume, VeraCrypt récupère la liste des chemins d'accès des keyfiles à utiliser. Si l'utilisateur renseigne des fichiers stockés sur le disque ou sur une mémoire amovible, leur chemin absolu est ajouté à la liste. Si l'utilisateur sélectionne un dossier de keyfiles, seuls les chemins absolus des fichers stockés dans le dossier sont ajoutés à la liste. Il n'y a pas de récursivité dans les sous-dossiers. Si l'utilisateur stocke ses keyfiles sur une carte PKCS#11, VeraCrypt leur attribue des chemins d'accès particuliers. Dans la version actuelle du logiciel, ces chemins suivent le format suivant : `token://slot/<reader_id>/file/<file_id>`. Les paramètres variables sont `reader_id` et `file_id` représentant respectivement le numéro du lecteur de cartes et le numéro du fichier sur la carte à utiliser.

La sélection par l'utilisateur des keyfiles à utiliser diffère suivant la manière dont il utilise VeraCrypt. Via l'interface en ligne de commande, les keyfiles sont renseignés en passant manuellement la liste des chemins d'accès en argument, sous la forme `keyfiles=[...]`. Via l'interface graphique, il n'est pas possible de rentrer les chemins manuellement. À la place, une boîte de dialogue séparée apparaît, permettant de sélectionner un fichier, un dossier ou un fichier sur une carte PKCS#11. Est alors automatiquement créé et ajouté à la liste le chemin d'accès particulier du keyfile.

5.3 Application des keyfiles

Les keyfiles renseignés dans la liste sont ensuite traités selon l'organigramme visible sur la Figure 8 ci-dessous. La liste est passée en paramètre de la fonction `ApplyListToPassword` de la classe `Keyfile`, qui applique à chaque chemin la fonction `Apply`. Ce chemin d'accès est alors passé en paramètre de la fonction `IsKeyfilePathValid` de la classe `SecurityToken`. Cette fonction renvoie un booléen représentant la présence de «token ://» en début du chemin passé en paramètre. Elle permet donc d'identifier les chemins correspondant à des keyfiles stockés sur une carte PKCS#11. Dans ce cas-là, la fonction `GetKeyfileData` est appelée afin d'aller récupérer les données sur la carte PKCS#11 sous la forme d'un vecteur d'octets. Dans le cas contraire, le chemin désigne un fichier lambda dont les données sont directement récupérées sur le disque et stockées dans un vecteur d'octets. Enfin, au maximum un mega octet des données du vecteur d'octets est combiné avec le mot de passe de l'utilisateur, suivant l'algorithme détaillé en section 3.3.



5.4 Adaptation aux cartes EMV

L'utilisateur pourra choisir d'utiliser les données d'une carte EMV comme keyfiles. De manière similaire au chemin des fichiers des cartes PKCS#11, les cartes EMV se verront attribuer un chemin particulier suivant le format `emv://slot/<reader_id>`. Seul le numéro `reader_id` est variable, permettant de préciser quel lecteur de cartes utiliser pour lire les données. Le paramètre <file_id> disparaît puisque, contrairement aux cartes PKCS#11, l'utilisateur n'aura plus besoin de choisir un fichier sur la carte. Ce format sera utilisable en ligne de commande lors du renseignement des chemins des keyfiles par l'utilisateur. L'interface graphique sera modifié en conséquence (voir section 5.5) afin de permettre de choisir le lecteur de cartes à utiliser.

L'extraction des données d'une carte EMV sera implémentée dans une nouvelle classe `EMVToken`, dont le fonctionnement sera similaire à la classe `SecurityToken` gérant les cartes PKCS#11. Elle vérifiera que le chemin passé en paramètre de sa fonction `IsKeyfilePathValid` contient «emv ://», récupèrera alors les données de la carte EMV et les stockera dans un vecteur d'octets. La vérification se fera après celle de la classe `SecurityToken`. Les ajouts sont visibles dans les cases vertes en pointillé sur la Figure 8.

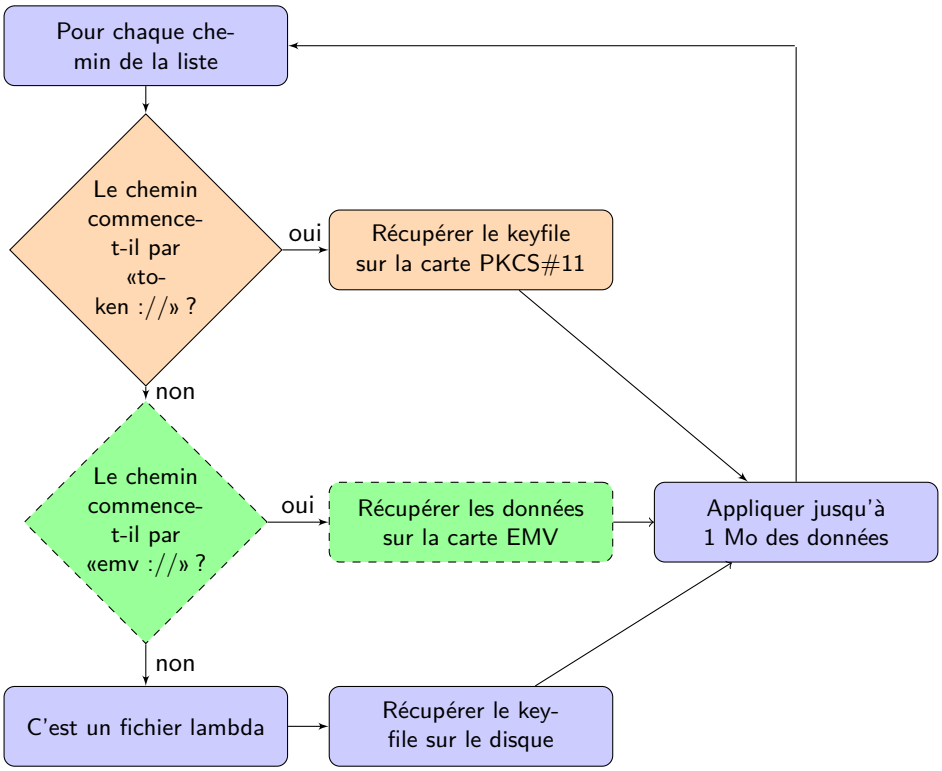


Figure 8 – Organigramme simplifié de la fonction de traitement des keyfiles, avec les ajouts nécessaires en pointillés

5.5 Maquette de l'interface graphique

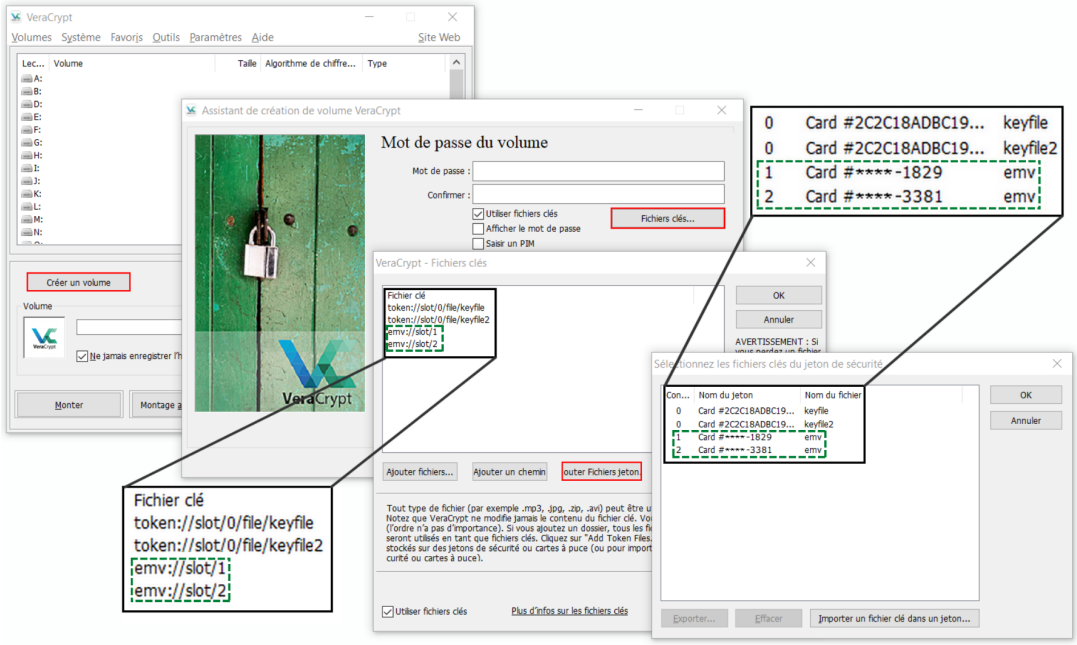
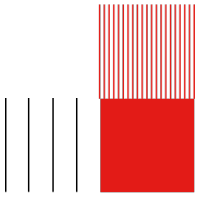


Figure 9 – Maquette graphique de l'intégration des cartes EMV

Version actuelle Dans la version existante (Figure 9 sans les encadrés verts en pointillés), un utilisateur souhaitant créer un volume passe par plusieurs étapes. Il clique tout d'abord sur «*Créer un volume*» dans la fenêtre principale de VeraCrypt. Dans la nouvelle fenêtre (*Assistant de création de volume VeraCrypt*), il renseigne le chemin du fichier à créer, les algorithmes de chiffrement et de hachage, la taille, le mot de passe et enfin les keyfiles. Pour préciser quels keyfiles il souhaite utiliser, il clique sur «*Fichiers clés*». Une nouvelle fenêtre s'ouvre (*VeraCrypt - Fichiers clés*) où s'affiche la liste des chemins des keyfiles sélectionnés par l'utilisateur, conformément à la description en partie 5.2.

Pour ajouter à cette liste un keyfile stocké sur une carte PKCS#11, l'utilisateur clique sur «*Ajouter Fichiers jetons*». Dans cette dernière fenêtre (*Sélectionnez les fichiers clés du jeton de sécurité*) sont analysées les cartes PKCS#11 connectées aux lecteurs branchés sur l'ordinateur et affichés les keyfiles disponibles. Lors du montage d'un volume, l'utilisateur accède directement aux deux dernières fenêtres afin de renseigner les keyfiles nécessaires au déchiffrement. Dans la figure 9 (sans les encadrés verts en pointillés) est simulée la présence d'un seule carte PKCS#11 connectée au lecteur du slot 0 et contenant deux keyfiles utilisables. Ceux-ci ont été sélectionés par l'utilisateur et se sont vu attribués des chemins spéciaux commençant par 'token ://', ajoutés à la liste dans la fenêtre *Fichiers clés*.

Modifications Dans le but de simplifier la sélection des données d'une carte EMV comme keyfiles par l'utilisateur (Figure 9 avec les encadrés verts en pointillés), le choix des cartes EMV se fera naturellement lorsque l'utilisateur accèdera à la fenêtre «*Sélectionnez les fichiers clés du jeton de sécurité*». En plus de vérifier l'ensemble des keyfiles présents sur les cartes PKCS#11 connectées, VeraCrypt analysera les lecteurs présentant des cartes EMV et affichera ces dernières. Dans le champ «*Nom du jeton*», les quatres derniers numéros de la carte apparaîtront tandis que le champ «*Nom du fichier*» précisera la nature EMV de la carte. Les cartes EMV sélectionnées verront leur chemin commencer par 'emv ://' lorsqu'elles seront ajoutées à la liste dans la fenêtre *Fichiers clés*. Dans les encadrés verts en pointillés de la figure 9 est simulée la présence additionnelle de deux cartes de type EMV finissant respectivement par 1829 et 3381, respectivement connectées aux lecteurs des slots 1 et 2, sélectionnées par l'utilisateur.



6 Proofs Of Concept

6.1 Premier POC : Extraction des données EMV

Après avoir étudié le standard EMV et sélectionné les données utilisables comme keyfiles, l'objectif suivant est l'extraction de ces données. Pour cela, un court programme démonstratif a été développé, permettant de mieux cerner les étapes à suivre et les problèmes à résoudre. Ce programme est un premier *Proof Of Concept* (POC) dont le rôle est de démontrer la possibilité de communiquer avec une carte EMV et d'en extraire les certificats et données (voir sections 4.8.1 et 4.8.2) pour les afficher dans la sortie standard. Le langage de script Python a été retenu pour sa facilité et sa rapidité de programmation, le rendant idéal pour un premier POC. Windscard étant utilisable seulement dans du code en langage C/C++, une librairie Python suivant le standard PC/SC a été utilisée : PyScard [5]. Afin de manipuler plus aisément le format BER-TLV dans lequel les données des cartes EMV sont encodées, la librairie Python Ber_TLV [25] a été utilisée. PyScard permet de coder en dur dans des variables des commandes APDU, partielles ou complètes, sous la forme de tableaux de valeurs en hexadécimal (voir ci-dessous).

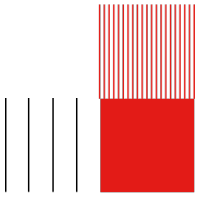
```
# define the apdus used in this script
GET_RESPONSE = [0XA0, 0XC0, 00, 00]
SELECT = [00, 0xA4, 0x04, 00, 0x07]
GET_DATA = [0x80, 0xCA]
CPLC_ID = [0x9F, 0x7F]
# Application Identifier (AID) on 7 bytes + 1 byte to complete the apdu
MASTERCARD = [0xA0, 00, 00, 00, 0x04, 0x10, 0x10, 0x00]
VISA = [0xA0, 00, 00, 00, 0x03, 0x10, 0x10, 0x00]
AMEX = [0xA0, 00, 00, 00, 00, 0x25, 0x10, 0x00]

APPS = {"MASTERCARD": MASTERCARD, "VISA": VISA, "AMEX": AMEX}
```

Ces définitions de variables globales permettent de programmer plus rapidement et d'avoir une bonne vision des différentes parties des APDU utilisées. Le POC est basé sur une fonction principale, `get_emv_data()`, extrayant les données de la carte EMV en appelant les deux sous-fonctions `get_cplc()` et `get_certificates(name)`, récupérant respectivement les données CPLC et les certificats de l'application dont le nom a été passé en paramètre. La vérification de la présence d'au moins l'une des trois applications requises est assurée par la fonction `connect_app(name)`. Appelée dans une boucle `for` parcourant le tableau `APPS` répertoriant Visa, Mastercard et American Express, elle essaie de se connecter à l'application dont le nom est passé en paramètre et renvoie un booléen mis à faux en cas d'échec (absence de l'application) et vrai dans le cas contraire (présence de l'application et connexion réussie).

```
def get_emv_data():
    card_ok = False
    get_cplc()
    for name in APPS:
        if(connect_app(name)):
            card_ok = True
            get_certificates(name)
    if(not card_ok):
        print("Error : this card doesn't contain any application supported by this script.")
get_emv_data()
print_cplc()
print_certificates()
```

Ce programme[25] a été testé sur une dizaine de cartes à puces de type EMV possédant les applications bancaires Visa et Mastercard. Malheureusement, n'étant pas en possession de carte American Express et n'ayant pas la possibilité d'en obtenir une malgré les multiples recherches de solutions, cette application n'a pu être testée pour l'instant. Cependant, le succès de ce Proof Of Concept en Python a permis de démontrer la faisabilité technique de l'extraction des données d'une carte EMV.

25. Premier POC en python : https://github.com/veracrypt-EMV-INSA-Rennes-4INFO/EMV_Extractor/blob/main/PYCC_Extractor/PYCCExtractor.py



6.2 Deuxième POC : Intégration basique dans VeraCrypt

Une fois le premier POC en Python réalisé, une traduction en langage C a été développée [26]. La classe utilitaire `EMVExtractor.c` a été créée, contenant les fonctions nécessaires à la communication avec une carte EMV et permettant de récupérer les certificats et données CPLC. Cette communication se fait via la librairie `Winscard` et une deuxième classe `Tlv.c` contenant les fonctions pour manipuler le format BER-TLV. Cet utilitaire reprend une structure analogue à celle du POC en Python, à quelques exceptions près. Les lecteurs de cartes sont détectés grâce à la fonction `GetReaders()` et celui utilisé est sélectionné par `ConnectCard(int reader_nb)`. La fonction dénommée `GettingAllCerts(unsigned char* ICC_DATA, int* ICC_DATA_SIZE)` permet la récupération et le stockage (dans le tableau ICC_DATA passé en paramètre) des données CPCL, du premier `ICC Public Key Certificate` et du premier `Issuer Public Key Certificate` de la carte si elle suit le standard EMV. Ces données concaténées sont par la suite utilisées en tant que keyfile pour le chiffrement de volume. Cette classe a ensuite été utilisée pour réaliser une première intégration dans VeraCrypt, se concentrant sur l'utilisation en ligne de commande. La classe `EMVToken.cpp` a vu le jour, contenant les signatures de fonction suivantes :

```
class EMVToken {
    public:
        static void GetKeyfileData (const EMVTokenInfo &keyfile, vector <byte> &keyfileData);
        static bool IsKeyfilePathValid (const wstring &securityTokenKeyfilePath);
};
```

La première méthode fait appel aux fonctions d'EMVExtractor.c afin d'insérer dans le *vector* d'octets passé en paramètre le contenu des certificats et des données CPLC de la carte. Cette dernière est précisée dans à la structure `EMVTokenInfo` passée en paramètre, contenant entre autres le numéro du lecteur et une fonction permettant de générer le chemin d'accès associé à la carte. La deuxième méthode de la classe renvoie un booléen représentant la présence d' «emv ://» en début du chemin passé en paramètre.

Enfin, la classe `Keyfiles.c` pour Windows, et `Keyfile.cpp` pour Linux [27] ont été modifiées pour ajouter l'utilisation des cartes EMV. L'ordre de vérification des chemins y a été codé conformément à la section 5.4. Afin de compiler ce deuxième POC ont été modifié en conséquence le Makefile pour Unix et le projet VisualStudio pour Windows. La version de VeraCrypt obtenue est fonctionnelle, multiplateforme et permet l'usage de cartes EMV en ligne de commandes. La création d'un volume se fait avec les commandes suivantes, les paramètres à préciser étant le chemin du fichier où stocker le volume dans `<path>` et le numéro du lecteur dans `<reader_id>` :

Sous Unix :

```
./veracrypt -t -c <path> --password="password" --keyfiles="emv://slot/<reader_id>" --size=10M
```

Sous Windows :

```
"VeraCrypt Format" /create <path> /password password /keyfile emv://slot/<reader_id> /size 10M
```

Le montage se fait de manière analogue :

Sous Unix :

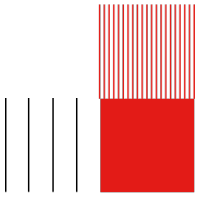
```
./veracrypt -t <path> --password="password" --keyfiles="emv://slot/<reader_id>"
```

Sous Windows :

```
veracrypt /v <path> /l x /a /p password /keyfile emv://slot/<reader_id>
```

Ce deuxième POC [28] est aussi un succès et permet l'utilisation de cartes EMV supportant les applications Visa et Mastercard, les test d'American Express étant toujours impossibles. Des tests ont montré que l'utilisation d'une carte EMV ne rallonge que de quelques secondes le temps de création et de montage de volume. Des tests plus poussés et rigoureux seront effectués et présentés dans un futur rapport.

26. Ayant été codée avant la réunion avec M. Idrassi (voir section 7.3), afin de suivre ses recommendations elle sera traduite en C++ par la suite pour la version finale du projet.
27. Ces fichiers contiennent un code similaire, mais ne sont pas situés au même endroit à cause de la base de code différente entre la version Windows et Linux/MacOS. De plus l'un contient un «s» à la fin du nom, mais pas l'autre.
28. POC en C : `https://github.com/veracrypt-EMV-INSA-Rennes-4INFO/EMV_Extractor/tree/main/ICC_Extractor`



7 Gestion de projet

Nous avons choisi de nous orienter vers une méthode agile pour mener à bien ce projet. En effet, celle ci nous semble adéquate :

1. Nous pouvons facilement découper le projet en taches élémentaires, les classer en fonction de leur avancement, difficulté, etc ; et définir des *milestones* en conséquence.
2. Nous privilégions la communication et la collaboration entre les membres du projet et équipes formées pour avancer dans le projet et s'organiser.

Néanmoins, nous nous éloignons d'une méthode agile conventionnelle sur certains points, notamment de par la présence d'un cahier des charges.

7.1 Réunions et outils

Nous nous réunissons chaque semaine avec nos encadrants Gildas Avoine et Jules Dupas sur le créneau prévu dans l'emploi du temps, dans le but de discuter des points suivants :

1. Avancement technique du projet.
2. Difficultés rencontrées.
3. Répartition des rôles et des tâches.
4. Outils.
5. Rapports à rendre.
6. Notions techniques.

Notre organisation va de pair avec l'utilisation des outils suivants :

— **Discord** est notre outil de communication interne et de réunions en distanciel lorsque le besoin s'en fait sentir.

— **Zimbra** est la messagerie de l'INSA que nous utilisons pour communiquer avec nos encadrants.

— **Google Drive** nous permet de partager des documents entre nous.

— **Notion** est un site web nous permettant de rédiger proprement dans un style markdown les comptes rendus lors des réunions notamment.

— **Overleaf** est un éditeur LaTeX en ligne, collaboratif et en temps réel que nous utilisons pour la rédaction de nos rapports.

— **Clockify** est un site web nous permettant d'estimer le temps passé sur le projet collectivement et individuellement.

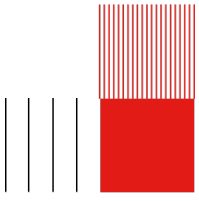
— **Git** et **GitHub** nous servent à mettre en commun le code développé.

Durant les premières semaines, M. Avoine nous a fourni du matériel (différents types de lecteurs et de cartes), ainsi que des sujets de travaux pratiques à réaliser. Cela nous a permis de comprendre le fonctionnement des cartes à puces et prendre en main les outils associés comme CardPeek.

7.2 Répartition des rôles et tâches

Pour réussir au mieux ce projet et éviter les quiproquos, nous avons défini et nous sommes répartis les rôles suivants :

Responsable du temps	Gère Clockify durant les réunions avec les encadrants et entre nous, pour estimer le temps passé.
Gérant communication	Echange avec les encadrants et envoie l'ordre du jour de chaque réunion à l'avance.
Animateur	Anime les réunions, en reprenant les points-clés de l'ordre du jour.
Responsable planification	Utilise MS Project pour planifier le projet en définissant des dates clés (ce rôle prendra effet lors de l'adoption de MS Project).
Rapporteurs (2)	Prennent des notes lors des réunions et les retranscrivent sur Notion.



Pour l'aspect technique du projet, nous nous sommes divisés en plusieurs équipes et nous y avons réparti les tâches.

Equipe EMV : 3 personnes étudient le standard EMV, repèrent les données utilisables et travaillent sur l'extraction de ces dernières.

Equipe VeraCrypt : 4 personnes étudient le code source de VeraCrypt afin d'en comprendre les mécanismes et identifier où/comment intégrer l'utilisation des données extraites par l'équipe EMV, implémentent notre fonctionnalité notamment en modifiant l'interface graphique.

Lorsque des étapes importantes sont franchies (POC fonctionnel, avancement majeur dans la compréhension d'un principe, etc), nous organisons une réunion entre nous (en présentiel ou sur Discord) pour mettre au point l'avancement du projet, discuter des éventuelles difficultés rencontrées, et redéfinir les prochains objectifs.

Enfin, nous nous sommes répartis l'écriture du rapport. Un responsable supervisant la rédaction du rapport a été choisi : il assure la structure du rapport, la répartition des parties entre les membres et la relecture. Les autres membres du groupe se sont vu attribuer chacun la rédaction de plusieurs parties du rapport.

7.3 Entrevue avec Mounir Idrassi

Mercredi 9 Novembre 2022, nous avons eu l'occasion de rencontrer Mounir Idrassi lors d'une réunion en distanciel via l'outil `Zoom`, notre équipe et nos encadrants étant réunis dans une salle du département informatique de l'INSA Rennes. Pour rappel, M. Idrassi est le principal développeur du logiciel VeraCrypt.

Cet entretien d'une heure et demie a été rendu possible grâce à nos encadrants M. Avoine et M. Dupas. Ces derniers étant déjà entrés en contact avec M. Idrassi, ils nous ont fixé cette entrevue. Toutefois la planification de cette réunion a été laissée entièrement à charge des étudiants. Au cours des semaines précédant le rendez-vous, nous avons listé les questions et points techniques à aborder avec M. Idrassi pour qu'il nous éclaire sur le fonctionnement de VeraCrypt. Par ailleurs nous avons planifié une courte présentation des étudiants et des objectifs du projet ainsi que de son avancement, pour nous introduire avant lesdites questions.

Nos différents points de questionnement portaient sur certaines parties du logiciel et son fonctionnement. Les quatre principaux thèmes sur lesquels nos questions se sont concentrées sont :

Structure d'un volume Trois champs de l'entête sont décrits comme réservés dans la section 2.1. Cela est soit dû à des raisons de rétrocompatibilité avec la première version du format de l'entête, soit dans le but de garder la possibilité d'y stocker de nouveaux champs lors de la création d'une nouvelle version du format de l'entête dans le futur.

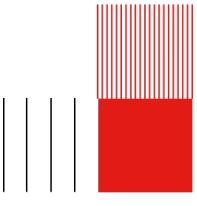
Code Originellement, VeraCrypt (TrueCrypt) a été écrit en C pour Windows, mais l'ajout de nouveau code et fonctionnalités (notamment en C++) au cours du temps a été fait sans refactorisation du code. Ainsi, aujourd'hui, le code du logiciel est très hétérogène (code inutile en fonction de l'OS, code legacy etc.), ce qui complexifie sa compréhension. M. Idrassi nous a néanmoins communiqué qu'un étudiant allemand travaillait sur sa restructuration, et que dans le cadre de notre projet, nous devrions écrire du code en C++ selon les conventions les plus récentes.

Cryptographie Similairement, l'algorithme de hachage RIPEMD-160 est proposé lors du montage d'un volume dans une optique de rétrocompatibilité, mais pas lors de la création d'un volume pour cause d'obsolescence. Concernant le mode de chiffrement XTS, il n'est pas combiné avec CTS [29] car le système d'exploitation se charge d'envoyer à VeraCrypt un clair dont la taille est multiple de 512 octets, rendant ainsi inutile l'utilisation de CTS.

UX M. Idrassi nous a rappelé l'importance de bien définir l'UX (*User eXperience*) finale souhaitée pour notre projet, et d'en déduire un workflow efficace : Quels choix seront laissés aux utilisateurs ? Lesquels seront transparents et automatisés ? Le-dit workflow sera explicité dans un futur rapport.

En fin d'entrevue, il a été évoqué la possibilité de tenir une nouvelle réunion avec M. Idrassi, vers la mi-janvier, pour discuter des avancements effectués et de la direction prise par le projet.

29. CipherText Stealing est un mécanisme permettant de produire un chiffré de même taille que le clair sans ajouter de bourrage, malgré le fait que la taille de ce clair ne soit pas un multiple de la taille d'un bloc requise par l'algorithme de chiffrement.



7.4 Analyse et Gestion du risque

Nous avons identifié certains risques organisationnels et techniques auquels notre groupe pourrait être confronté durant le projet. Nous avons estimé leur probabilité d'avoir lieu et leur gravité s'ils venaient à apparaître sur une échelle de 1 à 5 (5 étant très probable / gravissime). Nous avons aussi défini des mesures préventives et curatives à appliquer pour chaque risque.

Table 7 – Table d'analyse des riques

Description	Probabilité	Gravité	Prévention	Réaction
Avis divergents sur des décisions techniques ou organisationnelles menant à des tensions internes dans le groupe.	4	3	Définir les rôles et responsabilités de chacun. Utiliser les outils et l'organisation définis.	Rester dans le dialogue et trouver une solution à l'amiable.
Matériel fourni défaillant empêchant la réalisation de certaines tâches.	2	2	Manipulation et stockage conformes aux besoins du matériel.	Réparation ou remplacement du matériel.
Matériel personnel de l'étudiant défaillant empêchant la réalisation de certaines tâches.	2	4	Maintenance régulière et personnelle.	Tout d'abord en parler entre nous puis avec les encadrants si ce n'est pas résolu.
Inégalité de l'investissement dans le projet.	3	4	S'assurer de la motivation de chacun régulièrement.	
Inégalité des charges entre les membres du projet.	2	2	Attribuer les tâches le plus équitablement.	Réattribuer certaines tâches aux membres disponibles.
Difficultés dans la réalisation des tâches attribuées menant à un retard du projet.	3	5	Vérifier l'avancée de chacun dans ses tâches. Le dire si un problème se présente.	Aider à la correction du problème voire réattribuer la tâche.

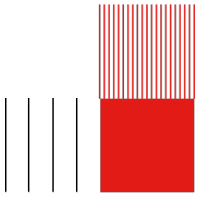
7.5 Faisabilité du projet

La prise en main de CardPeek et de librairies suivant le standard PC/SC nous ont permis de récupérer des données sur des cartes EMV. L'analyse de l'utilisation actuelle des cartes PKCS#11 par VeraCrypt nous a permis de concevoir un mécanisme similaire pour l'utilisation des cartes EMV. La réalisation de deux `POC` nous conforte dans la faisabilité technique du projet.

De ce fait, nous considérons aussi le développement de fonctionnalités secondaires, évoquées par M. Dupas dans le cahier des charges ou par M. Idrassi lors de notre entrevue :

- Prise en charge du sans contact des cartes EMV.
- Émulation d'une carte EMV par une application bancaire sur un téléphone.
- Stockage d'un keyfile sur téléphone, récupérable via NFC ou Bluetooth.
- Séparation de l'entête et du corps du volume.
- Utiliser les fonctions disponibles sur les cartes à puce pour générer de l'aléa.

Dans le cadre de tests, nous avons déjà commencé à nous intéresser au sans contact et réfléchissons aux changements que son implémentation engendredrait dans le code.



7.6 Avancement du projet

En début de projet, nous avons défini quelques grandes étapes :

— **Mi-novembre** : comprendre le standard EMV, et être familiarisé avec la structure du code de VeraCrypt.

— **Mi-décembre** : avoir des POC indépendants entre la lecture de cartes et l'implémentation dans VeraCrypt.

— **Janvier** : faire fonctionner conjointement le travail et POC des 2 équipes EMV et VeraCrypt, pour obtenir un résultat fonctionnel.

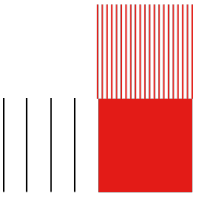
— **Mars** : ajout de l'interface graphique et de l'UX.

— **Avril** : mise au propre et incorporation dans le logiciel officiel.

Comme présenté précédemment, nous sommes actuellement en avance sur ce premier planning prévisionnel. Par ailleurs nous avons listé l'ensemble des taches nécessaires à la réalisation du projet. Cette liste pourra être étandue dans le futur, ou certaines taches sous-divisées.

Table 8 – Liste des taches et leur avancement

Tâches	Avancement
Étude du fonctionnement de VeraCrypt	100%
Étude de la cryptographie dans VeraCrypt	100%
Étude du standard EMV	100%
Choix des données EMV à extraire	100%
Étude du code source de VeraCrypt	80%
Déterminer où intégrer notre fonctionnalité dans le code	100%
Extraction des données des cartes EMV	80%
Démontrer la faisabilité technique du projet	90%
Etude et définition de l'UX	20%
Réaliser une maquette de l'interface graphique	100%
Développement de l'interface graphique pour Unix	0%
Développement de l'interface graphique pour Windows	0%
Avoir une version fonctionnelle du projet sur Unix	40%
Avoir une version fonctionnelle du projet sur Windows	40%
Avoir une version propre du projet incorporable au logiciel officiel	20%
Prise en charge du sans contact	30%
Prise en charge des cartes EMV émulées sur un téléphone	0%
Stockage d'un keyfile sur téléphone, récupérable via NFC ou BT	0%
Séparation de l'entête et du corps du volume	0%
Générer de l'aléa avec des cartes à puces	0%



Conclusion

L'objectif de ce projet est d'ajouter un mécanisme d'authentification à double facteur, basé sur l'utilisation de cartes à puces suivant la norme EMV, pour le chiffrement de volumes virtuels par le logiciel VeraCrypt.

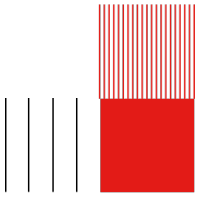
Jusqu'à présent, VeraCrypt permet à ses utilisateurs d'utiliser des keyfiles pour renforcer l'entropie de leurs mots de passe. Ces keyfiles peuvent notamment être stockés sur des cartes PKCS#11, un type de cartes à puces dédié à une utilisation cyber-sécuritaire. Dans certaines situations tendues, posséder une carte PKCS#11 réduit fortement le déni plausible de l'utilisateur. Pour pallier ce problème, l'idée est de permettre l'utilisation d'un type de carte possédé par n'importe qui : les cartes EMV. Suivant la norme du même nom, ce sont des cartes utilisées pour réaliser des opérations bancaires, répandues sur tout le globe. Utiliser comme keyfiles des données internes à la carte EMV de l'utilisateur permettra de renforcer la sécurité de ses données tout en conservant son déni plausible. Cet objectif a été formalisé dans un cahier des charges spécifiant les réalisations techniques requises pour ce projet.

VeraCrypt permet principalement la création de volumes de stockages virtuels chiffrés et peut aller jusqu'à les cacher aux yeux des attaquants. Afin d'accéder à un volume, l'utilisateur doit le monter dans son système de fichiers. Basé sur un fonctionnement complexe, ce montage repose sur l'intégrité de l'entête du volume contenant de nombreuses informations essentielles aux opérations cryptographiques. Afin de réaliser ces opérations, VeraCrypt dispose de multiples mécanismes pour générer de l'aléa et des clés cryptographiques utilisées par un mode de chiffrement dédié aux volumes de données.

L'étude du standard EMV a démontré son hégémonie à l'international. La communication entre un lecteur et une carte EMV est encadrée par l'utilisation de commandes structurées appelées APDU. Ces dernières permettent la connexion aux applications déployées sur la carte EMV pour en lire les données. Les deux certificats Icc et Issuer, ainsi que les données CPLC propres à la carte, ont été sélectionnés pour être utilisés comme keyfiles si l'utilisateur choisit de tirer profit au maximum de sa carte EMV. L'extraction de ces données a été testée et démontrée par la production d'un premier *proof of concept* en Python.

L'analyse du code libre de VeraCrypt a révélé une duplication du code entre Windows et Unix, le principe algorithmique derrière restant le même. L'ajout de l'utilisation de cartes EMV dans le processus de chiffrement de VeraCrypt a été réfléchi pour être analogue à l'utilisation actuelle des cartes PKCS#11. Pour cela, un chemin spécifique à une instance de carte EMV a été développé, et son intégration dans le processus d'application des keyfiles a été définie. L'extraction des données EMV étant possible suite à la réalisation du premier proof of concept, une première utilisation basique de ces données dans VeraCrypt a été développée. Un second proof of concept a donc été codé en C++, utilisant une traduction en langage C du premier et permettant l'utilisation de cartes EMV pour la création et le montage de volumes VeraCrypt. Multiplateforme, cette version exécutable l'est pour le moment en ligne de commande seulement. En parallèle, une maquette de l'interface graphique a été réfléchie, expliquant pas à pas quels changements l'intégration de cette nouvelle fonctionnalité engendrerait sur l'écran de l'utilisateur.

Au-delà de la motivation de chacun, notre organisation au sein de l'équipe est l'un de nos points forts. Nous utilisons plusieurs outils facilitant la communication, que ce soit d'informations ou de données. La définition et la répartition de rôles est apparue comme évidente pour assurer le bon déroulement du projet. De même, la séparation en deux équipes travaillant chacune sur leur aspect du projet nous permet la répartition des charges et d'être conscient des responsabilités de chacun. De plus, nous avons eu l'occasion de rencontrer Mounir Idrassi, principal développeur de VeraCrypt. Nous en avons profité pour échanger sur de nombreux points techniques, mieux comprendre le fonctionnement de VeraCrypt et prendre en considération ses conseils quant au projet. Ce dernier avance bien, la réalisation de deux programmes démontrant sa faisabilité technique nous conforte dans la finalisation du projet dans les temps impartis. Nous allons nous concentrer sur la production d'une version totalement fonctionnelle de VeraCrypt, mais considérons aussi possible la réalisation de plusieurs fonctionnalités secondaires.



Bibliographie

Références

[1] Morris DWORKIN (NIST). *Recommendation for Block Cipher Modes of Operation : the XTS-AES Mode for Confidentiality on Storage Devices*. Jan. 2010.
URL : https://csrc.nist.gov/publications/detail/sp/800-38e/final.

[2] ISO/IEC JTC 1/SC 17. *ISO/IEC 14443-1 : 2018 Cartes et dispositifs de sécurité pour l'identification personnelle — Objets sans contact de proximité*. Avr. 2018.
URL : https://www.iso.org/fr/standard/73596.html.

[3] ISO/IEC JTC 1/SC 17. *ISO/IEC 7816-1 : 2011 Cartes d'identification — Cartes à circuit intégré*. Fév. 2011.
URL : https://www.iso.org/fr/standard/54089.html.

[4] *Article 434-15-2 du code pénal*. Juin 2011.
URL : https://www.legifrance.gouv.fr/codes/article_lc/LEGIARTI000032654251. (Dernière consultation : 20/11/2022).

[5] Jean-Daniel AUSSEL et Ludovic ROUSSEAU. *pyscard*.
URL : https://pyscard.sourceforge.io. (Dernière consultation : 18/11/2022).

[6] *Communiqué relatif à la décision rendue par l'assemblée plénière le 7 novembre 2022 - Pourvoi n° 21-83.146*. Nov. 2022.
URL : https://www.courdecassation.fr/toutes-les-actualites/2022/11/07/code-de-deverrouillage-dun-ecran-de-telephone-et-cryptologie. (Dernière consultation : 20/11/2022).

[7] Florian CRISTINA. *PBKDF2 et génération des clés de chiffrement de disque*. 2009.
URL : http://www.artiflo.net/2009/08/pbkdf2-et-generation-des-cles-de-chiffrement-de-disque/. (Dernière consultation : 18/11/2022).

[8] EMVCO. *EMV Integrated Circuit Card Specifications for Payment Systems Book 3 Application Specification*. Nov. 2011.
URL : https://www.emvco.com/wp-content/uploads/2017/04/EMV_v4.3_Book_3_Application_Specification_20120607062110791.pdf.

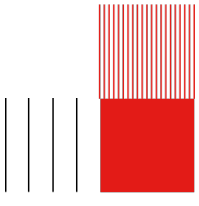
[9] Mounir IDRASSI. *Changing Passwords and Keyfiles*.
URL : https://www.veracrypt.fr/en/Changing%20Passwords%20and%20Keyfiles.html. (Dernière consultation : 18/11/2022).

[10] Mounir IDRASSI. *Encryption Algorithms*.
URL : https://www.veracrypt.fr/en/Encryption%20Algorithms.html. (Dernière consultation : 18/11/2022).

[11] Mounir IDRASSI. *Encryption Scheme*.
URL : https://veracrypt.fr/en/Encryption%20Scheme.html. (Dernière consultation : 18/11/2022).

[12] Mounir IDRASSI. *Hash Algorithms*.
URL : https://www.veracrypt.fr/en/Hash%20Algorithms.html. (Dernière consultation : 18/11/2022).

[13] Mounir IDRASSI. *Header Key Derivation, Salt, and Iteration Count*.
URL : https://www.veracrypt.fr/en/Header%20Key%20Derivation.html. (Dernière consultation : 18/11/2022).



[14] Mounir IDRASSI. *Hidden Volume*.
URL : https://www.veracrypt.fr/en/Hidden%20Volume.html. (Dernière consultation : 18/11/2022).

[15] Mounir IDRASSI. *Keyfiles*.
URL : https://www.veracrypt.fr/en/Keyfiles.html. (Dernière consultation : 18/11/2022).

[16] Mounir IDRASSI. *Random Number Generator*.
URL : https://www.veracrypt.fr/en/Random%20Number%20Generator.html. (Dernière consultation : 18/11/2022).

[17] Mounir IDRASSI. *VeraCrypt*.
URL : https://launchpad.net/veracrypt. (Dernière consultation : 18/11/2022).

[18] Mounir IDRASSI. *VeraCrypt*.
URL : https://sourceforge.net/projects/veracrypt. (Dernière consultation : 18/11/2022).

[19] Mounir IDRASSI. *Volume Format Specification*.
URL : https://www.veracrypt.fr/en/VeraCrypt%20Volume%20Format%20Specification.html. (Dernière consultation : 18/11/2022).

[20] L1L1. *CardPeek*.
URL : https://github.com/L1L1/cardpeek. (Dernière consultation : 18/11/2022).

[21] Andreas POLLER et al. *Security Evaluation of VeraCrypt*. Déc. 2020.
URL : https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Veracrypt/Veracrypt.pdf?__blob=publicationFile&v=1.

[22] QUARKSLAB. *VeraCrypt 1.18 Security Assessment*. 2016.
URL : https://blog.quarkslab.com/resources/2016-10-17-audit-veracrypt/16-08-215-REP-VeraCrypt-sec-assessment.pdf.

[23] *Socle interministériel de logiciels libres - Veracrypt*.
URL : https://sill.etalab.gouv.fr/software?name=VeraCrypt. (Dernière consultation : 21/11/2022).

[24] Hector SUDAN. *Guide de Protection numérique des Sources journalistiques*. 2021.
URL : https://sourcesguard.ch/documents/GuideProtectionSources-20210423.pdf. (Dernière consultation : 18/11/2022).

[25] Andrey TOUMILOVICH. *BER-TLV Package*.
URL : https://github.com/toumilov/python-ber-tlv. (Dernière consultation : 18/11/2022).

[26] Michael WARD. *How EMVCo is Supporting Card Data Encryption Advancements for Card Personalisation*.
URL : https://www.emvco.com/emv_insights_post/how-emvco-is-supporting-card-data-encryption-advancements-for-card-personalisation/. (Dernière consultation : 18/11/2022).