

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

Rapport de conception



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

Département informatique
Projet de 4^{ème} année, 2022-2023



Table des matières

Introduction	2
1 Philosophie et méthodologie adoptées	3
2 Représentation structurelle des keyfiles	4
2.1 Structures existantes pour les keyfiles PKCS#11	4
2.2 Nouvelles structures différenciant les keyfiles des cartes	5
2.3 Construction des chemins	6
3 Classes gérant les structures	7
3.1 Classe gérant les SecurityTokenKeyfile	7
3.2 Classe gérant les EMVTokenKeyfile	8
3.3 Classe gérant les TokenKeyfile	8
4 Extraction des données des cartes EMV	10
4.1 Module pour l'extraction des données EMV	10
4.2 Gestion des erreurs	13
5 Intégration dans les interfaces	14
5.1 Intégration en ligne de commande	14
5.2 Intégration à l'interface graphique	14
5.3 Optionnalité de la fonctionnalité	17
5.4 Gestion des erreurs liées à la librairie PKCS#11	17
6 Tests et mesures de performance de l'application	19
6.1 Tests unitaires et d'intégration	19
6.2 Tests de détection de fuites de mémoires	19
6.3 Tests de performances	19
7 Gestion de projet	20
7.1 Répartition des rôles	20
7.2 Suivi de Mounir Idrassi	20
7.3 Avancement	21
7.4 Planification des tâches	21
Conclusion	22
Bibliographie	23

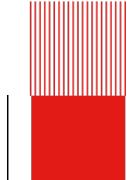
A Diagramme de classe UML de la nouvelle architecture (ajouts en orange)

Table des figures

1 Structures représentant les SecurityTokenKeyfile	4
2 Diagramme UML des nouvelles structures (en orange, les ajouts)	5
3 La classe SecurityToken	7
4 Classe EMVToken	8
5 Classe proxy Token (en orange, les ajouts)	8
6 Diagramme UML du module permettant la communication avec la carte	10
7 Organigramme simplifié de la fonction GetReaders	11
8 Organigramme simplifié de la fonction GettingPAN	11
9 Organigramme simplifié de la fonction GettingAllCerts	12
10 Ajout d'un keyfile stocké sur une carte PKCS#11 à la création d'un volume	15
11 Ajout d'un keyfile stocké sur une carte PKCS#11 à l'ouverture d'un volume	15
12 Interface permettant l'ajout de keyfile depuis une carte PKCS#11	16
13 Interface permettant l'ajout de cartes EMV comme keyfile	17
14 Diagramme de cas d'utilisation	18

Liste des tableaux

1 Liste des tâches et leur avancement	21
---	----



Introduction

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 de ce projet est d'ajouter un mécanisme d'authentification à double facteur, basé sur l'utilisation de cartes à puce suivant la norme EMV¹, pour renforcer la sécurité du mot de passe ainsi que la propriété de déni plausible de l'utilisateur.

Jusqu'à présent, VeraCrypt permet à ses utilisateurs d'utiliser des keyfiles pour renforcer l'entropie de leurs mots de passe. Ces keyfiles sont des fichiers additionnels optionnels n'ayant pas l'obligation d'être spécialement créés dans cet objectif, pouvant être de n'importe quel type et stockables sur n'importe quel support physique. Ils peuvent notamment être stockés sur des cartes PKCS#11², un type de cartes à puce dédié à une utilisation cyber-sécuritaire. Dans certaines situations tendues, posséder une telle carte réduit donc fortement le déni plausible de l'utilisateur. Pour pallier ce problème, l'idée est de permettre l'utilisation d'un type de carte à puce possédé par n'importe qui : les cartes EMV. Suivant la norme du même nom, ce sont des cartes utilisées pour réaliser des opérations bancaires, extrêmement répandues à l'international. Utiliser comme keyfiles des données internes de la carte EMV de l'utilisateur permettra ainsi de renforcer la sécurité de ses données tout en conservant son déni plausible. Cet objectif a été formalisé par Jules Dupas (co-encadrant avec Gildas Avoine, il joue aussi le rôle de client dans le cadre de ce projet) dans un cahier des charges spécifiant les réalisations techniques requises pour ce projet.

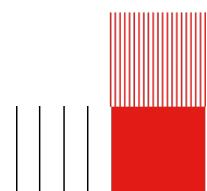
L'étude du standard EMV a permis de comprendre comment extraire les données d'une carte EMV et de déterminer celles utilisables comme keyfiles. L'analyse du code libre de VeraCrypt a permis de comprendre l'implémentation actuelle de l'utilisation des cartes PKCS#11 comme sources de keyfiles et ainsi déterminer à quel niveau du processus intégrer le traitement des cartes EMV, que ce soit dans l'algorithmique comme dans l'interface utilisateur. Les résultats de ces études ont été présentés dans un rapport dédié, tandis que la faisabilité technique du projet a été démontrée à travers la production de deux *proofs of concept*.

Ce rapport-ci, réalisé durant la phase de conception logicielle du projet, décrit l'architecture interne du logiciel VeraCrypt telle qu'elle est actuellement, notamment grâce à sa modélisation en UML et une description précise de l'interfaçage des différents modules la composant. Les modifications du code source apportées par ce projet d'ajout de fonctionnalité y sont bien entendu présentées, suivant, elles aussi, les spécifications évoquées. Ce rapport a pour objectif parallèle de décrire à Mounir Idrassi³, les modifications apportées au code source de son logiciel. Ceci explique le fond de ce rapport qui peut s'avérer plutôt détaillé techniquement. Suite à une réduction conséquente du temps disponible pour livrer notre produit, et cela, après la planification des étapes à réaliser dans ce projet, une nouvelle planification des tâches à effectuer est présentée en fin de rapport.

1. Standard technique pour les cartes de paiement et les terminaux, abréviation de "Europay, Mastercard et Visa" (les trois entreprises à l'origine de ce standard) il est actuellement managé par le consortium EMVCo.

2. Public-Key Cryptography Standard #11 est une API définissant une interface générique pour les périphériques cryptographiques.

3. Développeur principal de VeraCrypt.



1 Philosophie et méthodologie adoptées

Avant de développer la partie technique et conception de ce rapport, il est important de décrire l'approche adoptée quant à l'intégration de notre fonctionnalité au logiciel. L'objectif de ce projet est en effet de modifier le code source d'une application existante. Cette dernière est utilisée dans le monde entier par plusieurs centaines de milliers de personnes. Plusieurs éléments sont par conséquent à prendre en compte pour favoriser l'incorporation de notre code dans une version Bêta de VeraCrypt. Au cours de plusieurs mois, notre fonctionnalité pourra alors être testée par les utilisateurs à travers cette Bêta et l'acceptation définitive de cet ajout par Mounir Idrassi dépendra directement de la qualité de nos modifications.

Tout d'abord, le code de VeraCrypt a déjà fait l'objet de plusieurs audits de sécurité⁴. Ayant pour but le chiffrement de données, c'est un critère qui demeure essentiel. Pour cette raison, notre code devra suivre la même rigueur. Il ne faut surtout pas introduire de failles de sécurité lors du développement de notre fonctionnalité. Par exemple, il faudra assurer à l'utilisateur que les données sensibles de sa carte EMV manipulées comme keyfiles seront supprimées de la mémoire vie de la machine après utilisation.

Notre code devra s'intégrer le plus naturellement possible au code source de VeraCrypt. Le but est de modifier le moins possible le code d'origine pour limiter les problèmes techniques tout en collant avec le comportement actuel pour les cartes à puce de type PKCS#11. Minimiser les modifications au sein du logiciel rassure et facilite sa relecture par les autres développeurs, notamment M. Idrassi, ce qui renforce sa possible acceptation dans une version Bêta.

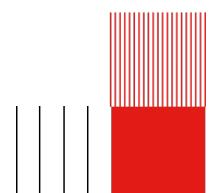
Enfin, la nouvelle fonctionnalité doit aussi être adoptée par les utilisateurs actuels de VeraCrypt. Cette mesure est particulièrement importante dans le cadre de l'interface graphique. Il est important de faire savoir à l'utilisateur qu'il est toujours maître de ce qu'il fait avec le logiciel. Il pourra choisir ou non d'utiliser cette nouvelle fonctionnalité liée aux cartes de type EMV. De plus, la manière d'utiliser des keyfiles sur une carte à puce devra rester simple, quel que soit le cas d'usage (une ou plusieurs cartes, de même ou de types différents). Il est même possible d'imaginer à terme plusieurs modes d'utilisation de VeraCrypt : un mode simplifié et un mode expert, permettant de définir des paramètres avancés.

Les réunions avec Mounir Idrassi permettent de connaître son avis concernant les différents moyens de respecter les points cités précédemment et d'avoir un retour sur l'implémentation de la fonctionnalité au fur et à mesure. Plusieurs de ses suggestions sont à l'origine de mesures décrites dans ce rapport.

Comme la fonctionnalité vise à être intégrée à un logiciel open-source comme VeraCrypt, il est aussi important de s'assurer que le code soit facile à maintenir et à modifier, en veillant à ce qu'il soit bien documenté et structuré de manière claire et cohérente. Ainsi, il sera plus facile pour d'autres développeurs de comprendre et de maintenir le code, ce qui accélérera l'évolution du projet en général. De plus, une documentation complète peut aider à promouvoir l'adoption de la fonctionnalité par d'autres développeurs et utilisateurs. C'est pourquoi il sera nécessaire de modifier la documentation de VeraCrypt déjà existante afin de rajouter les pages concernant l'authentification par carte EMV. Elle devra respecter le style du reste du document et devra être écrite en anglais. Cette documentation fait partie inhérente du projet puisqu'elle sera livrée avec le code à M. Idrassi en vue d'une possible intégration au projet officiel.

Ce projet s'inscrit donc dans une optique particulière de développement logiciel.

4. Audits réalisés par QuarksLab [5] en 2016 et par la BSI [4] en 2020



2 Représentation structurelle des keyfiles

Ci-dessous sont présentées les structures, existantes puis développées pour le projet, utilisées pour manipuler les keyfiles stockés sur des cartes à puce ainsi que leur représentation sous la forme de chemins pour faciliter leur affichage et leur application.

2.1 Structures existantes pour les keyfiles PKCS#11

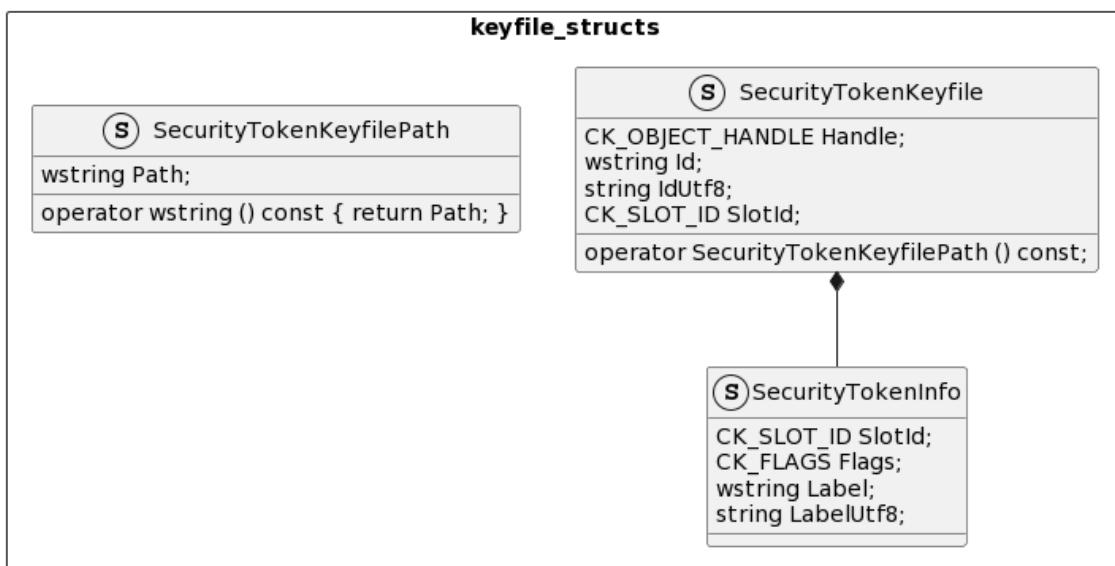


Figure 1 – Structures représentant les SecurityTokenKeyfile

Dans la version actuelle de VeraCrypt, les keyfiles provenant de cartes PKCS#11 sont gérés et stockés grâce à trois différentes structures modélisées en Figure 1, qui sont :

- **SecurityTokenInfo** : représente une carte PKCS#11 connectée à l'ordinateur, où SlotId identifie le lecteur utilisé.
- **SecurityTokenKeyfile** : représente le keyfile concerné, où Id est le nom du fichier sur la carte, Token de type SecurityTokenInfo la carte PKCS#11 source et SlotId l'identifiant du lecteur. L'opérateur SecurityTokenKeyfilePath () cast en SecurityTokenKeyfilePath.
- **SecurityTokenKeyfilePath** : représente un SecurityTokenKeyfile sous la forme d'un chemin Path, utile dans le cadre de son affichage en ligne de commande et dans l'interface graphique, mais aussi dans le cadre de son application pour l'ouverture ou la création d'un volume.

2.2 Nouvelles structures différenciant les keyfiles des cartes

Pour pouvoir stocker les nouveaux keyfiles provenant de cartes EMV, de nouvelles structures ont été pensées et une réorganisation se basant sur l'héritage des structures en C++ a été mise en place (Figure 2). Tout d'abord, dans le rectangle `token_keyfile`, trois structures serviront de bases génériques pour stocker tout type de keyfile provenant de cartes à puce. Parmi ces structures, deux d'entre elles seront dérivées pour correspondre respectivement aux types PKCS#11 et EMV. Les attributs visibles dans ces structures génériques sont communs aux deux types de cartes qui seront disponibles. Les rectangles colorés correspondent à l'ensemble des ajouts réalisés pour ce projet par rapport à la version d'origine.

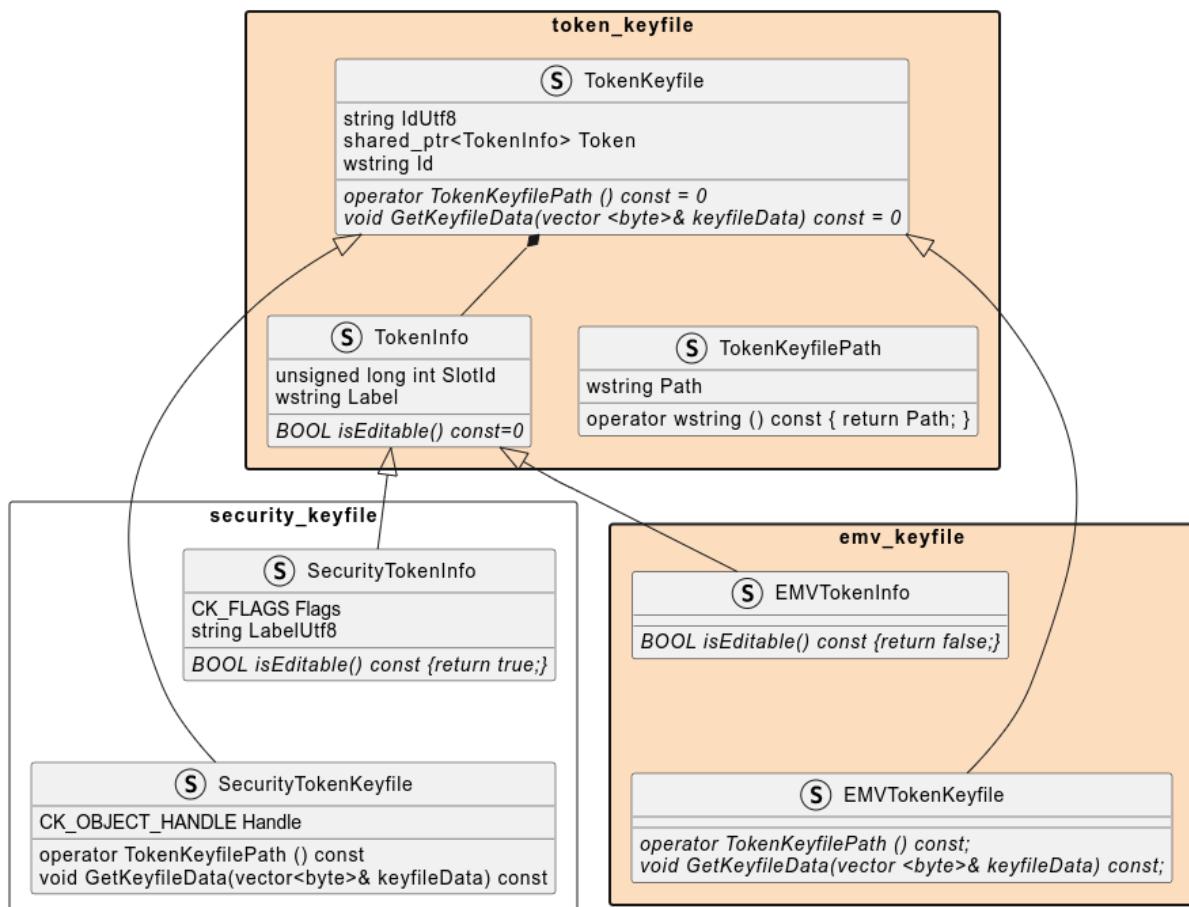


Figure 2 – Diagramme UML des nouvelles structures (en orange, les ajouts)

Les structures mères utilisées pour regrouper les différents types de keyfiles et pouvoir itérer sur ceux-ci sont les suivantes :

- **TokenInfo** : représente une carte à puce de n'importe quel type.
- **TokenKeyfile** : représente un keyfile provenant d'une carte à puce. Cette classe possède les attributs communs à tout type de keyfile provenant d'une carte à puce.
- **TokenKeyfilePath** : représente un TokenKeyfile sous la forme d'un chemin Path.

De nouvelles structures associées les cartes EMV ont été développées :

- **EMVTokenInfo** : représente une carte à puce de type EMV, dispose d'une fonction booléenne `isEditable()` retournant faux par défaut, indiquant que la carte n'est pas disponible en écriture à la différence des cartes PKCS#11 et par conséquent désactivant certaines options (voir section 5.2). Le choix d'implémenter une simple fonction booléenne plutôt que caster pour vérifier le type de la structure s'explique par le fait que les cast sont relativement coûteux et complexiferaient le code si d'autres structures devaient être ajoutées à l'avenir.
- **EMVTokenKeyfile** : représente un keyfile EMV, dispose d'un opérateur de cast `TokenKeyfilePath ()` redéfini pour créer le chemin décrit en partie 2.3 et également de la fonction `GetKeyfilePathData()` extrayant les données de la carte EMV en appelant les méthodes de la classe `ICCExtractor` (voir section 4).

Pour coller à cette nouvelle architecture, les anciennes structures **SecurityTokenInfo**, **SecurityTokenKeyfile** et **SecurityTokenKeyfilePath** ont dû être modifiées, voire supprimées. Tout d'abord, la structure `SecurityTokenKeyfilePath` n'existe plus, `TokenKeyfilePath` remplaçant déjà ce rôle. Ainsi, il n'existe pas non plus de structure `EMVTokenKeyfilePath`. Les structures `SecurityTokenInfo` et `SecurityTokenKeyfile` sont très similaires à leurs versions antérieures, au détail près que certains attributs ou méthodes ont été remontés dans les structures parentes `TokenKeyfile` et `TokenInfo`. Dans `SecurityTokenInfo`, la fonction booléenne `isEditable()` rend vrai par défaut, les cartes de type PKCS#11 étant disponibles en lecture comme en écriture.

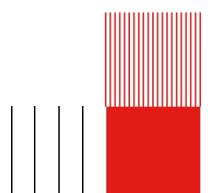
2.3 Construction des chemins

Le seul moyen de créer un `TokenKeyfilePath` et par la même occasion de différencier les chemins keyfiles PKCS#11 et EMV est d'utiliser l'opérateur `TokenKeyfilePath ()` de la classe `TokenKeyfile`. Cet opérateur est redéfini dans les classes dérivées `SecurityTokenKeyfile` et `EMVTokenKeyfile`. Pour rappel, la structure `TokenKeyfilePath` contient un unique attribut `wstring Path`, représentant ledit chemin.

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.

De manière similaire au chemin des fichiers des cartes PKCS#11, les cartes EMV se voient à présent 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'a pas besoin de choisir un fichier en particulier sur la carte.

Ces chemins sont utilisés par les fichiers `Keyfiles.cpp` et `Keyfile.c` (respectivement Linux et Windows) pour identifier les keyfiles et les appliquer lors de la création d'un volume ou bien l'ouverture d'un volume déjà créé auparavant.



3 Classes gérant les structures

Afin de manipuler les structures présentées dans la section précédente, des classes sont chargées de la communication avec les cartes à puce et de l'extraction des keyfiles. De manière similaire, de nouvelles classes ont été implémentées pour permettre l'ajout des cartes EMV.

3.1 Classe gérant les SecurityTokenKeyfile

```
SecurityToken
map <CK SLOT ID, Pkcs11Session> Sessions;
...
...
void void InitLibrary(...);
void CloseLibrary();
void CheckLibraryStatus()

void Login (CK_SLOT_ID slotId, const char* pin);
void LoginUserIfRequired (CK_SLOT_ID slotId);
void OpenSession (CK_SLOT_ID slotId);
void CloseSession (CK_SLOT_ID slotId)
void CloseAllSessions()

void CreateKeyfile(CK_SLOT_ID slotId, vector <byte> &keyfileData, const string &name)
void DeleteKeyfile(const SecurityTokenKeyfile &keyfile)

list <SecurityTokenInfo> GetAvailableTokens()
SecurityTokenInfo GetTokenInfo (CK_SLOT_ID slotId);
vector <SecurityTokenKeyfile> GetAvailableKeyfiles(...)
bool IsKeyfilePathValid(const wstring& securityTokenKeyfilePath)
void GetKeyfileData(const SecurityTokenKeyfile &keyfile, vector <byte> &keyfileData)
...
```

Figure 3 – La classe SecurityToken

La classe responsable de l'extraction et du stockage des SecurityTokenKeyfile est la classe `SecurityToken` (Figure 3), dont les méthodes sont regroupables en quatre différents ensembles avec leurs objectifs propres :

- **Gérer la librairie PKCS#11** : permettent d'initialiser (`InitLibrary`), fermer (`CloseLibrary`) et vérifier (`CheckLibraryStatus`) la librairie PKCS#11 utilisée.
- **Administrer l'authentification auprès des cartes** : les cartes PKCS#11 authentifiant l'utilisateur à l'aide d'un code PIN, des fonctions (`Login`, `Open` & `Close Session`) permettent donc de gérer les sessions authentifiées auprès des cartes, stockées dans l'attribut `Sessions`.
- **Modifier le stockage des keyfiles sur les cartes** : les cartes PKCS#11 stockent des keyfiles, il est donc possible d'en importer (`CreateKeyfile`) ou d'en supprimer (`DeleteKeyfile`).
- **Récupération des keyfiles et informations complémentaires** : ces dernières fonctions permettent de récupérer les informations sur les cartes PKCS#11 (`GetTokenInfo` & `GetAvailableTokens`), sur les keyfiles (`GetAvailableKeyfiles`) pour les stocker sous la forme des structures décrites partie 2.2 et enfin, dans le cadre de l'application de ces keyfiles, de vérifier leurs chemins sur les cartes (`IsKeyfilePathValid`) et d'extraire leurs données (`GetKeyfileData`).

Les trois premières familles de fonctions sont propres à l'utilisation des cartes de type PKCS#11. La dernière famille concerne l'extraction des données et sera donc adaptée par la nouvelle classe responsable des cartes de type EMV.

3.2 Classe gérant les EMVTokenKeyfile

Une nouvelle classe statique a été créée : EMVToken, modélisée en figure 4. Responsable de l'extraction des données des cartes EMV et de leur stockage dans une instance de EMVTokenKeyfile, son comportement et ses méthodes sont similaires à celui de la classe SecurityToken.



Figure 4 – Classe EMVToken

Les fonctions statiques `IsKeyfilePathValid()`, `GetAvailableKeyfiles()` et `GetTokenInfo()` permettent respectivement de vérifier qu'un chemin vers une carte EMV est valide (cf partie 2.3), de récupérer un keyfile, et de récupérer des informations d'un token EMV; notamment via les méthodes de la classe `IccDataExtractor` (voir section 4). Néanmoins, de par les différences entre PKCS#11 et EMV, de nombreuses fonctions présentes dans `SecurityToken` n'ont pas d'équivalent de `EMVToken`. À titre d'exemple, pour lire une carte PKCS#11, il est nécessaire de rentrer un code PIN pour ouvrir une session⁵, ce qui n'est pas le cas pour lire une carte EMV.

3.3 Classe gérant les TokenKeyfile

Dans le but d'unifier et de simplifier l'utilisation des classes `EMVToken` et `SecurityToken`, la classe parent `Token` a été créée (Figure 5 ci-dessous). Elle correspond au patron de conception proxy : son rôle est de réunir tous les appels provenant d'autres classes dans le projet, et gérer les appels vers `EMVToken` et `SecurityToken`.

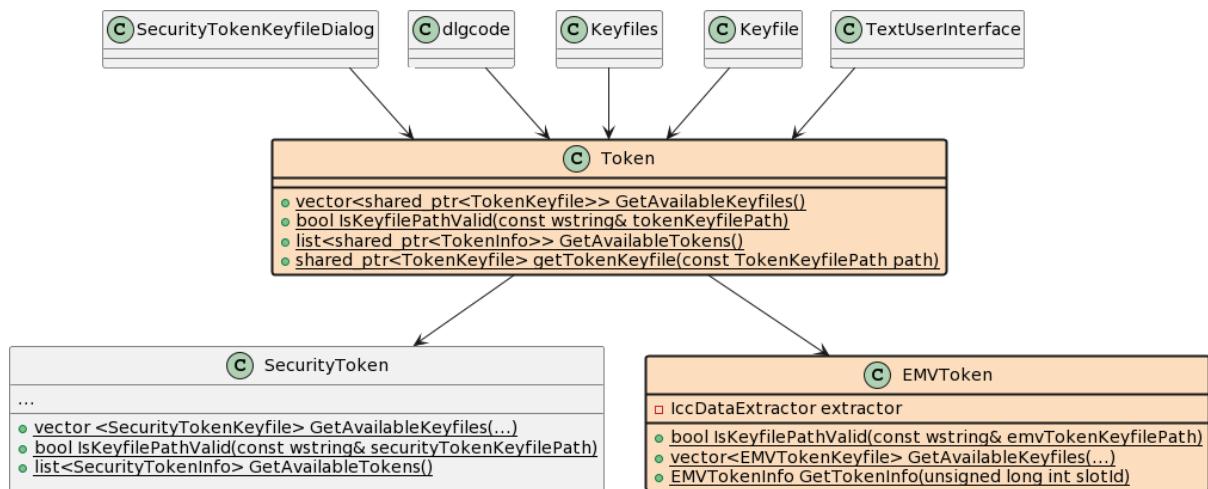
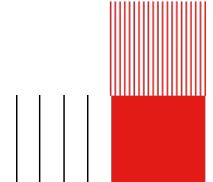


Figure 5 – Classe proxy Token (en orange, les ajouts)

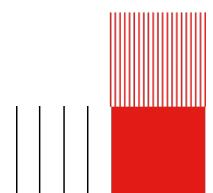
5. Géré par les fonctions `Login()`, `OpenSession()`, et `CloseSession()` dans `SecurityToken`



La classe Token contient donc, similairement à SecurityToken et EMVToken, les fonctions statiques IsKeyfilePathValid(), GetAvailableKeyfiles(), GetAvailableTokens(), et GetTokenKeyfile(). Au travers de ces fonctions, Token fait appel aux fonctions statiques de SecurityToken ou EMVToken présentées précédemment pour récupérer les données adéquates et les renvoyer aux classes supérieures. Ces-dites classes supérieures (comme dlgcode, Keyfile, ou Keyfiles) peuvent correspondre aux classes gérant l'interface graphique par exemple. Il est à noter qu'il n'existe aucun lien d'héritage entre Token, EMVToken, et SecurityToken.

L'utilisation d'une classe proxy comme Token présente 3 avantages majeurs :

1. **Faciliter l'ajout futur de nouvelles cartes** : si à l'avenir de nouvelles cartes à puce similaires à EMV et PKCS#11 devraient être ajoutées, il suffirait de créer une classe statique et les structures des keyfiles correspondantes (ainsi que modifier en adéquation Token), sans avoir à se soucier des appels supérieurs.
2. **Gérer tous les types de cartes sans distinction** : inversement depuis le reste du programme, cela permet de n'avoir à faire qu'à une seule classe, la classe Token.
3. **Une meilleure relecture** : rajouter ce niveau d'abstraction permet une meilleure maintenabilité du code et facilite sa relecture.



4 Extraction des données des cartes EMV

4.1 Module pour l'extraction des données EMV

Afin que la classe `EMVToken` puisse récupérer les différentes informations sur la carte EMV, utilisées pour la création et la gestion de keyfiles, la classe `IccDataExtractor` ainsi que la classe utilitaire `TLVParser` ont été créées (voir Figure 6). Cette dernière est une implémentation d'un parseur pour le BER-TLV⁶, permettant de limiter l'utilisation de librairies externes et donc l'ajout de dépendances au projet. Les deux classes ont été conçues de manière à constituer un module qui peut être utilisé indépendamment du reste de l'application, facilitant ainsi les tests.

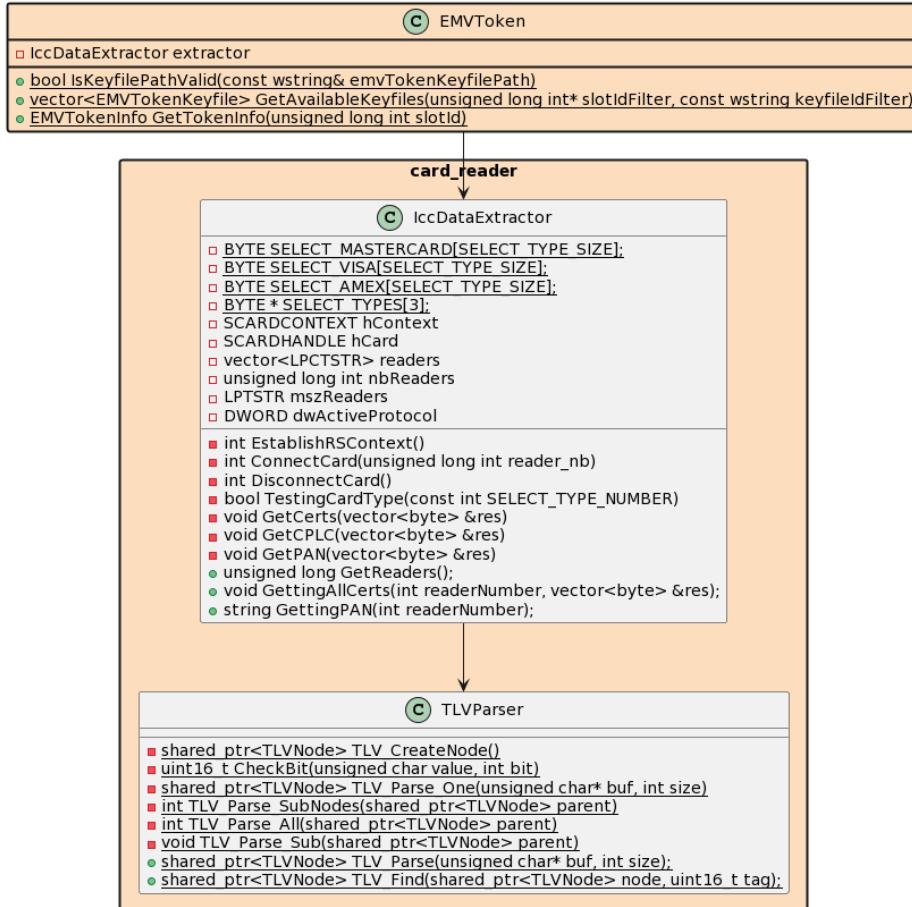
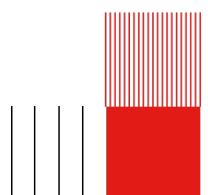


Figure 6 – Diagramme UML du module permettant la communication avec la carte

`IccDataExtractor` (voir Figure 6) s'occupe de toute la partie communication avec la carte EMV en lui envoyant différentes APDUs grâce à la librairie `winscard` [7]. Elle dispose de quelques fonctions principales publiques, appelant d'autres fonctions privées correspondant aux étapes intermédiaires de l'échange avec la carte. Cette architecture permet de faciliter l'utilisation du module au sein de l'application sans avoir à se soucier du fonctionnement sous-jacent de la communication avec la carte. Ainsi, les fonctions principales sont les suivantes :

- **unsigned long GetReaders()** : récupère la liste des lecteurs de carte à puce connectés à la machine de l'utilisateur, de les stocker dans le vecteur (`readers`) afin d'y avoir accès par la suite et enfin de renvoyer le nombre total de lecteurs disponibles (voir Figure 7).

6. Basic Encoding Rules - Type-Length-Value, un format d'encodage défini par le standard ASN.1.



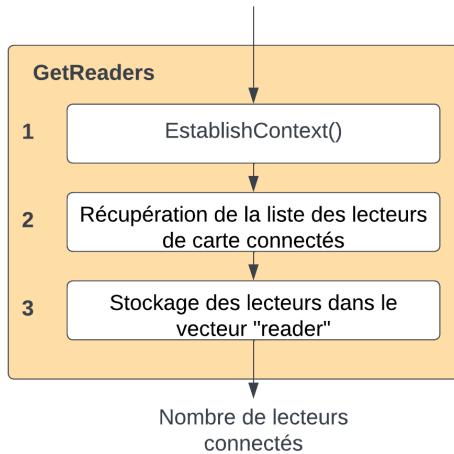


Figure 7 – Organigramme simplifié de la fonction GetReaders

`EstablishContext` (voir Figure 7, étape 1) établit le contexte du gestionnaire de ressources, qui gère l'accès aux lecteurs et aux cartes à puce. Le contexte du gestionnaire de ressources est principalement utilisé par les fonctions de requête et de gestion lors de l'accès à la base de données de la carte. La portée du contexte du gestionnaire de ressources sera celle du système. Cela signifie que l'application peut accéder à tous les lecteurs de carte à puce connectés au système et pas seulement ceux associés à l'utilisateur actuellement connecté. Les lecteurs sont ensuite récupérés et stockés dans le vecteur `readers` (voir Figure 7, étapes 2 et 3).

- **string GettingPAN(int readerNumber)** : vérifie si la carte dans le lecteur numéro `readerNumber` est bien de type EMV. Le cas échéant, récupère et retourne le numéro (PAN⁷) de la carte sous la forme d'une chaîne de caractères (voir Figure 8).

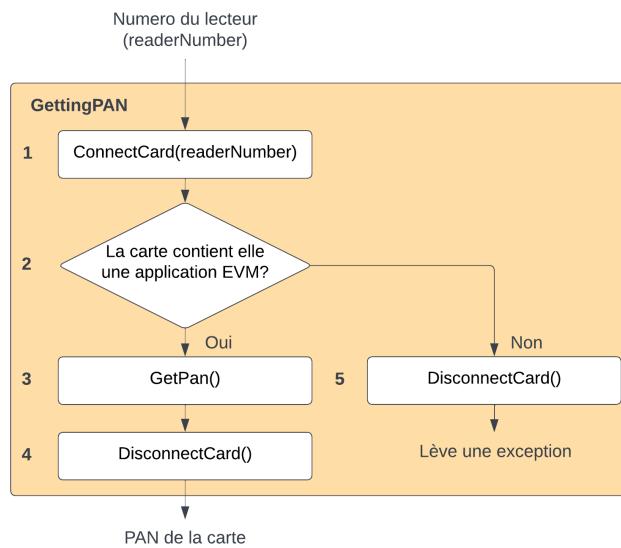
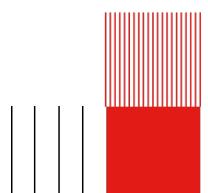


Figure 8 – Organigramme simplifié de la fonction GettingPAN

7. Primary Account Number



La fonction va tout d'abord établir une connexion entre l'application et la carte contenue par le lecteur spécifié par `readerNumber` (voir Figure 8, étape 1), ce dernier correspondant à l'indice du lecteur dans le vecteur `readers` vu précédemment.

Une fois la carte connectée, la fonction va tester itérativement si la carte est de type EMV (si elle contient au moins une application Mastercard, Visa ou American Express) (voir Figure 8, étape 2) grâce à la fonction `TestingCardType` (voir Figure 6). Pour ce faire, cette fonction va tenter de sélectionner l'application sur la carte en lui envoyant l'APDU [SELECT FILE + AID]. Si l'application existe, la carte renverra le code de réponse 0x61 signifiant que la commande a bien été exécutée. Cette action sera donc effectuée successivement pour chaque AID jusqu'à obtenir ce code de réponse.

L'application étant sélectionnée, `GetPAN` (voir Figure 8, étape 3) va s'occuper de la récupération du PAN de la carte. Il est possible de récupérer directement cette donnée, mais l'APDU prévue à cet effet est différente pour chaque type de carte. Afin de faciliter l'ajout potentiel de nouveaux types de cartes, il a été choisi de faire un compromis en termes de performance en implémentant une solution plus généraliste. Comme il n'est pas possible de connaître le contenu des fichiers de l'application à l'avance, il est nécessaire d'itérer sur chaque fichier de la carte. Pour lire chacun d'entre eux, l'APDU [READ RECORD + Identifiant du fichier] est envoyée à la carte. Une fois les données du fichier récupérées, elles doivent être analysées grâce à la fonction `Parse` de la classe `TLVParser` (voir Figure 6). La fonction `Find` (voir Figure 6) cherche dans l'arborescence du BER-TLV les données associées au tag 0x5A correspondant au PAN.

- **void GettingAllCerts(int readerNumber, vector<byte> &res)** : récupère le premier ICC public key certificate, le premier Issuer public key certificate ainsi que les données CPLC sur la carte dans le lecteur numéro `readerNumber`. Ces données sont concaténées et retournées dans le vecteur d'octets `res` (voir Figure 9).

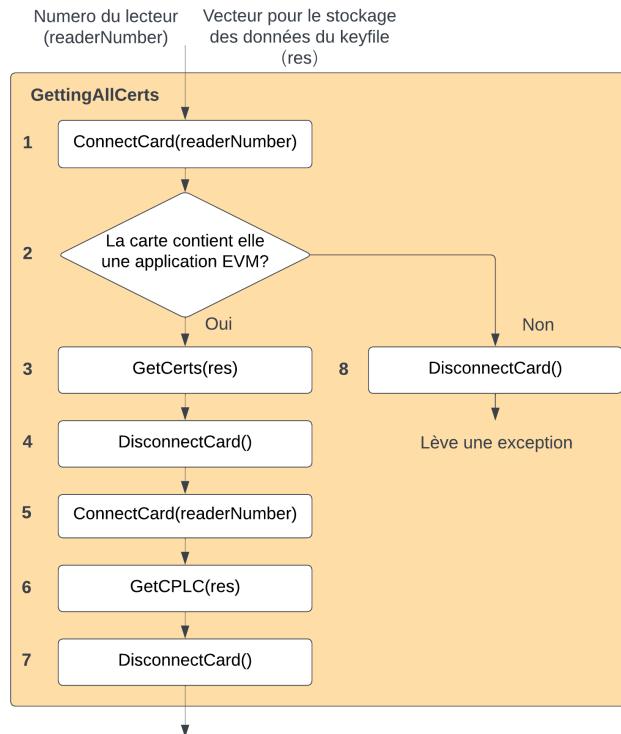
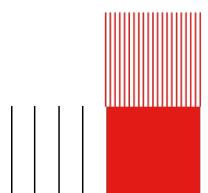


Figure 9 – Organigramme simplifié de la fonction GettingAllCerts



Tout comme GettingPAN, GettingAllCerts établit la connexion avec la carte et vérifie qu'elle est bien de type EMV (voir Figure 9, étapes 1 et 2). Ces actions permettent d'utiliser la fonction telle quelle sans avoir à appeler GettingPAN au préalable.

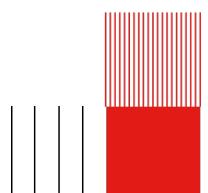
La fonction de récupération des certificats GetCerts (voir Figure 9, étape 3) fonctionne de façon similaire à GetPAN vu précédemment, en utilisant le tag 0x9F46 (resp. 0x90) pour l'ICC Public Key Certificate (resp. l'Issuer Public Key Certificate).

Pour la récupération des données CPLC (voir Figure 9, étape 6), il n'est pas nécessaire d'avoir sélectionné d'application en particulier. En effet, il est possible de récupérer directement ces données grâce à l'APDU 0x80, 0xCA permettant de récupérer les informations sur les capacités et la configuration de la carte. Il n'est pas non plus nécessaire de parser le BER-TLV puisque les données CPLC ne sont pas encodées dans ce format. L'avantage des CPLC réside donc dans le fait que leur récupération est plus rapide que les certificats.

Les étapes de déconnexion/reconnexion de la carte (voir Figure 9, étapes 4 et 5) sont obligatoires puisqu'il faut sortir de l'application sélectionnée lors des étapes 2 et 3 afin de récupérer les données CPLC.

4.2 Gestion des erreurs

Actuellement, de nombreuses exceptions peuvent être levées dans le module en charge de récupérer les données EMV. Cependant, en l'état actuel, ces erreurs ne sont pas encore traitées de manière adéquate. En effet, elles ne sont pas communiquées à l'utilisateur et certaines d'entre elles ferment tout bonnement l'application. Il sera donc important de travailler sur la gestion des erreurs pour éviter les échecs inattendus et garantir un fonctionnement fiable du système. Cela inclura la définition de codes pour les différents types d'erreurs, la gestion des exceptions et la notification de l'utilisateur, que ce soit par l'intermédiaire du terminal ou de l'interface graphique. Pour ce dernier point, il conviendra de produire une implémentation similaire à ce qu'il se fait déjà dans le reste de l'application, permettant ainsi de maintenir une cohérence visuelle et fonctionnelle pour l'utilisateur.



5 Intégration dans les interfaces

L'intégralité des modifications du code source est modélisée en UML en annexe A. Ces changements ne sont pas visibles par l'utilisateur que via les interfaces de gestion des volumes chiffrés. Il convient donc de modifier ces interfaces en conséquence, cette partie y est dédiée.

Cette fonctionnalité d'utilisation de données d'une carte EMV comme keyfile s'intègre dans deux cas d'utilisation de volumes chiffrés : la création et l'ouverture dudit volume. Dans ces deux cas, l'utilisateur doit pouvoir accéder à la liste des cartes EMV présentes dans les lecteurs de cartes branchés à sa machine, afin de pouvoir choisir lesquelles utiliser. À l'instar des keyfiles stockés sur les cartes PKCS#11, les données des cartes EMV devront être exportables sur la machine de l'utilisateur quand ce dernier le souhaite. Ces deux fonctionnalités seront intégrées en ligne de commande ainsi que sur l'interface graphique, et ce, de manière quasi identique à ce qui existe pour les cartes PKCS#11 rendant ainsi transparente l'utilisation d'une carte EMV.

5.1 Intégration en ligne de commande

Tout d'abord, la fonctionnalité de lister les keyfiles stockés sur une carte PKCS#11, via la ligne de commandes, n'est présente que sous Unix. Son absence sous Windows s'explique pour des raisons techniques ainsi que le fait que la version Windows de VeraCrypt n'est pas faite pour être utilisée en ligne de commande. Le paramètre équivalent avec les cartes EMV ne sera donc intégré que sous Unix.

Actuellement, le paramètre `--list-token-keyfiles` permet de lister les keyfiles stockés sur des cartes PKCS#11. À terme, il permettra de lister les keyfiles des deux types de cartes (PKCS#11 & EMV). Les paramètres `--list-securitytoken-keyfiles` et `--list-emvtoken-keyfiles` seront ajoutés afin d'afficher respectivement les keyfiles stockés sur des cartes PKCS#11 et les cartes EMV.

La prise en charge des cartes EMV pour la création et l'ouverture de volumes se traduira par la mise à disposition de structure de chemin dédiée aux cartes EMV (voir partie 2.3), à utiliser comme paramètre de keyfile. Cette fonctionnalité n'induit aucune modification du point de vue de l'utilisateur, mais simplement un ajout de fonctionnalité.

Utilisation d'un keyfile stocké sur une carte PKCS#11 :

```
1 veracrypt -t --keyfiles=token://slot/0/file/fileName
```

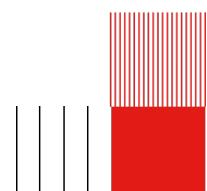
Utilisation d'une carte EMV comme keyfile :

```
1 veracrypt -t --keyfiles=emv://slot/0
```

L'export de keyfile sera, lui aussi, possible lorsque l'on utilise un chemin pointant sur une carte EMV. La démarche ne changera donc pas du point de vue utilisateur qui utilisera toujours le paramètre `--export-token-keyfile` puis en précisant le chemin de la carte EMV. Les octets des données des cartes EMV (certificats ICC et ISSUER, données CPLC) seront tout simplement écrites concaténées dans le fichier.

5.2 Intégration à l'interface graphique

VeraCrypt prenant déjà en charge le type de carte PKCS#11, l'intégration du type de carte EMV dans la même fenêtre de l'interface est le choix le plus judicieux. En effet, limiter les modifications graphiques du logiciel et le rassemblement en un même endroit des deux types de keyfiles permet de simplifier l'expérience utilisateur tout en gardant une cohérence avec le logiciel existant. La démarche pour créer et ouvrir un volume ne changera donc pas, la seule différence résidera dans la fenêtre du choix des keyfiles issus de cartes à puce. L'accès à la liste des keyfiles stockés sur une carte PKCS#11 se fait dans la fenêtre de choix (Figure 12), accessible en cliquant sur "Fichier clé" puis "Ajouter fichiers jeton". Les cartes à puce connectées à la machine de l'utilisateur sont alors lues et leurs keyfiles affichés dans une nouvelle fenêtre. Cet enchaînement d'actions est visible ci-dessous en figures 10 et 11.



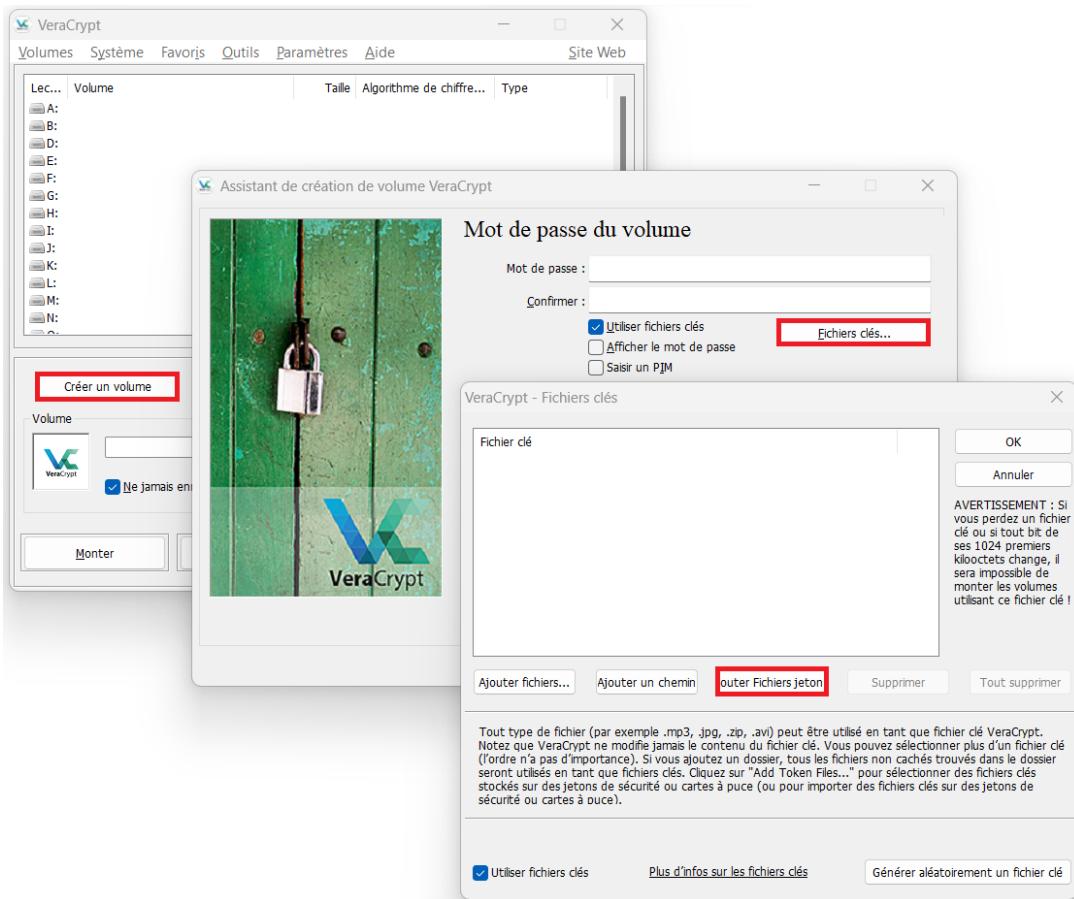


Figure 10 – Ajout d'un keyfile stocké sur une carte PKCS#11 à la création d'un volume

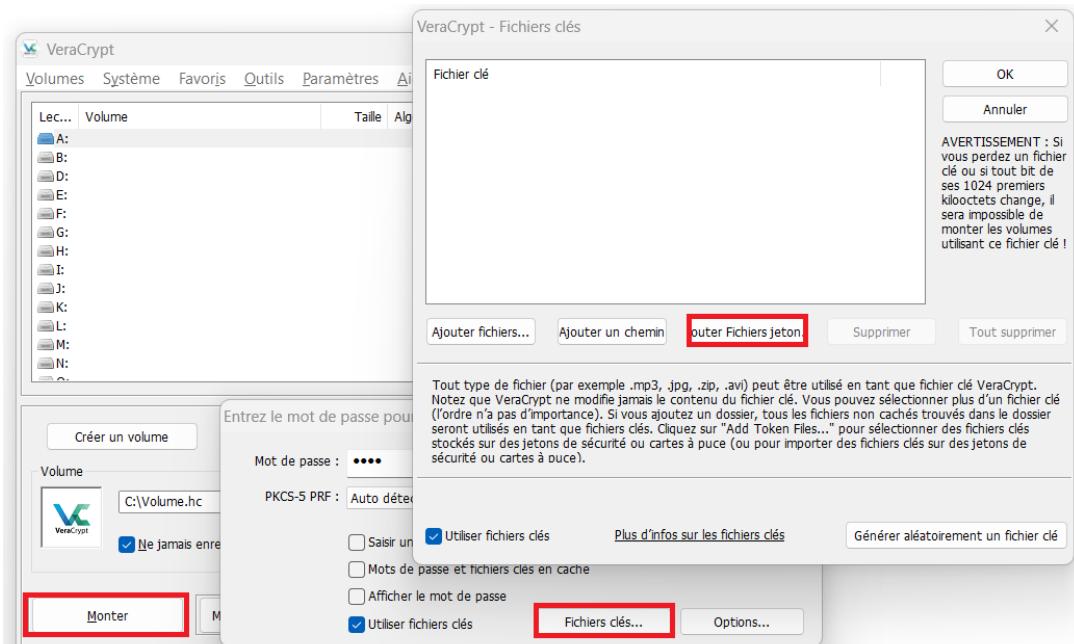


Figure 11 – Ajout d'un keyfile stocké sur une carte PKCS#11 à l'ouverture d'un volume

Actuellement, lorsqu'un keyfile PKCS#11 est sélectionné dans la fenêtre de sélections (voir Figure 12), plusieurs options s'offrent à l'utilisateur. Il peut cliquer sur "OK" pour pouvoir l'intégrer dans le processus de création ou de déchiffrement du volume. Le bouton "*Importer un fichier clé*" permet de placer sur la carte PKCS#11 un keyfile stocké sur l'ordinateur. Le bouton "*Exporter*" permet d'exporter un keyfile de la carte PKCS#11 sur un disque dur sous forme de fichier binaire, tandis que le bouton "*Effacer*" permet de le supprimer de la carte.

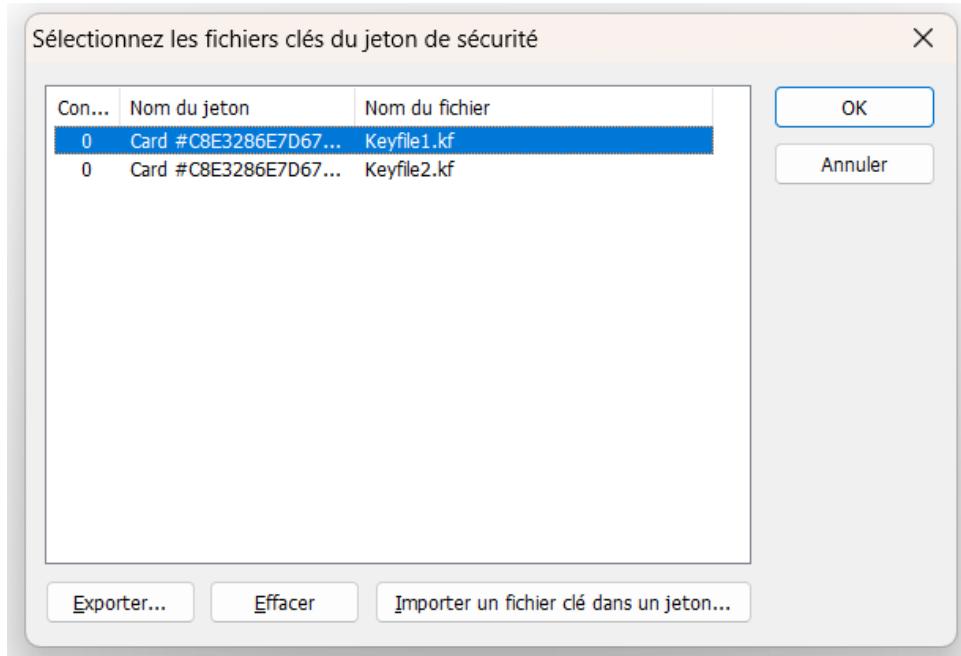
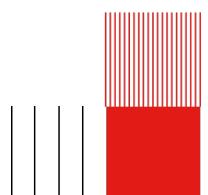


Figure 12 – Interface permettant l'ajout de keyfile depuis une carte PKCS#11

Les cartes à puce de type EMV n'étant pas disponible en écriture, mais seulement en lecture, l'import et la suppression des keyfiles ne sont pas disponibles à l'utilisateur. Dans la nouvelle interface de VeraCrypt lorsqu'une carte EMV est sélectionnée (voir Figure 13), le bouton "*Effacer*" est désactivé et le bouton "*Importer un fichier clé*" mène vers une fenêtre ne proposant pas l'import de données sur la carte EMV. Cependant, le bouton "*Exporter*" restera disponible afin de permettre à l'utilisateur de stocker sur sa machine les données de sa carte EMV sous la forme du fichier binaire de son choix. Ce fichier contient la concaténation des certificats utilisés comme keyfile, récupérés comme décrite partie 4.



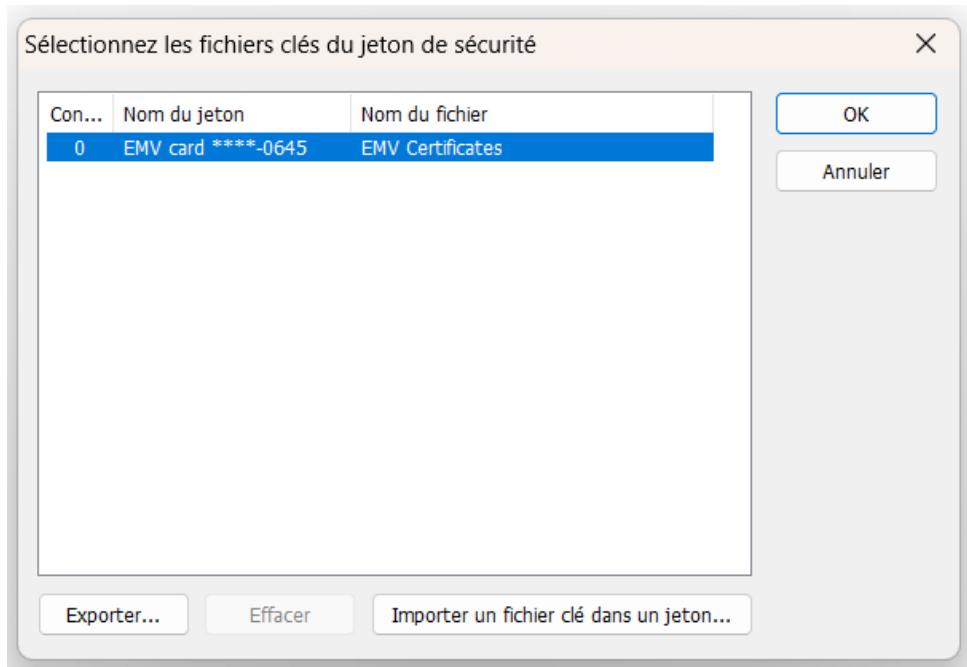


Figure 13 – Interface permettant l'ajout de cartes EMV comme keyfile

Actuellement, l'interface graphique de VeraCrypt est disponible dans de nombreux langages. Ces multiples traductions sont fournies par des contributeurs bénévoles appartenant aux communautés parlant ces langues, donnant ainsi aussi une bonne vision des zones du monde dans lesquelles ce logiciel est utilisé. Ce projet étant mené par une équipe francophone d'étudiants en ingénierie, seulement deux versions de l'interface graphique seront produites : une Française et une Anglaise.

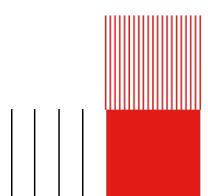
5.3 Optionnalité de la fonctionnalité

VeraCrypt est un logiciel de cybersécurité dont les utilisateurs aiment être en maîtrise complète de fonctionnalités de l'application. Ainsi, dans la version Bêta de VeraCrypt, l'utilisateur pourra choisir ou non d'activer la fonctionnalité EMV dans les paramètres du logiciel. Cette optionnalité a été pensée pour les personnes réticentes quant à cet ajout et souhaitant continuer d'utiliser VeraCrypt uniquement avec des cartes PKCS#11. Leur fournir ce choix l'acceptation de la fonctionnalité.

Cette option devra être modifiable dans les préférences de l'utilisateur, ce qui implique des modifications de l'interface graphique de VeraCrypt. Il peut aussi être utile d'ajouter des informations sur l'UI sur les conséquences potentielles de l'activation ou de la désactivation de cette fonctionnalité, afin que les utilisateurs puissent prendre une décision informée. Les utilisateurs sont en général plus enclins à adopter cette fonctionnalité s'ils comprennent clairement ce qu'elle fait.

5.4 Gestion des erreurs liées à la librairie PKCS#11

Pour pouvoir utiliser des cartes PKCS#11 au sein de VeraCrypt, l'utilisateur doit au préalable configurer une librairie dans les paramètres du logiciel. L'ouverture de la fenêtre "Sélectionnez les fichiers clés du jeton de sécurité" (voir Figure 12) renvoie une erreur si aucune librairie PKCS#11 n'est paramétrée. Ce comportement devra donc changer afin que le cas d'usage de carte EMV sans librairie PKCS#11 paramétrée ne crée pas d'erreur.



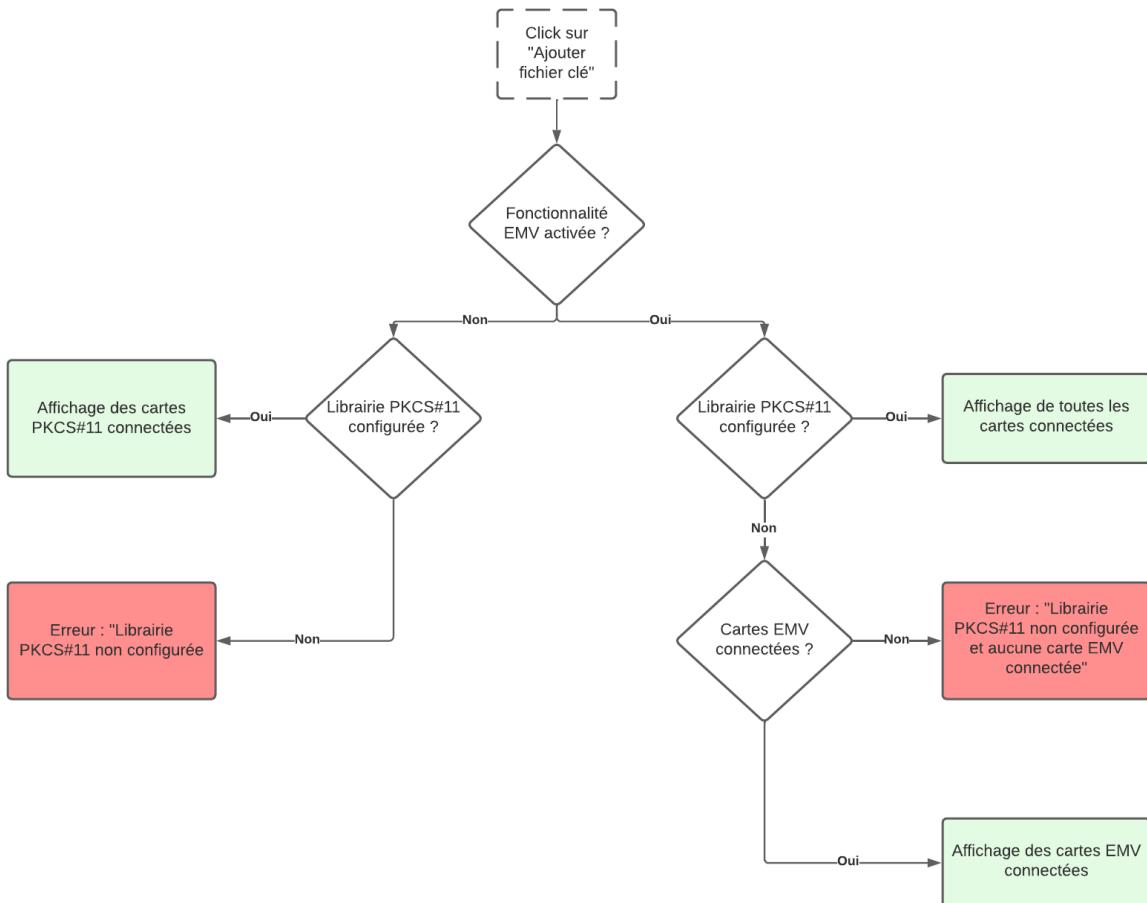
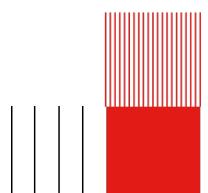


Figure 14 – Diagramme de cas d'utilisation

Le diagramme de cas pensé est disponible à la figure 14. Après avoir cliqué sur le bouton pour ajouter un kefile, si la fonctionnalité EMV est désactivée, le fonctionnement restera le même : un message d'erreur s'affiche si aucune librairie PKCS#11 n'est configurée. Dans le cas contraire où la fonctionnalité EMV est activée, le programme vérifie que la librairie PKCS#11 est installée. Si elle ne l'est pas mais qu'il détecte des cartes EMV connectées, alors aucun message d'erreur n'apparaîtra et uniquement les cartes EMV seront affichées. Ainsi, l'utilisateur ne souhaitant utiliser que des cartes EMV ne sera pas confronté à un message d'erreur à chaque utilisation. Toutefois, si aucune librairie n'est détectée et qu'aucune carte EMV n'est connectée, alors l'utilisateur obtiendra un message d'erreur, indiquant ces deux faits.

En analysant avec attention le diagramme, il est possible d'y voir un cas d'usage non pris en compte. En effet, dans la situation où à la fois des cartes de type EMV et PKCS#11 sont connectées mais que la librairie PKCS#11 n'est pas configurée, selon la figure 14 le message d'erreur n'est pas affiché à l'utilisateur. Ce dernier accède donc à la fenêtre de sélection des keyfiles pour n'y voir que ses cartes EMV et pas ses keyfiles stockés sur ses cartes PKCS#11. Sans le message d'erreur, il ne saura pas d'où vient ce problème, lié à l'absence de la librairie. Toutefois, ce cas d'usage a été évoqué avec Mounir Idrassi, d'après qui ce cas de figure est assez rare pour être négligé. Le comportement de l'utilisateur face à ce problème a été supposé : il testera uniquement avec ses cartes PKCS#11 connectées et verra apparaître le message d'erreur.



6 Tests et mesures de performance de l'application

6.1 Tests unitaires et d'intégration

Les tests unitaires sont une partie importante du développement logiciel pour s'assurer que le code fonctionne comme prévu. Ils peuvent aider à détecter les erreurs de manière précoce et à garantir que les modifications apportées au code n'affectent pas les fonctionnalités existantes. En ce qui concerne le C++, il existe plusieurs frameworks de tests unitaires populaires tels que Google Test [3], Boost Test [1] et CppUnit [2].

Le choix s'est porté sur la première alternative, puisque Google Test est un framework de test plus récent et plus avancé. En effet, il offre de nombreuses fonctionnalités utiles, telles que le support pour les tests basés sur le type et sur la valeur, les tests fixtures, les filtres de test et beaucoup plus. Google Test a également une grande et active communauté, ce qui signifie qu'il est facile de trouver de l'aide et du support.

Lors de l'écriture de tests unitaires pour le C++, il faudra couvrir autant de cas d'utilisation que possible pour s'assurer que le code fonctionne correctement dans toutes les situations possibles. Les tests seront écrits de manière à être automatisables et répétables pour minimiser les erreurs humaines.

En outre, il sera important de suivre une approche de test par couche pour s'assurer que les tests sont suffisamment exhaustifs et qu'ils couvrent la plus grande partie du code possible. Cela peut inclure des tests pour les classes et les fonctions individuelles, ainsi que des tests pour les interactions entre les différentes parties du code.

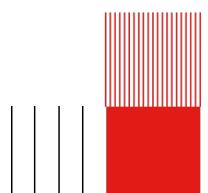
Ces tests seront d'une grande utilité lorsqu'il faudra s'assurer du bon fonctionnement du code sur tous les systèmes d'exploitation que supporte VeraCrypt.

6.2 Tests de détection de fuites de mémoires

Comme le code est développé en C++, un langage sans "garbage collector", il convient de s'assurer qu'il n'y a aucune fuite de mémoire pouvant impacter les performances ou même faire planter l'application. Valgrind [6], un outil de débogage de la mémoire, sera utilisé en conjonction de GDB pour surveiller les allocations et les libérations de mémoire et pour indiquer toutes les fuites de mémoire détectées, y compris le nombre de fuites, la taille totale des fuites et la ligne de code où elles se produisent.

6.3 Tests de performances

Finalement, des tests de performances seront réalisés afin d'évaluer l'impact de la communication avec la/les carte/s EMV sur le chiffrement/déchiffrement de volume VeraCrypt. En utilisant des scénarios de test représentatifs pour mesurer les temps d'exécution, les résultats seront comparés aux performances actuelles de VeraCrypt (notamment avec des cartes PKCS#11) pour déterminer l'impact de la nouvelle fonctionnalité. Il sera aussi intéressant de quantifier le nombre d'APDU envoyés lors de l'utilisation de la fonctionnalité EMV. En cas de problèmes de performance (plus de 1,5-2s d'attente), les ajustements seront apportés au code pour améliorer les performances et garantir un fonctionnement optimal.



7 Gestion de projet

7.1 Répartition des rôles

En janvier 2023, le nombre d'étudiants travaillant sur ce projet a baissé de sept à cinq suite au départ en mobilité de deux membres de l'équipe. Toutefois, nous sommes toujours assez nombreux pour nous répartir en deux équipes pour la suite du projet, de manière similaire à la phase de pré-étude. En effet, la présence de deux systèmes d'exploitation rend pertinente cette répartition des tâches :

Equipe Windows : 3 personnes sont chargées de l'intégration de notre fonctionnalité dans cette version VeraCrypt et des tests sur cette plateforme.

Equipe Linux : 2 personnes sont chargées de l'intégration de notre fonctionnalité dans cette version VeraCrypt et des tests sur cette plateforme.

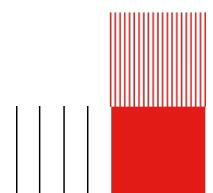
7.2 Suivi de Mounir Idrassi

Début novembre 2022, nous avions rencontré Mounir Idrassi, principal développeur du logiciel VeraCrypt, lors d'une visioconférence. Nous en avions profité pour échanger sur de nombreux points techniques, mieux comprendre le fonctionnement de VeraCrypt et prendre en considération ses conseils quant au projet. Le 1er février 2023, une deuxième réunion en distanciel avec M. Idrassi a été organisée par nos encadrants, dont les objectifs étaient les suivants :

Avancée du projet Démonstration de la version modifiée de VeraCrypt permettant l'utilisation de cartes EMV, présentation de nos ajouts et surtout justification de nos choix notamment au niveau de l'interface graphique.

Échange avec M. Idrassi Prise en compte de son avis sur la démonstration et sur nos choix, débat sur les points restants à développer et des erreurs à gérer.

Le retour que M. Idrassi nous a fait indique que tous les éléments sont réunis pour l'intégration de notre fonctionnalité dans une version Bêta de VeraCrypt. Il a confirmé nos choix, notamment celui de peu modifier l'interface graphique, ce qui facilite la prise en main. Le fait d'avoir un comportement quasi identique entre les deux cartes ne perd pas l'utilisateur suite à l'ajout de cette fonctionnalité.



7.3 Avancement

La faisabilité technique du projet a été démontrée par la réalisation de deux *proofs of concept* durant la phase de spécification fonctionnelle du projet. Depuis, ce dernier a avancé : de nombreuses tâches ont été complétées, certaines ont été abandonnées faute de temps (voir section 7.4) et d'autres redéfinies. L'avancement des tâches du projet, présenté dans la table 1 ci-dessous, démontre la finition dans les temps impartis du projet.

Table 1 – Liste des tâches et leur avancement

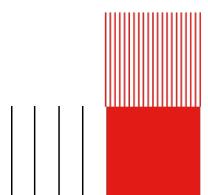
Tâches	Avancement
Étude du fonctionnement de VeraCrypt	100%
Étude de la cryptographie dans VeraCrypt	100%
Étude du standard EMV	100%
Choix des données EMV à extraire	100%
Étude du code source de VeraCrypt	100%
Déterminer où intégrer notre fonctionnalité dans le code	100%
Extraction des données des cartes EMV	100%
Démontrer la faisabilité technique du projet	100%
Etude et définition de l'UX	100%
Réaliser une maquette de l'interface graphique	100%
Développement de l'interface graphique pour Unix	100%
Développement de l'interface graphique pour Windows	100%
Avoir une version fonctionnelle du projet sur Unix	100%
Avoir une version fonctionnelle du projet sur Windows	100%
Effectuer des tests d'utilisation	40%
Effectuer des tests de performance	0%
Avoir une version propre du projet incorporable au logiciel officiel	60%
Prise en charge du sans contact	30%
Prise en charge des cartes EMV émulées sur un téléphone	retrait
Stockage d'un keyfile sur téléphone, récupérable via NFC ou BT	retrait
Séparation de l'entête et du corps du volume	retrait
Générer de l'aléa avec des cartes à puce	retrait

7.4 Planification des tâches

En début de projet, nous avions défini les grandes étapes qui allaient le rythmer. Néanmoins, à cause des modifications du calendrier universitaire (perte d'un mois de travail), ces dates clés ont été revues et certaines ajoutées :

- **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 Fin-décembre** : faire fonctionner conjointement le travail et POC des 2 équipes EMV et VeraCrypt, pour obtenir un résultat fonctionnel.
- **Mars Janvier** : ajout de l'interface graphique et de l'UX.
- **Février** : gestion et correction des erreurs, tests.
- **Février Mars** : mise au propre et incorporation dans le logiciel officiel.

Ce raccourcissement du délai de livraison du produit n'aura pas d'impact sur sa qualité finale. Seule la réalisation des fonctionnalités facultatives est contrainte à une révision. En effet, nous nous voyons dans l'incapacité de réaliser au moins quatre des cinq fonctionnalités suggérées par messieurs Dupas et Idrassi. Seule la prise en charge du sans-contact des cartes EMV est envisageable, car ne nécessitant pas d'énormes changements au niveau du logiciel, mais en aucun cas garantie.



Conclusion

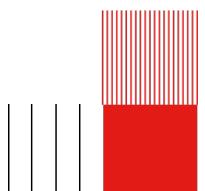
Ce projet de développement logiciel présente une philosophie propre aux logiciels de cybersécurité. L'introduction de failles dans la sécurité de la machine de l'utilisateur est à proscrire pour assurer à ce dernier l'intégrité de cette nouvelle fonctionnalité. Les modifications du code source de VeraCrypt doivent donc être chirurgicales afin que le strict nécessaire soit adapté tout en gardant la consistance du code. De même, les modifications des interfaces utilisateur doivent être limitées afin de garder une simplicité d'utilisation des cartes à puce. Mounir Idrassi suit l'avancement du projet et donne d'importants conseils sur la façon d'implémenter cette fonctionnalité, qui pourrait être intégrée dans une version Bêta de VeraCrypt. Cette dernière nécessitera forcément une maintenabilité du code et une documentation utilisateur en anglais incorporée dans celle du logiciel.

Les keyfiles stockés sur des cartes de type PKCS#11 sont gérés par trois structures permettant de représenter le keyfile sous deux formes, une structurelle et un chemin, et sa carte source. À l'instar de l'implémentation actuelle, une nouvelle organisation des structures a été développée. Trois structures génériques permettent de représenter tout type de carte à puce et leurs keyfiles. Héritant de deux de ces structures, les cartes EMV et PKCS#11 possèdent chacune deux structures dédiées pour les représenter. Elles permettent entre autres d'informer sur la disponibilité ou non de la carte à puce en écriture. Ces deux types de keyfiles sont représentables par des chemins dédiés dont les paramètres renseignent sur le lecteur auquel est connectée la carte associée. Ainsi trois classes gérant ces trois groupes de structures sont utilisées dans l'algorithme de VeraCrypt. Elles assurent la communication avec les cartes à puce afin de s'y connecter et d'en extraire les données des keyfiles, pour les stocker dans les structures évoquées ci-dessus. La classe responsable des cartes PKCS#11 est restée inchangée mais a servi de modèle pour la création de celle en charge des cartes EMV. Les avantages du design pattern proxy ont été utilisés lors de la création d'une classe de base gérant les appels de classes extérieures en les redirigeant vers la bonne classe de carte à puce. L'extraction des données des cartes EMV est assurée par deux classes, une pour l'extraction pure et l'autre pour l'analyse des données récupérées, encodées sous le format BER-TLV. De nombreuses méthodes ont été développées et assurent le bon fonctionnement du processus. Il reste cependant plusieurs cas d'erreurs à communiquer à l'utilisateur.

L'utilisateur dispose de deux interfaces pour manipuler ses volumes VeraCrypt. Via l'interface en ligne de commande, il peut accéder à la liste des keyfiles disponibles sur les cartes connectées à sa machine et même se limiter à certains types de cartes, et cela grâce à trois paramètres. Évoqués plus haut, les keyfiles sont notamment représentés en interne sous la forme de chemins structurés, utilisable en ligne de commande pour préciser ceux à utiliser pour ouvrir/créer le volume. Enfin, un paramètre de commande permet d'exporter les données EMV dans un fichier binaire local. Via l'interface graphique, les cartes EMV sont affichées tel que spécifié dans le rapport de la phase éponyme. L'indisponibilité des cartes EMV en écriture est prise en compte, désactivant certaines fonctionnalités de suppression et d'import de keyfiles sur la carte. Cette fonctionnalité EMV devra être rendue optionnelle dans les paramètres du logiciel afin de permettre aux utilisateurs d'avoir le choix quant à son utilisation. De nombreux cas d'usage liés à la présence ou non d'une librairie pour les cartes PKCS#11 ont été pensés et présentés dans un diagramme explicatif.

La validation de cette fonctionnalité se fait à travers de tests et de mesures de performances. Les tests unitaires et d'intégration assureront le bon comportement du code. Pour cela, un framework a été choisi pour sa qualité de développement et de communauté. Les fuites mémoires seront investiguées à l'aide d'un outil de débogage mémoire afin d'éviter toute fuite dans le code en cpp. Les performances de l'application seront quantifiées et validées après d'éventuels améliorations du code.

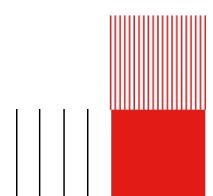
Pour réaliser la réalisation du projet, l'organisation des rôles a été revue, toujours basées sur le travail en deux équipes, une par système d'exploitation. Mounir Idrassi continue son suivi à distance lors de visioconférences permettant de lui présenter les avancées du projet et d'échanger avec lui sur le chemin à prendre pour se diriger vers une intégration de la fonctionnalité dans une version Bêta de VeraCrypt. L'avancement du projet a été mis à jour, certaines tâches ont été retirées de la liste à cause d'un changement du calendrier universitaire, et par conséquent de la planification des tâches.



Bibliographie

Références

- [1] Boost Test.
URL : https://www.boost.org/doc/libs/1_66_0/libs/test/doc/html/index.html. (Dernière consultation : 12/02/2023).
- [2] CppUnit.
URL : <https://www.freedesktop.org/wiki/Software/cppunit/>. (Dernière consultation : 12/02/2023).
- [3] Google Test.
URL : <https://google.github.io/googletest/>. (Dernière consultation : 12/02/2023).
- [4] 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.
- [5] 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>.
- [6] Valgrind.
URL : <https://valgrind.org/docs/manual/index.html>. (Dernière consultation : 12/02/2023).
- [7] winscard.
URL : <https://learn.microsoft.com/en-us/windows/win32/api/winscard/>. (Dernière consultation : 12/02/2023).



A Diagramme de classe UML de la nouvelle architecture (ajouts en orange)

