Analyse forensique de la mémoire des cartes à puce

Thomas Gougeon

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité : Informatique

Préparée au sein de l'ENSICAEN et de l'UNICAEN

Analyse forensique de la mémoire des cartes à puce

Présentée et soutenue par
Thomas GOUGEON

Thèse soutenue publiquement le 04/10/2017 devant le jury composé de		
Mr Hervé DEBAR	Professeur de l'Institut Mines-Telecom à Telecom SudParis	Rapporteur
Mr Vincent NICOMETTTE	Professeur des Universités à l'INSA de Toulouse	Rapporteur
Mr Aurélien FRANCILLON	Maître de conférences à EURECOM	Examinateur
Mr Pierre PARADINAS	Professeur des Universités au CNAM	Examinateur
Mr Gildas AVOINE	Professeur des Universités à l'INSA de Rennes	Co-directeur de thèse
Mr Christophe ROSENBERGER	Professeur des Universités à l'ENSICAEN	Directeur de thèse
Mr Morgan BARBIER	Maître de conférences à l'ENSICAEN	Examinateur
Mr Patrick LACHARME	Maître de conférences à l'ENSICAEN	Examinateur

Thèse co-dirigée par Christophe ROSENBERGER, laboratoire GREYC et Gildas AVOINE, laboratoire IRISA

UNIVERSITÉ CAEN NORMANDIE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

MI ΣΙ

GREYC

Résumé

Dans notre monde toujours plus connecté, les cartes à puce sont impliquées quotidiennement dans nos activités, que ce soit pour le paiement, le transport, le contrôle d'accès ou encore la santé. Ces cartes contiennent des informations personnelles liées aux faits et gestes de leur possesseur. Le besoin d'interpréter les données contenues dans les mémoires de ces cartes n'a jamais été aussi important. Une analyse forensique peut être utile dans différents scénarios (i) pour établir des preuves en relation avec des enquêtes criminelles (ii) pour retrouver des informations à propos d'une personne disparue, ou (iii) pour vérifier si un système est conforme avec les affirmations du fabricant ou de la législation en vigueur, ou encore (iv) prendre conscience de la fuite potentielle de nos informations personnelles.

Dans notre contexte, un dump mémoire représente le contenu de la mémoire non volatile de la carte à puce. Dans la plupart des cas, ni l'accès ni le contenu de ces données ne sont protégés par un mécanisme cryptographique. Cependant, sans les spécifications de l'application, il est difficile de connaître quelles informations sont stockées dans la carte, leur emplacement précis, ou encore l'encodage utilisé.

L'objectif de cette thèse est de proposer une méthode qui retrouve les informations stockées dans les dumps de mémoire non volatile des cartes à puce. Ces informations peuvent être reliées au possesseur de la carte, aux actions effectuées avec la carte ou encore à l'application. Les informations recherchées dans cette thèse représentent des dates (e.g., une date de naissance ou une date d'utilisation de la carte) et des informations textuelles (e.g., des noms, prénoms, adresses, nom de l'application). Pour retrouver ces informations, un décodage exhaustif des données à l'aide de différentes fonctions de décodage est possible. Malheureusement, cette technique génère de nombreux faux positifs. Ces derniers apparaissent lorsqu'une fonction de décodage est appliquée sur des données qui ont été encodées avec une fonction différente. Cette thèse s'appuie alors sur trois contributions exploitant les spécificités des cartes à puce pour éliminer ces faux positifs.

Les cartes à puce pouvant contenir des objets cryptographiques, la première méthode proposée élimine les faux positifs liés au décodage effectué sur objets cryptographiques. Ces objets cryptographiques peuvent être des données chiffrées ou hachées, des clés cryptographiques, des signatures, etc. Aucune information personnelle ne peut être obtenue à partir des objets cryptographiques si l'on considère que les algorithmes cryptographiques utilisés sont sûrs. Le décodage de ces données générera donc de nombreux faux positifs. Les deux autres méthodes permettent de retrouver les informations textuelles et les dates stockées dans les dumps. La première phase de chacune de ces méthodes consiste à décoder les données binaires pour générer des chaînes de caractères ou des dates. Ensuite, ces méthodes éliminent les faux positifs générés à l'aide d'une méthode spécifique à chacune. Pour les informations textuelles, l'élimination des faux positifs repose sur des statistiques de textes en analysant la fréquence des n-grammes. Quant aux dates, l'élimination des faux positifs est basée sur l'exploitation d'informations contextuelles (e.g., informations imprimées sur la carte à puce ou connaissance de la période d'utilisation de la carte).

Chacune des trois méthodes est enrichie par une analyse multi-dump. Celle-ci exploite les spécificités des cartes à puce en analysant plusieurs dumps provenant de la même application simultanément. Enfin, afin de valider ces méthodes, elles sont chacune appliquées sur 371 dumps provenant de cartes à puce de la vie réelle.

Table des matières

Introduction

Positionnement du problème

Le premier brevet lié à la carte à puce a été déposé en 1974 par Roland Moreno. Une carte à puce est une carte comportant un circuit intégré pouvant analyser et traiter des données. Dès la fin des années 1980, des millions de cartes sont utilisées dans le monde. Dans les années 2000, ces cartes se comptent en milliards. Parallèlement à cela, les puces à lecture sans contact sont apparues dans les années 80-90. Elles ont véritablement pris leur envol dans les années 2000. Certaines applications fonctionnent même sur des cartes qui possèdent deux interfaces de communication : avec et sans contact.

Dans un monde toujours plus connecté, ces cartes à puce sont impliquées dans les activités quotidiennes et peuvent contenir des informations personnelles liées à leur possesseur et leurs faits et gestes. Quelques exemples sont ainsi donnés. Les forfaits de ski peuvent contenir des informations à propos de la période de validité du forfait ou bien du domaine sur lequel il est valide. Ils peuvent aussi contenir des informations à propos de la dernière remontée mécanique utilisée telles que la date, l'heure et l'identifiant de celle-ci. Les cartes d'abonnement de transport public peuvent enregistrer des informations à propos des dates de validité et du type de l'abonnement. La carte enregistre aussi généralement des informations à propos des derniers trajets effectués comme la date, l'heure, la station, le numéro de la ligne, l'identifiant du lecteur, etc. Elles peuvent aussi stocker des informations à propos du possesseur de la carte comme : son âge, son code postal et son nom. Les cartes de paiement contiennent, quant à elles, des objets cryptographiques tels que des clés publiques ou des certificats. Elles stockent aussi d'autres informations telles que le numéro, la date de validité et le nom du possesseur de la carte. Elles enregistrent aussi l'historique des transactions effectuées en stockant pour chaque transaction la date, le montant, le type et le pays où a été effectué la transaction. Les cartes d'assurance

maladie (telle que la carte Vitale en France), peuvent contenir des informations à propos de leur possesseur comme : son nom, sa date de naissance et son adresse complète. Elles peuvent aussi stocker les prénoms, noms, et dates de naissance des éventuels ayant droits (e.g., enfants). Des dates d'événements médicaux peuvent aussi être enregistrées par la carte, telles que le début d'une ALD (affection longue durée) ou une utilisation de la carte dans une pharmacie. Elles contiennent également des objets cryptographiques utilisés pour l'authentification et le contrôle de l'intégrité des données de la carte. Les clés de contact dans les véhicules récents contiennent des informations à propos de la voiture. Elles sont utiles lors du diagnostic effectué par un garagiste. Ces informations incluent le temps restant avant le remplacement de certaines pièces mécaniques, les températures extérieures, les niveaux des liquides, etc. Ces clés contiennent aussi des informations liées au profil du conducteur dans l'habitacle. Ces informations peuvent être le réglage du siège du conducteur et des rétroviseurs. Ainsi, une configuration automatique de l'habitacle peut être effectuée lorsque deux personnes utilisant une même voiture possèdent chacun leur clé.

Le besoin d'interpréter les données contenues dans ces mémoires n'a jamais été aussi important. Cela peut être utile dans différents scénarios, (i) pour établir des preuves en relation avec des enquêtes criminelles, (ii) pour retrouver des informations à propos d'une personne disparue, (iii) pour vérifier si un système est conforme avec les affirmations du fabricant ou de la législation en vigueur, ou encore (iv) pour prendre conscience de la fuite possible de nos informations personnelles. En effet, plusieurs scénarios sont identifiés dans lesquels nos informations personnelles peuvent fuiter. Cela peut-être lors de la perte de notre porte-feuille : bien que des informations soient imprimées sur les cartes et peuvent être lues directement, la puce contient des informations personnelles supplémentaires telle que l'historique d'utilisation de la carte. On peut imaginer des scénarios d'attaque plus avancés lors d'une lecture à distance pour les puces sans contact ou lorsqu'un mouchard est présent sur un lecteur de carte contact.

Le domaine de la forensique informatique englobe la récupération des données des supports numériques ainsi que leur analyse. Les travaux existants se focalisent sur les supports tels que les mémoires mortes et volatiles des ordinateurs et des téléphones. Les techniques proposées pour analyser ces supports sont trop spécifiques pour être adaptées au contexte des mémoires des cartes à puce. En dépit du fort intérêt de pouvoir interpréter les données des mémoires des cartes à puce, il n'existe quasiment aucun travail traitant ces mémoires dans le domaine de l'analyse forensique.

L'objectif de cette thèse n'est pas d'extraire les données de la mémoire non volatile des cartes à puce mais de les interpréter. L'objectif n'est pas non plus d'analyser

le programme chargé dans la carte mais plutôt les données utilisées et produites par ce programme. Sans les spécifications de l'application, il est très difficile de retrouver les informations qui y sont stockées. En effet, leur emplacement précis dans la mémoire et la manière dont elles sont encodées sont inconnus. Cette thèse propose donc la première méthode générique permettant de retrouver automatiquement les informations stockées dans les mémoires non volatiles de cartes à puce.

Contexte de réalisation de la thèse

Cette thèse a été financée par une bourse du Ministère de l'Enseignement Supérieur et de la Recherche (MESR). C'est une collaboration entre les équipes Monétique et Biométrie du laboratoire GREYC à Caen et EMSEC du laboratoire IRISA à Rennes.

Contributions

L'objectif de ce manuscrit est de présenter les travaux effectués au cours des trois années de thèse en dont l'objectif est l'analyse forensique des mémoires des cartes à puce. Cela consiste à retrouver les informations personnelles qui sont stockées dans ces mémoires. Pour y parvenir, diverses fonctions de décodage sont appliquées sur l'intégralité des données de la mémoire. Malheureusement, cette technique naïve génère de nombreux faux positifs. Un faux positif apparaît lorsqu'une fonction de décodage est appliquée sur des données qui n'ont pas été encodées avec la fonction d'encodage correspondante. Les contributions visent donc à réduire le nombre de faux positifs générés.

La première contribution distingue les données *informationnelles* des données *cryptographiques*. Ces objets cryptographiques peuvent être des données chiffrées ou hachées, des clés cryptographiques, des certificats, des signatures, etc. Aucune information personnelle ne peut être obtenue à partir d'objets cryptographiques si l'on considère que les algorithmes cryptographiques utilisés sont sûrs. Les données dites *informationnelles* représentent toute les autres données (informations personnelles et techniques). Appliquer la technique de décodage exhaustive sur ces données ne pourra donc pas retrouver d'informations mais générera des faux positifs. On propose ainsi d'ignorer les données considérées comme cryptographiques lors de la phase de décodage, dans le but de réduire le nombre de faux positifs.

La deuxième contribution est une méthode qui retrouve automatiquement les informations textuelles stockées dans les mémoires des cartes à puce. Ces informations textuelles peuvent être des noms, des prénoms, des adresses postales ou électroniques, des noms d'applications, etc. La première étape de cette méthode est le décodage des données binaires extraites de la mémoire afin de générer des chaînes de caractères.

Ensuite, cette méthode élimine les faux positifs générés par le décodage. Pour cela, des statistiques de textes sont utilisées et les spécificités des cartes à puce sont exploitées.

La troisième contribution est la proposition d'une méthode récupérant automatiquement les dates stockées dans les mémoires des cartes à puce. Ces dates peuvent représenter une date de naissance, une date d'utilisation de la carte, une date de création, une date de validité, etc. La première étape de cette méthode décode les données binaires extraites de la mémoire afin de générer des dates. La suite de la méthode élimine les faux positifs générés par le décodage. Pour cela, elle exploite les spécificités des mémoires des cartes à puce et les informations contextuelles liées à la carte (e.g., informations imprimées sur la carte à puce).

Organisation du manuscrit

Le premier chapitre constitue le positionnement du problème de cette thèse. Tout d'abord, la problématique de l'analyse forensique des mémoires des cartes à puce est explicitée. Ensuite, ce chapitre définit les notions de dump mémoire de cartes à puce et présente la base de dumps utilisée pour les expériences et son acquisition. Par la suite, un état de l'art des différentes techniques d'analyses forensiques est présenté. Enfin, les spécificités de l'analyse forensique des cartes à puces sont décrites.

Le deuxième chapitre présente la première contribution consistant à séparer les données *cryptographiques* des données *informationnelles* dans les mémoires des cartes à puce. Une méthode combinant une analyse statistique et de l'apprentissage automatique est proposée. Des expériences sur des mémoires de cartes à puce de la vie réelle sont présentées.

Le troisième chapitre présente la deuxième contribution retrouvant les informations textuelles dans la mémoire d'une carte à puce. Une méthode analysant la fréquence des n-grammes à l'aide d'un classifieur Bayésien est proposée. Cette méthode est améliorée en analysant simultanément différentes mémoires de cartes à puce. Des expériences sur des dumps mémoires de la vie réelle sont présentées.

Le quatrième chapitre introduit la troisième contribution retrouvant les dates dans la mémoire des cartes à puce. Ce chapitre exploite essentiellement les spécificités des cartes à puce, à savoir l'analyse simultanée de différents dumps de cartes à puce ainsi que l'analyse des informations contextuelles. Des expériences sur des dumps mémoires de la vie réelle sont présentées.

Enfin, la conclusion présente le bilan des travaux ainsi que les perspectives ouvertes par cette thèse.

Chapitre 1

Positionnement du problème

Ce chapitre présente la problématique liée à l'analyse forensique des mémoires de cartes à puce. Après une introduction générale de cette dernière, le contexte de cette thèse lié aux dumps de mémoire des cartes à puce est défini. La base de dumps de la vie réelle acquise est également présentée, ainsi que les méthodes d'acquisition utilisées. Pour illustrer cette base, quelques exemples de dumps sont détaillés. Ensuite, ce chapitre fait l'état de l'art des différentes techniques d'analyse forensique existantes, démontrant leurs limites face à la problématique. Enfin, il est présenté les spécificités liées au contexte de la thèse qui seront utiles pour l'analyse forensique.

Sommaire

1.1 Problématique

L'objectif de cette thèse est de proposer une méthode automatique et générique qui interprète les données contenues dans les dumps de mémoire des cartes à puce. La difficulté du problème réside dans le fait que les spécifications des applications sont rarement publiques. Ajouté à cela, les spécifications accessibles sont souvent partielles. En effet, des libertés sont laissées à l'organisme qui développe l'application

installée sur chaque carte. Il est donc considéré dans cette thèse que les spécifications des dumps de mémoire analysés sont inconnues.

Il n'existe à l'heure actuelle aucune méthode automatique qui soit capable de traiter efficacement ce problème. De plus, très peu de travaux s'intéressent à l'analyse forensique des mémoires de cartes à puce. En effet, la communauté scientifique se concentre sur les différentes mémoires des ordinateurs ou téléphones. Les rares travaux qui ont été effectués sur les cartes à puce ont été faits de manière ad-hoc et le plus souvent manuellement. De telles analyses sont coûteuses en temps et ne sont pas adaptées pour traiter des applications variées dans un contexte opérationnel. Une méthode automatique est donc nécessaire pour améliorer l'efficacité et la qualité des investigations des experts, comme cela est souligné dans [1].

Sans ces spécifications, la présence, l'emplacement et la manière dont sont encodées les données dans la carte sont inconnus. Différents types de données peuvent être enregistrés dans les cartes, tels que des objets cryptographiques, des informations textuelles, des dates, des identifiants, des nombres, etc. Toutes ces informations sont stockées à des emplacements arbitraires dans le dump et peuvent être accolées les unes aux autres. Par exemple, il est possible qu'à côté d'une date l'on retrouve un nom, puis un objet cryptographique, puis de nouveau une date, etc. De plus, ces informations peuvent être encodées avec divers encodages au sein d'un même dump. Ces derniers sont très nombreux, allant des plus populaires comme l'ASCII ou BCD, aux plus compressés utilisant 5 bits pour représenter les lettres de l'alphabet. Il en existe même des plus spécifiques aux cartes à puce, par exemple : dans certains dumps de mémoires de cartes de transport, les dates sont représentées par le nombre de secondes écoulées depuis le 01/01/1997.

Afin de retrouver les informations stockées dans les mémoires de cartes à puce, il est proposé dans cette thèse d'effectuer une phase de décodage des dumps de mémoire. Ce processus décode les données du dump avec de nombreuses fonctions de décodage et cela à tous les emplacements possibles du dump. Appliquer une telle fonction sur un dump génère donc quasiment autant de sorties qu'il y a de bits dans ce dernier. Chaque fonction de décodage génère donc quelques centaines ou milliers de sorties par dump. Malheureusement, il n'existe pas d'oracle pouvant déterminer efficacement si oui ou non l'information décodée est correcte. De ce fait, de nombreux faux positifs sont générés, ce qui rend cette approche inutilisable en pratique. Plus précisément, un faux positif apparaît lorsqu'une séquence de bits dans un dump est décodée avec une fonction différente de celle qui a été utilisée pour l'encoder. En pratique, il est impossible d'effectuer le décodage avec toutes les fonctions de décodage imaginables. Les fonctions de décodage les plus pertinentes seront donc choisies.

Le travail se concentre donc sur l'élimination de ces faux positifs. Pour cela, plusieurs approches sont proposées. La première utilise des tests statistiques et de l'apprentissage automatique. La seconde utilise une technique d'analyse de textes et de l'apprentissage automatique. La dernière exploite exclusivement les spécificités de cartes à puce. À partir des dumps de mémoire de cartes à puce, la méthode proposée dans cette thèse retourne ainsi l'ensemble des informations générées par le décodage et qui n'ont pas été éliminées. Cette méthode est optimale si elle retrouve l'intégralité des informations qui sont stockées tout en retournant aucun faux positif. En réalité, il est difficile d'obtenir ce cas idéal, il est donc nécessaire d'effectuer un compromis entre les faux positifs et les faux négatifs. Un faux négatif correspond à une information qui a été éliminée par erreur. Dans ce travail, la présence d'un faux négatif est plus problématique que la présence d'un faux positif. Cependant, le nombre de faux positifs générés doit être maîtrisé afin que les résultats restent exploitables pour un analyste humain.

1.2 Dumps de mémoire de carte à puce

Cette section définit ce qu'est un dump de mémoire dans le contexte de cette thèse ainsi que les cartes à puce dont ils proviennent. Les outils matériels et logiciels permettant de les extraire sont aussi brièvement présentés. Ensuite, la liste des dumps de la vie réelle ainsi collectés est présentée et le contenu de quelques uns de ces dumps est détaillé.

1.2.1 Terminologie

On définit ici ce que sont les dumps de mémoire ainsi que les cartes à puce dont ils sont extraits.

Dump de mémoire

Un dump de mémoire est un ensemble de données binaires représentant le contenu de la mémoire à un instant donné. Les dumps de mémoire considérés dans cette thèse représentent le contenu de la mémoire non volatile des cartes à puce. Ces mémoires sont appelées EEPROM (pour *Electrically-Erasable Programmable Read-Only Memory* ou encore mémoire morte effaçable électriquement et programmable). Ici, un dump est donc l'ensemble des données binaires extraites de la mémoire EEPROM d'une carte à puce. Dans cette thèse, les dumps ne sont pas récupérés en faisant une copie bit à bit de l'EEPROM mais en communiquant avec l'API de la

carte. Au travers de l'utilisation de cette API il est possible de sélectionner et de récupérer des fichiers ou des secteurs de la mémoire. Parfois, certaines parties de l'EEPROM ne sont pas accessibles via l'API, ou bien, elles sont protégées par un mécanisme d'authentification. Il est donc possible que les dumps considérés dans cette thèse ne soient pas une copie exacte de l'EEPROM.

Les dumps considérés dans cette thèse sont les dumps où les données ne sont pas chiffrées ni obfusquées et dont l'accès n'est pas non plus protégé par un mécanisme d'authentification. La section 1.2.3 montre que la quasi-totalité des dumps rencontrés vérifient ces critères.

Carte à puce

Une carte à puce est une carte comportant un circuit intégré pouvant réaliser des calculs et stocker des informations. Il est possible d'extraire les données d'une carte à puce à travers deux interfaces : contact et sans contact.

1.2.2 Méthode d'acquisition des dumps

Comme expliqué précédemment, les dumps ne sont pas créés en faisant une copie bit à bit de l'EEPROM. Ils sont récupérés à l'aide de l'API de la carte considérée. Même si la méthode générale est identique pour tous les dumps, il existe certaines différences selon l'application utilisée. Cette section décrit les outils logiciels et matériels permettant d'extraire un dump de mémoire de l'EEPROM. Le matériel permet d'envoyer les commandes à la carte et les logiciels en définissent le contenu. Les différences entre les applications concernent donc le contenu de ces commandes. L'acquisition des dumps est en dehors du champ d'étude de cette thèse. Cependant, c'est une étape nécessaire pour mener à bien les expérimentations sur des données provenant d'applications de la vie réelle.

Interfaces

Il est possible d'extraire les données à travers deux interfaces, contact et sans contact. Cette section identifie les différents protocoles utilisés pour communiquer avec ces interfaces.

Contact Nombreuses sont les cartes qui sont compatibles avec la norme ISO 7816 qui définit notamment les protocoles de communication. Parmi les éléments de cette norme, figure la description des commandes dites *APDU* qui permettent d'échanger des messages entre un lecteur et une puce. Si le format des APDU est défini dans la

FIGURE 1.1 – Lecteur Omnikey 3121

FIGURE 1.2 – Lecteur ACR 122

norme ISO 7816, c'est en revanche aux concepteurs des applications disponibles sur les cartes d'en définir le contenu.

Sans contact Les cartes à puce sans contact communiquent par ondes radio avec un lecteur adapté à la fréquence et au protocole de la carte. La fréquence utilisée impacte indirectement la distance maximale de communication. La plus utilisée pour les applications à courte portée est 13,56 MHz. Dans cette bande de fréquence, il existe les normes ISO 14443 et ISO 15693 qui définissent le protocole de communication de bas niveau. Dans le cas de l'ISO 14443, la communication de haut niveau est définie par la norme ISO 7816, déjà mentionnée pour les cartes avec contact.

Matériel

Contact Des lecteurs permettant de communiquer avec des cartes à puce avec contact au travers des APDU sont disponibles pour quelques dizaines d'euros. On peut citer par exemple le lecteur *OMNIKEY 3121 USB* de HID Global ou un lecteur de la gamme *ACR38* de ACS. On peut voir un lecteur de cartes à puce Omnikey sur la figure 1.1.

Sans contact Des lecteurs permettant de communiquer avec des cartes à puce sans contact sont aussi disponibles pour moins d'une centaine d'euros. Des exemples de lecteurs sont le *Prox'N'Roll* de SpringCard, le *SCL 3711* de SCM, ou encore le *ACR 122* de ACS. On peut voir ces trois lecteurs sur les figures 1.2, 1.3 et 1.4. Une autre possibilité pour lire les cartes à puce sans contact est l'utilisation d'un téléphone muni d'un lecteur NFC. Ces lecteurs NFC sont compatibles avec une part importante des cartes sans contact. Le standard de la NFC est une extension de la norme ISO 14443 permettant, entre autres, de communiquer en *peer-to-peer*.

FIGURE 1.3 – Lecteur SCL3711

FIGURE 1.4 – Lecteur Prox'N'Roll

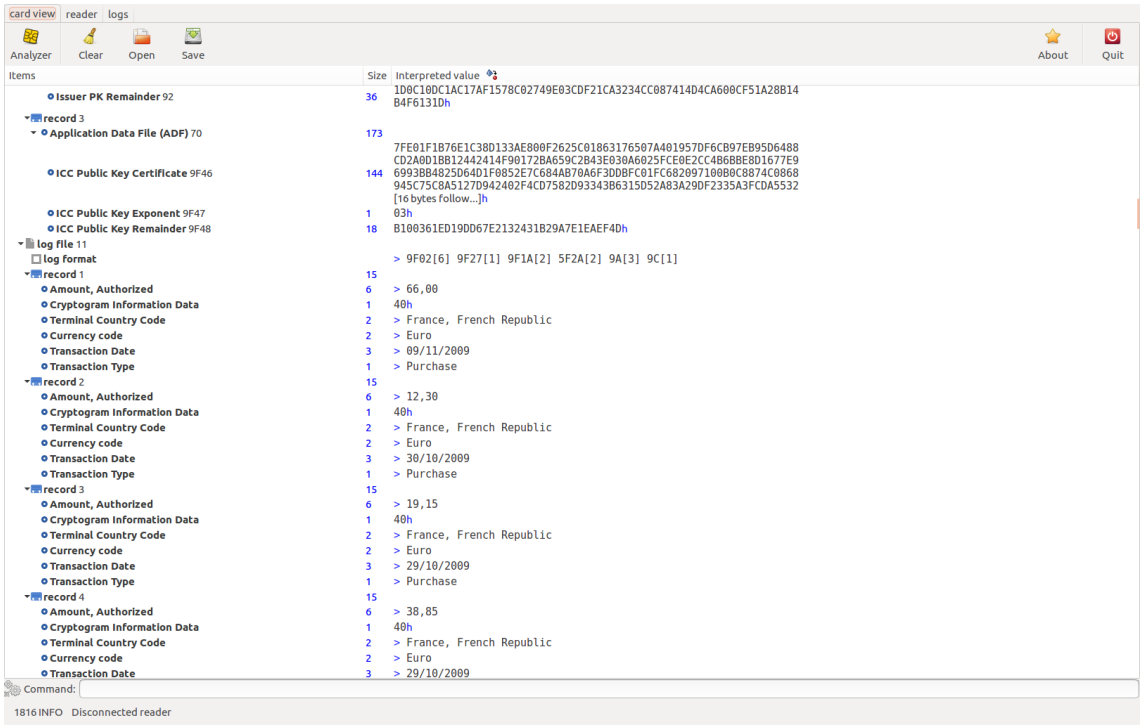


FIGURE 1.5 – Capture d'écran de Cardpeek

Logiciel

Détenir le bon matériel n'est pas suffisant pour récupérer les données contenues dans les mémoires EEPROM des cartes à puce. Il reste à savoir quels APDU doivent être envoyés à la carte. Ces APDU varient en fonction du type de la carte dont on veut extraire les données. Il existe de nombreux logiciels qui permettent d'extraire les données des mémoires de carte à puce, les principaux utilisés sont cités ci-dessous.

Cardpeek [2] est un outil utilisable avec Windows ou Linux qui est capable de

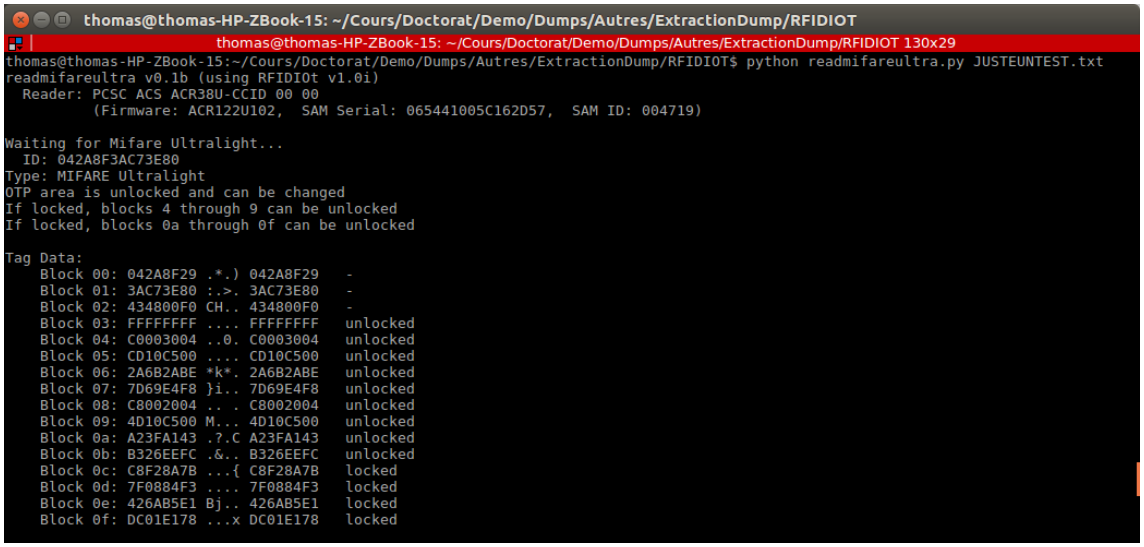


FIGURE 1.6 – Capture d'écran de RFIDIOt

récupérer les données de certaines applications : Calypso, EMV, Moneo, Vitale 2, les cartes d'identité électronique (eID) belge, les passeports et certaines cartes de la famille NXP Mifare. Cet outil fonctionne à la fois pour les cartes à puce avec contact et les cartes à puce sans contact. On peut voir un exemple d'utilisation de Cardpeek avec une carte EMV sur la figure 1.5.

Il existe aussi RFIDIOt [3] qui est une collection d'outils écrits en python pour communiquer avec la technologie RFID. Ces outils permettent, entre autres, d'extraire le contenu des mémoires de nombreuses cartes à puce sans contact. Ces cartes incluent les NXP Mifare Classic 1k et 4k, NXP Mifare Ultralight et les cartes compatibles avec la norme ISO 15693. On peut voir un exemple d'utilisation de RFIDIOt avec une carte Mifare Ultralight sur la figure 1.6.

Il existe aussi des applications sur téléphones portables pour extraire les données de certaines cartes sans contact compatibles avec le lecteur NFC de l'appareil. On peut par exemple citer l'application Android NFC TagInfo [4] de NFC Research Lab Hagenberg ou encore l'application NFC TagInfo By NXP [5] de NXP. On peut voir un exemple d'utilisation de NFC TagInfo sur la figure 1.7.

1.2.3 La base de dumps

371 dumps de mémoires de cartes à puce ont été récupérés à l'aide des outils précédemment décrits provenant d'environ 200 cartes de diverses applications. Cet ensemble ne contient évidemment pas tous les dumps de toutes les cartes à puce existantes, ni même, un dump de chaque application existante. Cependant, je considère

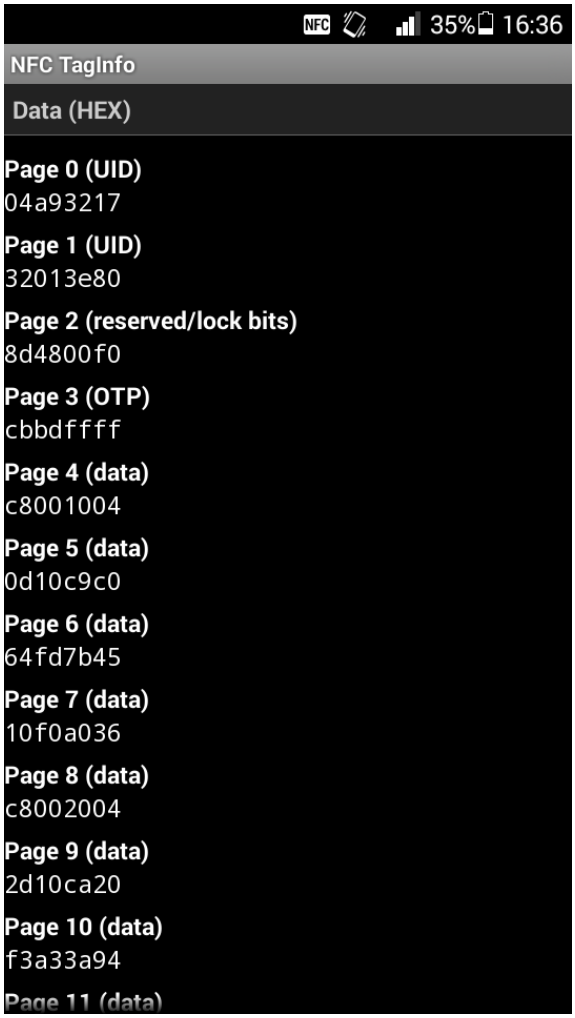


FIGURE 1.7 – Capture d'écran de NFC TagInfo

que cet échantillon est suffisamment représentatif des types d'informations stockées et des fonctions d'encodage utilisées de l'ensemble des applications de la vie réelle. Les méthodes proposées pour l'analyse forensique seront testées et validées sur cet ensemble de dumps.

Il est à noter qu'aucune donnée chiffrée n'a jamais été détectée dans les dumps considérés. Aussi, il n'y a pas plus de 10 cartes à puce parmi les 200 analysées qui sont protégées par un mécanisme d'authentification. C'est le cas de certaines Mifare DESFire ou Mifare Classic. Ajoutons à cela que, dans de rares cas, seulement l'accès à une partie des données est restreint. C'est le cas pour les passeports électroniques et la carte Vitale. Des détails sont fournis sur ces deux cas dans la suite de la thèse.

Ces dumps sont regroupés en différentes catégories :

— 91 dumps Calypso provenant de 21 cartes ou tickets de 7 villes différentes ;

— 55 tickets de transport de 12 villes de différents pays (France, Italie, Finlande,

Chine, etc.) ;
— 142 dumps de 40 forfaits de ski provenant de 19 stations de ski ;
— 8 dumps de passeports électroniques (7 belges et 1 français) ;
— 12 cartes Moneo ;
— 5 cartes Vitale ;
— 28 cartes de paiements EMV ;
— 30 dumps d'autres applications variées (carte d'identité électronique belge, cartes de meeting d'entreprises, Thalys, hôpital, etc.).

Calypso, EMV, Moneo

Calypso est un standard utilisé pour la billetterie dans le monde entier. Il est très employé pour les cartes d'abonnement de transport public. On peut citer le Pass Navigo à Paris ou la carte MOBIB à Bruxelles. L'accès au standard est restreint.

EMV pour Europay Mastercard Visa, est le standard international des cartes de paiement. Les spécifications fournies par EMV garantissent, entre autres, l'interopérabilité et la sécurité. Ce standard est public.

Moneo est un système de porte monnaie électronique. Ce système n'est plus développé depuis 2015. Les spécifications sont confidentielles.

Les mémoires de ces trois applications peuvent être extraites à l'aide du logiciel CardPeek. Les mémoires de type EEPROM de ces applications contiennent différents fichiers. Cardpeek sélectionne un à un ces fichiers et en extrait le contenu.

Vitale et Vitale 2

La carte Vitale est la carte d'assurance sociale en France. La carte Vitale de première génération a été remplacée par la carte Vitale 2 en 2007. Néanmoins, de nombreuses cartes Vitale de première génération sont encore utilisées. On peut les distinguer visuellement car la photo du titulaire est désormais imprimée sur la carte. Outre cet aspect visuel, la nouvelle carte possède plus de données. En effet, la carte Vitale 1 contenait seulement un volet administratif. La carte Vitale 2, quant à elle, contient non seulement ce volet administratif mais aussi un volet médical. Elle est aussi censée amener plus de sécurité. Dans la carte Vitale 2, l'accès à la photo du titulaire dans la mémoire EEPROM nécessite une authentification. Le secret utilisé pour s'authentifier est un code PIN que le titulaire de la carte ne connaît pas. Pour extraire les données de la carte Vitale 2, il est possible d'utiliser Cardpeek. Le système des données ressemble à celles des cartes Calypso et EMV.

Passeport électronique

En 2004, l'Organisation de l'aviation civile internationale (OACI) a publié une version du standard DOC 9303 qui préconise l'utilisation d'une puce sans contact pour renforcer la sécurité des passeports. Le contenu de ce standard est public.

La puce contient plusieurs fichiers dont le contenu est décrit dans le standard. Ces fichiers peuvent contenir, entre autres, la photo du titulaire, la photo de la signature manuscrite, l'empreinte biométrique ainsi que des informations à propos du titulaire et de la création du passeport. Selon le standard, certains fichiers sont obligatoires alors que d'autres sont facultatifs. Ainsi, l'ensemble des fichiers présents dans la puce du passeport électronique varie en fonction du pays émetteur du passeport.

Pour accéder aux données de la puce, une authentification est nécessaire. Le secret utilisé pour cette authentification est le contenu de la zone à lecture optique (Machine Readable Zone, ou MRZ), c'est-à-dire les deux lignes rigoureusement formatées qui se situent en bas de la page contenant la photo du titulaire. Autrement dit, toute personne ayant un accès physique au passeport est capable d'en extraire l'EEPROM. Néanmoins, l'intégralité du contenu de la puce n'est pas accessible à l'aide de cette authentification. Les données biométriques telles que les empreintes digitales nécessitent d'avoir un lecteur authentifié (réservé aux autorités). Pour extraire les données de la puce du passeport, il est possible d'utiliser Cardpeek.

Mifare

Il existe différents types de carte Mifare, voici les principaux rencontrés : Ultralight, DesFire, Classic 1k et Classic 4k. Les cartes Mifare Classic sont très répandues, elles sont utilisées pour du contrôle d'accès, de la billetterie, des porte-monnaie électroniques, etc. Beaucoup de tickets de transport à usage occasionnel utilisent des cartes Mifare Ultralight en raison de leur faible coût. Notons que très peu de Mifare DesFire ont été rencontrées, et que celles rencontrées ne contenaient que très peu de fichiers. Certaines utilisent seulement l'UID de la carte. Toutes ces cartes se comportent comme des cartes mémoires.

Les données sont séparées en secteurs pour les Mifare Classic, et chaque secteur est divisé en blocs. Leurs nombres et leurs tailles varient en fonction du type de la carte. La Mifare Classic 1k totalise 768 octets de données et la Mifare Classic 4k totalise 4 096 octets de données. Chacun des secteurs peut être protégé en lecture et en écriture. Cependant, différentes attaques [6, 7] ont été publiées pour mettre à mal ces protections. Des outils implémentant ces attaques sont disponibles [8, 9].

La carte Mifare Ultralight, quant à elle, contient seulement 512 bits, ses données sont découpées en 16 pages mémoire de 4 octets chacune. Il n'existe aucun mécanisme de sécurité pour ces cartes.

Pour extraire les données des puces des cartes Mifare, il est possible d'utiliser les outils du logiciel RFIDIOt.

Forfaits de ski

Les forfaits de ski sont utilisés pour le contrôle d'accès aux remontées mécaniques.

Selon la station de ski, le forfait utilise des basses fréquences (125 kHz) avec des protocoles propriétaires, ou peut être basé sur la norme définie par l'ISO 15693 ou l'ISO 14443. La technologie qui est privilégiée reste tout de même la première. Elle permet une lecture allant jusqu'à 1, 5 m de distance, l'ISO 14443 ne permet une lecture qu'à une dizaine de centimètres. Selon [10], seul un identifiant unique (UID) de 64 bits suffit pour le fonctionnement du système. Cependant, des données supplémentaires peuvent être stockées sur le forfait de ski afin de garantir un mode de fonctionnement hors ligne.

Le contenu de la mémoire est divisé en secteurs, chacun contenant quelques octets. Le nombre de secteurs et le nombre d'octets de chaque secteur varie en fonction de l'entreprise qui fournit le forfait de ski. Néanmoins, tous les dumps semblent contenir le même type d'information et semblent structurés de la même manière. Pour extraire les données des forfaits de ski, il est possible d'utiliser les outils de RFIDIOt.

1.2.4 Exemples de dumps

Cette section présente 5 dumps provenant d'applications différentes. Ces applications sont EMV (paiement), Calypso (transport), Vitale (santé), OV (transport) et un forfait de ski. Une partie des différents champs d'informations de chacune de ces applications sont décrits. Différentes manières de stocker et d'encoder les informations sont ainsi présentées.

EMV

```
1.  9F3602004D
2.  9F13020046
3.  9F170103
4.  70615F201A4A4F484E2F534D4954482E4D52202020202020202020202020205F300202018C1B9F02069F
    03069F1A0295055F2A029A039C019F37049F45029F4C088D1A8A029F02069F03069F1A0295055F2A029A
    039C019F37049F4C08
5.  701A5F25030911015F2403100331 5A0849750000075945345F340100
6.  70329F080200028F01069F3201039224 34592451B87DA8C05BA7F1DE5DC802BF59D394D6CC034A046F46
    995E0245E437AED7B899
7.  00000000135040025009781003260 0
8.  00000000177040025009781003260 0
9.  00000000209040025009781003250 0
10. 00000000770740025009781003240 0
```

FIGURE 1.8 – Extrait de la mémoire d'une carte de crédit

Ce standard est public, les informations stockées dans ces cartes devraient donc être interprétées facilement. En réalité, certaines libertés sont laissées aux concep-

teurs de l'application installée sur la carte. En dehors des données nécessaires à leur bon fonctionnement, les cartes bancaires des différentes banques ne stockent pas nécessairement les mêmes données. Par exemple, la taille de l'historique des transactions peut varier ou les transactions effectuées à l'étranger peuvent être stockées dans un historique séparé. Le contenu de l'EEPROM de la carte est divisé en fichiers, et chacun de ces fichiers peut comporter un ou plusieurs enregistrements. Par exemple, dans le fichier stockant les dernières transactions, les informations de chaque transaction se retrouvent dans un enregistrement différent.

La figure 1.8 présente l'extrait d'un dump d'une carte EMV. Chaque ligne numérotée représente un enregistrement de la carte. La plupart des enregistrements sont encodés à l'aide d'un champ TLV (Type-Length-Value). Sur cet extrait, les enregistrements 1 à 6 sont des champs TLV. Ainsi, chaque champ est défini par un *type* représentant la nature de la donnée (clé publique, nom du possesseur, etc.), par une *longueur* représentant la taille des données de la *valeur*, et enfin la *valeur* représentant l'information du champ. Il existe des outils en ligne permettant de décoder ces champs EMV, on peut citer l'outil de emvlab [11].

La ligne 1 est définie par le type `0x9F36`. Ce type représente le compteur des transactions effectuées par l'application. Sa longueur est de `0x02` octets. Sa valeur est donc `0x004D`, ce qui représente l'entier 77. De la même manière, la ligne 2 représente le numéro de la dernière transaction dite *online*. La ligne 3 donne le nombre d'essais restant pour le code PIN (ici 3).

La ligne 4 contient plusieurs champs TLV. La séquence soulignée dans cet enregistrement représente les données de la *valeur* du premier champ TLV, ce qui correspond au nom du possesseur de la carte de paiement. Les données de ce champ sont encodées en utilisant l'ASCII. La séquence binaire encodant le nom est complétée avec une répétition de l'octet 0x20 jusqu'à atteindre une certaine taille prédéfinie. Ceci s'explique par le fait que la longueur du champ d'information est la même dans tous les dumps EMV. Lorsqu'on décode ces données soulignées, on obtient "Mr John Smith".

La ligne 5 contient aussi plusieurs champs TLV. Les séquences soulignées représentent des dates encodées à l'aide de BCD (Binary Coded Decimal ou décimal codé binaire) au format `YYMMDD`. L'encodage BCD découpe la séquence en bloc de 4 bits, chaque bloc est converti en base décimal et représente un chiffre entre 0 et 9. Les données encodées avec l'encodage BCD sont lisibles par un humain lorsque les données sont affichées en utilisant la représentation hexadécimale. La première date correspond à la date du début d'utilisation de la carte (11/09/2003), et la deuxième date correspond à la date d'expiration de la carte (31/03/2010).

La ligne 6 contient des informations sur la clé publique utilisée pour vérifier la signature des données contenues dans la carte. Les données soulignées représentent les bits de poids fort du module de la clé. Les bits de poids faible sont stockés dans un autre enregistrement de la carte.

Les autres lignes (7 à 10) représentent 4 des dernières transactions effectuées avec la carte. Ces données ne sont pas encodées sous la forme de champs TLV. La première séquence soulignée représente le montant de la transaction en BCD (13, 50 pour le septième enregistrement), la deuxième séquence représente la date de la transaction toujours représentée par le format (ici, le 26/03/2010). La séquence `0x0250` représente le code ISO du pays encodé avec BCD où a eu lieu la transaction (ici, la France). Les données `0x0978` correspondent à la devise (ici, l'euro). Enfin, les derniers octets représentent le type de la transaction (ici, `0x0` un achat). Ce sont des enregistrements cycliques, chaque carte enregistre un nombre prédéfini de transactions. Lorsque l'historique est complet, les informations liées à la nouvelle transaction écrasent les informations de la plus ancienne.

Comme mentionné précédemment, ces dumps EMV sont relativement faciles à interpréter car les spécifications sont publiques, et l'utilisation des champs TLV facilite grandement l'identification des données.

Calypso

```
1. 00 00 00 00 00 00 00 04 00 71 B3 00 00 00 00 00 01 B8 B2 4A 02 50 00 33 01 1A 14 43 00

2. 04 00 98 E5 94 C8 02 0D 60 C9 65 C7 D5 90 00 00 00 00 00 00 00 19 75 08 10 92 82 D2 CF

3. F3 6A 68 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

4. 08 38 2B 00 08 BD 59 2A 46 60 C4 81 98 E5 94 C8 02 0D 60 C9 65 C6 41 F4 00 00 00 00 00
5. 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

6. 09 0E E5 92 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 C0 00 00 51 08 66 E0 00 00
7. 09 0E E5 7A 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 80 00 00 11 08 66 E0 00 00
8. 09 0E E5 5A 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 40 00 00 91 08 66 E0 00 00

9. 11 2B 40 01 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

FIGURE 1.9 – Extrait de la mémoire d'une carte Calypso

La figure 1.9 fournit un extrait d'un dump de mémoire d'une carte Calypso. Chaque ligne numérotée représente un enregistrement. Les spécifications de l'application Calypso définissent, entre autres, les fichiers contenus dans l'EEPROM de la carte. Pour chaque fichier, le standard décrit brièvement les informations qui peuvent

y être stockées. Les détails des informations stockées et leur encodage sont laissés à la discrétion de celui qui développe l'application (i.e., la compagnie de transport).

La ligne 1 contient des informations à propos de la fabrication de la carte : son numéro de série, l'identifiant de son fabricant, le pays où elle a été fabriquée et sa date de création. La première séquence soulignée représente le code ISO du pays du fabricant, ici `0x250` encodé avec l'encodage BCD représentant la France. La seconde séquence soulignée représente la date de fabrication. Pour décoder cette date, la séquence binaire est convertie en sa représentation décimale, ce nombre représentant le nombre de jours écoulés depuis le 01/01/1990, ici le 12/04/2008. L'année 1990 correspond au début du programme de développement qui donna lieu à Calypso par la suite.

Les lignes 2 et 3 contiennent des informations à propos du possesseur de la carte : son nom, son genre, son code postal, etc. La première séquence soulignée correspond à la date de naissance du possesseur, encodée avec l'encodage BCD représentée par le format de date `YYYYMMDD`, ici le 10/08/1975. Les données entre les deux séquences soulignées représentent le genre du possesseur. La deuxième séquence soulignée représente le nom du possesseur. Cette information est encodée avec la fonction de décodage 5-bit, ici "James Smith". La fonction de décodage 5-bit est une fonction qui utilise 5 bits pour encoder chaque lettre. Chaque bloc de 5 bits est converti sous sa représentation décimale, et représente l'index de la lettre dans l'alphabet.

Les lignes 6, 7 et 8 contiennent des informations à propos des trois derniers voyages effectués avec la carte. Chacun des enregistrements contenant les informations d'un trajet. Ces informations incluent la date, l'heure, le numéro de la ligne utilisée, la station, etc. Tout comme l'historique des transactions des cartes bancaires, cet enregistrement est cyclique. Lorsqu'un quatrième trajet est effectué, les informations à propos du trajet le plus ancien sont écrasées. Le nombre de trajets enregistrés par la carte peut varier selon l'application (la ville). Les données soulignées sur ces trois lignes représentent la date du trajet. Pour retrouver la date, la séquence binaire est convertie en sa représentation décimale. Ce nombre correspond au nombre de jours écoulés depuis le 01/01/1997, c'est à dire l'année de création de la première carte (CD97) issue du projet Calypso.

Les dumps de l'application Calypso sont plus difficiles à décoder car les spécifications ne sont pas publiques. De plus, les données ne sont pas présentées sous la forme d'un champ TLV. Les informations présentées dans cet exemple ont été décodées à partir d'investigations manuelles.

Vitale

```
1.  1001FFFF3FFFFFF00000000B0892FF174095012913909103
2.  5105061403100000007FFFFFFFF87FFFFFFFF19810423101
3.  71125FFFFFFFFFFFFFFFFFF6A4682B200000000001B2560F
4.  75000000000000000000000007FC5D025404A44E42440D4
5.  41418544494000001FFFFFFFE1FFFFFF20181125FFFFFFF
6.  FFFFFFFFFFFFFFFFFFFFFFFFFFFFF0D9E97140000000000
7.  0000000000000001FFFD14D44185515526514D91996024F
8.  FC0EC0ED1032D65024D40DA9925EC0ED149351502C840DA9
9.  B2C940935034A74D6C869CA5FFFFFFFFFFFFFFFFFFFFFFF
10. FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
11. FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
12. FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
13. FFFFFFFFF002016082020160820FFFFD0000000000000000
14. 00000000000000000000000000003FFFFFFFFFFFFFFFFFF
15. FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC568900014E23000
16. 52088FF10621C003F886FF0F42D7FC3A080FF0DA20800308
17. 00FF0B42D3FC2B8B3FF0AC3F20C8E2FC0AC387F02B0E1FC0
18. A23D3F0270E3FC0903C7F8CEEADFC0842A07D100AC20E9AD
19. 44BAD9FF0F
```

FIGURE 1.10 – Dump complet de la mémoire d'une carte Vitale

La figure 1.10 présente un dump complet d'une carte Vitale de première génération. Les données sont contenues dans des mots de 32 bits. Les deux premiers bits de ces mots sont des bits dits *systèmes* et doivent être ignorés pour le décodage. Le dump de cette carte est représenté ici sous sa forme hexadécimale, avec 24 octets par ligne.

La carte enregistre de nombreuses informations à propos du titulaire et des éventuels ayant-droits. Sur la ligne 2, les données soulignées représentent la date de naissance du titulaire de la carte. La date est encodée avec l'encodage BCD, représentée par le format `YYYYMMDD`. La date représentée ici est le 23/04/1981. Sur les lignes 3 et 4, les séquences soulignées correspondent aux nom et prénom du titulaire de la carte, encodés avec l'encodage 5-bit. Ici, ces valeurs représentent "Michel Dupont". Sur la ligne 7, les données soulignées représentent le numéro de sécurité sociale du titulaire de la carte, ces données ont été encodées à l'aide de UU (Unix to Unix Encoding). Cet encodage découpe les données en bloc de 6 bits, chaque bloc correspondant à un caractère ASCII. Ici, le numéro de sécurité sociale représente donc "1340851529139". Sur les lignes 8 et 9, la séquence soulignée correspond à l'adresse complète du titulaire de la carte (numéro, rue, code postal et ville), elle aussi encodée avec l'encodage UU. Ici, l'adresse complète est "1 Rue du vide 12345 La Ville du titulaire". Sur la ligne 13, les données soulignées représentent la date de dernière utilisation de la carte Vitale dans une pharmacie. Ces dates sont aussi encodées à l'aide de BCD et du format `YYYYMMDD`, représentant ici le 20/08/2016.

Toutes les cartes Vitale de première génération ne contiennent pas nécessairement les mêmes informations. Cela dépend de l'existence d'éventuels ayant-droits ou des événements médicaux survenus. De même, pour l'adresse complète du titulaire, elle aussi n'est pas nécessairement présente dans toutes les cartes. Les cartes de deuxième génération contiennent des données supplémentaires. À noter qu'elles incluent un fichier reprenant les données de la carte Vitale de première génération. Ajouté à cela, elles stockent des objets cryptographiques utilisés pour l'authentification et le contrôle d'intégrité des données.

Il n'existe aucune version publique des spécifications de la carte Vitale. Les documents officiels indiquent que certaines informations peuvent être stockées dans la carte [12, 13]. Cependant, il n'y a pas de garantie qu'elles le soient vraiment et cela ne définit pas si elles sont protégées en accès ni comment elles sont encodées. Une investigation manuelle a donc été nécessaire pour décoder les informations présentées dans cet exemple.

OV-chipkaart

```
1.  04B3FBC4
2.  22F83380
3.  694800F0
4.  C9CDFFFE
5.  C0003004
6.  CD8CD521
7.  069F507F
8.  A98A5318
9.  C8002004
10. 2D8CC910
11. 24C8FB5D
12. DCC5941D
13. 7951631E
14. 84FD574D
15. CCBA5DDB
16. 58463DA8
```

FIGURE 1.11 – Dump complet de la mémoire d'une carte OV

Les tickets OV-chipkaart sont utilisés pour les transports publics aux Pays-Bas. La figure 1.11 fournit un exemple complet du dump d'un ticket de train. Ces tickets utilisent des cartes Mifare Ultralight.

Les 7 premiers octets du dump représentent l'UID (Unique Identifier) de la carte. Plus généralement, les deux premières lignes des cartes Mifare Ultralight contiennent toujours l'UID. La séquence soulignée sur la quatrième ligne représente la nature du voyage : aller simple ou aller-retour.

La séquence soulignée sur la ligne 6 représente la date et l'heure de la validation du trajet aller. Les deux premiers octets représentent la date. La séquence binaire

est convertie dans sa représentation décimale correspondant au nombre de jours écoulés depuis le `01/01/1997`. Les deux derniers octets représentent l'heure du trajet. La séquence binaire est convertie dans sa représentation décimale, correspondant au nombre d'heures écoulées depuis minuit. La séquence soulignée sur la ligne 10 représente aussi une date et une heure. Mais, cette fois-ci, pour le voyage retour. Les données sont encodées de la même manière.

Ces fonctions d'encodage sont les mêmes que pour les dumps Calypso. Cependant, il n'existe pas de spécifications accessible publiquement des données de ces dumps. Les informations ont donc été retrouvées à l'aide d'une investigation manuelle.

1.3 Techniques d'analyse forensique

La forensique informatique rassemble les techniques permettant l'extraction et l'analyse des supports numériques. Comme mentionné précédemment, cette thèse s'intéresse à l'analyse des données et non à l'extraction. Ainsi, seules les techniques d'analyse forensique seront développées dans cette section. L'objectif de la forensique informatique est de retrouver des preuves numériques liées à des affaires criminelles dans les supports analysés. Ces preuves peuvent être la récupération d'informations (e.g., mails échangés, images supprimées) ou la reconstruction d'une série d'événements (e.g., escalade de privilège d'un utilisateur, création de compte, changement de valeur pour une clé de registre). Une analyse forensique peut être requise dès lors qu'un support numérique est disponible dans une affaire criminelle. Ces supports numériques incluent (mais ne sont pas limités à) les ordinateurs, les téléphones, les cartes à puce, les objets connectés, mais aussi le cloud. La démocratisation de l'utilisation de supports numériques implique que les analyses forensiques sont de plus en plus sollicitées. Par exemple, lorsqu'un système est victime d'une intrusion (e.g., TV5 Monde), lorsqu'une personne est soupçonnée d'activité frauduleuse (e.g. possession de documents pédopornographiques), lorsqu'une personne est portée disparue, etc.

Cette section commence par une brève chronologie de l'analyse forensique allant de ces prémices jusqu'aux futurs challenges. Ensuite, elle décrit les différentes approches proposées et les outils disponibles pour analyser les différents supports numériques. L'accent est mis sur les rares travaux effectués sur l'analyse des mémoires non volatile des cartes à puce. Enfin, une discussion est proposée mettant en évidence que les techniques existantes d'analyse forensique ne permettent pas de répondre efficacement à la problématique de cette thèse. Cette section démontre aussi que les méthodes proposées dans cette thèse s'inscrivent bien dans les futurs challenges proposés par la communauté forensique.

1.3.1 Analyse forensique : historique et challenges du futur

À la fin des années 80 et durant les années 90, l'analyse forensique consistait à récupérer des fichiers ayant été supprimés sur un système ou sur un matériel endommagé. À cette époque, il n'existait pas de méthodologie ou d'outils permettant d'effectuer cette tâche. Les cas étaient traités de manières ad-hoc par des informaticiens.

Les années 2000 sont considérées comme l'âge d'or de l'analyse forensique. Durant cette période, les supports numériques sur lesquels étaient effectuées les analyses n'étaient pas très variées, que ce soit d'un point de vue logiciel ou matériel. Typiquement, l'analyse se fera sur un disque dur (IDE/ATA) d'un ordinateur utilisant le système d'exploitation Windows XP. La plupart des analyses se focalisent sur des fichiers contenant des images (JPEG) ou des vidéos (AVI, WMV). Ainsi, on voit apparaître les premiers logiciels commerciaux performants permettant de récupérer ces fichiers.

À la fin des années 2000, des difficultés se font ressentir. En effet, les tenants et les aboutissants de l'analyse forensique évoluent, les analyses incluent dorénavant l'analyse de systèmes ayant subi une attaque. Les supports ne se limitent plus seulement à des disques durs isolés mais à un ensemble de supports d'origines variés (e.g., téléphones), potentiellement chiffrés et dont la taille ne cesse d'augmenter. De plus, les logiciels malveillants ne laissent plus de traces dans la mémoire morte mais seulement dans la mémoire volatile. On assiste aussi à une explosion du nombre d'applications et de types de fichiers à analyser. De ce fait, les outils doivent être mis à jour sans arrêt, augmentant leur coût et complexifiant la tâche des enquêteurs qui doivent posséder un arsenal complet de ces logiciels.

Aujourd'hui, les outils souffrent à cause de ces difficultés. Les outils ont surtout pour objectif de retourner des preuves à partir du support numérique. Dans le cas où l'environnement diffère, même légèrement, du cadre défini par l'outil, ce dernier est souvent incapable de ressortir des informations. Ainsi, il serait préférable de définir des outils guidant l'analyse plutôt que de rechercher des preuves spécifiques. Un autre problème est l'interopérabilité des outils et méthodes existantes, chaque avancée est faite de manière ad-hoc et il n'existe aucun standard permettant de comparer les résultats obtenus. De plus, les recherches effectuées par la communauté scientifique atteignent rarement les outils commerciaux utilisés par les spécialistes.

Selon [14], les challenges auxquels doit répondre la communauté scientifique de l'analyse forensique sont les suivants : (i) être capable de traiter des volumes importants de données, (ii) effectuer une analyse intelligente permettant de guider l'enquêteur plutôt que de rechercher des preuves spécifiques, (iii) s'intéresser à des environnements non-standards, la mémoire non volatile des cartes à puce en est un

exemple, et enfin (iv) développer des outils d'analyse forensique.

Plus de détails sur l'évolution de l'analyse forensique sont disponibles dans ces deux papiers faisant le bilan des recherches effectuées par la communauté scientifique [15], [14].

1.3.2 Forensique sur la mémoire morte des ordinateurs

Il existe deux approches différentes pour analyser la mémoire morte des ordinateurs. La première est basée sur l'analyse des fichiers et la deuxième sur l'analyse des données brutes. Ces mémoires mortes peuvent être de différents types : magnétique (HDD), flash (SD), électronique (SSD) ou encore logique comme sur le cloud. Une des principales difficultés est la taille des supports qui ne cesse d'augmenter, les disques durs des ordinateurs personnels contiennent des centaines de giga-octets de données et il n'est pas rare qu'ils contiennent des tera-octets de données.

Approche basée sur les fichiers

Plus communément appelée *file carving*, cette approche consiste à retrouver tous les fichiers présents sur le disque dur. Les fichiers qui peuvent être retrouvés incluent des images (JPEG, PNG, etc.), des vidéos (AVI, MP4, etc.), des documents (PDF, Word, etc.), des archives (ZIP, RAR, etc.), des courriers électroniques (Outlook), etc. Les données de chacun de ces types de fichiers commençant généralement par des octets spécifiques, il est facile de les détecter. Par exemple, les fichiers JPEG commencent toujours par les octets `0xFFD8`.

La principale difficulté pour récupérer ces fichiers réside dans le fait que les données sont souvent stockées de manière non contiguë sur le disque dur. Cela est causé par la fragmentation de l'espace de stockage. Ainsi, le *file carving* récupère les différents fragments sur le disque et les réassemble. Les méthodes utilisées sont des outils statistiques et des techniques d'apprentissage automatique. On peut retrouver une synthèse de l'état de l'art de ces techniques dans [16].

L'avantage de ces techniques est que les résultats obtenus reflètent l'état actuel du système de fichiers, et sont donc facilement compréhensibles par un enquêteur. Par contre, elles ne permettent pas de retrouver les données qui ne seraient pas contenues dans un type de fichier connu. On peut citer comme outil *Foremost* [17] ou *PhotoRec* [18].

Approche basée sur les données brutes

Une autre approche est basée sur les données brutes contenues dans la mémoire. Plutôt que de rechercher des fichiers, cette dernière recherche des informations. Celles-ci peuvent être de différents types, adresses électroniques, numéros de cartes de crédit, numéros de téléphone, chaînes de caractères, etc. Dans [19], Garfinkel propose une méthode pour effectuer de manière efficace cette approche. Un outil, *bulk extractor*, implémentant cette méthode est disponible ici [20].

Rechercher des adresses électroniques dans la mémoire morte d'un ordinateur personnel générera potentiellement des milliers de résultats. Par exemple, les fichiers systèmes contiennent de nombreuses adresses électroniques relatives aux informations de contact de ceux qui les ont développés. Afin de retourner des résultats pertinents, il est donc nécessaire de trier les données retournées par cette recherche. Néanmoins, il faut aussi être vigilant dans la manière de les trier pour qu'un utilisateur malveillant ne profite pas de ces règles de triage. Pour cela, *bulk extractor* propose d'ignorer certaines informations en fonction de leur contexte. Par exemple, les adresses électroniques appartenant à un certain domaine contenues dans les fichiers systèmes seront ignorées.

L'avantage de cette technique est qu'elle peut être appliquée sur tout type de mémoire, disque dur, traces réseaux, fichier binaire, téléphone, etc. Elle est aussi capable de récupérer des informations contenues dans des formats de fichiers inconnus. Le problème avec cette technique est la quantité d'informations retournées, les plus pertinentes peuvent être noyées parmi une grande quantité d'informations peu intéressantes. Cela se vérifie même sur des dumps de quelques kilo-octets seulement, et c'est d'autant plus vrai lorsque différents types d'encodage sont considérés pour les retrouver.

1.3.3 Forensique sur la mémoire volatile des ordinateurs

L'analyse forensique des mémoires volatiles est plus récente que l'analyse forensique des mémoires mortes. Néanmoins, ces nouvelles techniques permettent de retrouver certaines informations qui ne sont pas disponibles dans la mémoire morte. En effet, les logiciels malveillants récents ont tendance à ne seulement laisser des traces dans la mémoire volatile.

L'analyse forensique sur les mémoires volatiles (de type RAM) des ordinateurs consiste à analyser les dernières actions effectuées par l'utilisateur et les programmes sur le système. Ces techniques analysent les connexions réseaux ouvertes (source, destination, port, etc.), les processus en cours d'exécution ou terminés, (chemin de l'exécutable, père du processus, etc.), les historiques de certaines applications

(navigateur web), les clés de registre (mots de passe Windows, programmes qui s'exécutent au démarrage, etc.), les utilisateurs connectés, les fichiers ouverts, etc.

Pour analyser cela, les techniques de forensique recherchent des chaînes de caractères spécifiques ou des signatures, interprètent des structures internes du noyau ou énumèrent et corrèlent différentes pages mémoires inhérentes aux processus. Durant les 10 dernières années, les recherches effectuées ont été implémentées dans des outils open-source tel que Volatility [21], Rekall [22], ou encore FATKit [23]. C'est un fait assez rare pour être souligné. Notons que, les techniques d'analyse des mémoires mortes peuvent aussi être appliquées sur ces dumps de mémoire pour retrouver des fichiers ou des informations chargés en RAM.

Il existe aussi des outils de type *Internet Evidence Finder* capables de combiner les analyses de la mémoire volatile et non volatile d'un ordinateur. Ces outils recherchent toutes les informations liées à l'utilisation d'internet : messagerie instantanée, courriers électroniques, historique des navigateurs, analyse des traces réseaux, réseaux sociaux, etc. IEF de Tracip [24] et Magnet IEF [25] en sont des exemples.

1.3.4 Forensique sur les mémoires des téléphones

L'analyse forensique des mémoires de téléphone mobile est aussi plus récente que celle des mémoires des ordinateurs. Sur un téléphone, il existe plusieurs types de mémoires : la carte SIM, les mémoires interne (flash) et externe (carte SD), ainsi que la mémoire volatile (RAM). L'analyse forensique sur téléphone consiste à retrouver le même type d'informations que sur les mémoires des ordinateurs. Néanmoins, il faut ajouter à cela certaines informations plus spécifiques à la téléphonie ou au système de géolocalisation. Une des difficultés qui s'ajoute est le fait que les technologies des téléphones mobiles ne cessent d'évoluer. Par conséquent, les données qui y sont stockées et la manière dont elles le sont évoluent aussi. C'est le cas des systèmes d'exploitation et des applications mobiles telles que les réseaux sociaux ou les messageries instantanées. Les techniques proposées pour les analyses forensiques des mémoires d'ordinateurs ne sont donc pas suffisantes.

Les solutions fournies sont donc souvent destinées à une application en particulier comme [26] pour les applications de messagerie instantanée, ou [27] pour les applications de réseaux sociaux. Les solutions les plus génériques sont des solutions commerciales telles que la valise Cellebrite [28] ou Susteen Secure View [29]. Néanmoins, ces solutions ne sont qu'une agrégation de très nombreuses techniques ad-hoc.

1.3.5 Forensique sur l'EEPROM

L'analyse forensique des mémoires EEPROM des cartes à puce diffèrent des autres analyses forensiques présentées précédemment. Certaines difficultés s'ajoutent alors : (i) les mémoires des cartes à puce sont peu structurées, différentes informations sont stockées côte à côte, encodées de manières différentes, (ii) il n'y a pas de système de fichiers ou de types de fichiers qui incluent des méta-données exploitables comme pour les fichiers JPEG, (iii) les dumps obtenus dans le contexte de la thèse ne sont pas une copie bit-à-bit des mémoires EEPROM, ils correspondent aux données accessibles avec l'API de la carte. Cependant, ces différences peuvent aussi être avantageuses : (i) il est possible d'effectuer une analyse multi-dump en considérant plusieurs dumps provenant de la même application ou de la même carte, (ii) des informations contextuelles peuvent être exploitées (e.g., informations imprimées sur un ticket), (iii) les dumps sont généralement longs de quelques kilo-octets seulement.

Quasiment toutes les contributions existantes sur l'analyse forensique des mémoires de cartes à puce considèrent des analyses manuelles et ad-hoc. Par exemple, c'est le cas pour les travaux de Avoine et al. [30] qui se sont intéressés au pistage des passagers utilisant des tickets de voyage anonymes dans un transport public. Un autre exemple, Lanet et al. [31] retrouvent la zone contenant les sections *code* et *data* d'une applet *javacard* dans le dump de mémoire EEPROM d'une carte à puce. Pour cela, ils utilisent l'indice de coïncidence (IC) en utilisant une fenêtre glissante sur l'ensemble du dump. Ils recherchent la zone de la mémoire qui correspond le plus à un IC préalablement calculé à partir de binaires *JAVA card*. Cette analyse est complétée par certaines heuristiques, comme le fait qu'un exécutable ne contienne pas de NOP, et que certains opcodes n'existent pas. Une fois que ces zones sont localisées, ils proposent de reconstruire le fichier CAP (jar contenant l'ensemble des modules à charger) correspondant à l'applet.

Une exception est le travail de T.Van Deursen, S.Mauw, et S.Radomirović [32], qui ont exploré le problème de l'analyse forensique pour des ensembles de dumps de mémoire, et ont appliqué leur méthode sur des tickets de transport public. Ils proposent de localiser automatiquement l'emplacement où les informations peuvent être stockées dans le dump. Pour cela, les auteurs éliminent les zones mémoires où les informations ne peuvent pas être stockées, en fonction des données des dumps et des relations entre les dumps collectés. Cependant, leur travail ne propose pas une interprétation automatique des données. En effet, pour les interpréter, ils ont analysé ces dernières manuellement aux emplacements retenus. Pour cela, ils se sont aidés des informations imprimées sur les tickets.

Il existe aussi un outil nommé Cardpeek [2] qui permet d'interpréter automati-

quement mais partiellement certains dumps d'applications connues (Vitale, EMV, SIM, Calypso). Cet outil utilise des scripts qui vont décoder de manière automatique les données présentes dans un dump de mémoire et les présenter à l'utilisateur. Ces scripts ont été proposés par le créateur de Cardpeek. Ils ont probablement été écrits en faisant une analyse manuelle préalable de la mémoire de ces applications et en utilisant leurs spécifications. Cette analyse est donc seulement automatisée pour les applications étudiées au préalable. De plus, ces analyses ne sont pas toutes nécessairement complètes.

D'autres analyses manuelles ont été effectuées comme par exemple les ingénieurs et chercheurs du GSI de l'Université Catholique de Louvain qui ont analysé le contenu des cartes MOBIB. Ces cartes utilisent le standard Calypso et sont utilisées dans la ville de Bruxelles. On peut aussi citer les travaux effectués par Patrick Gueulle sur la carte Vitale [33].

1.3.6 Discussion

L'objectif est de récupérer les informations stockées dans les mémoires EEPROM des cartes à puce. Aucune des techniques présentées ci-dessus ne répond de manière satisfaisante à la problématique. Bien que les techniques pour l'analyse des mémoires mortes d'ordinateur peuvent être utiles dans le contexte de cette thèse, elles ne sont pas suffisantes. En effet, dans certains cas, il peut être intéressant de rechercher des fichiers comme des images JPEG dans l'EEPROM. Par exemple, le passeport stocke la photo du titulaire et une image de la signature manuscrite du titulaire. Pour autant, cela ne répond qu'à une part peu significative du problème. D'autre part, l'approche de *bulk extractor* ne procède pas à une élimination suffisante des faux positifs. Les informations recherchées sont les informations ayant un format particulier, par exemple, une adresse de courrier électronique. D'autant plus que dans le contexte de cette thèse, différents encodages doivent être considérés, ce qui n'est pas le cas dans *bulk extractor*. Les résultats obtenus ne sont donc pas exploitables. Néanmoins, c'est un des rares outils permettant une analyse générique d'un dump de mémoire. Les méthodes proposées dans cette thèse sont similaires au principe de *bulk extractor* mais considèrent plusieurs encodages possibles et proposent une réelle technique d'élimination des faux positifs pour les informations sous forme de dates et de chaînes de caractères.

Concernant les analyses des mémoires volatiles, elles sont trop spécifiques pour être appliquées dans le contexte de cette thèse. Les informations recherchées par ces techniques ne sont pas contenues dans l'EEPROM. Par exemple, ce n'est pas l'EEPROM qui enregistre les connexions réseaux, les processus ouverts, etc.

Les analyses forensiques existantes sur les mémoires EEPROM des cartes à puce n'ont pas pour objectif d'interpréter de manière automatique et générique les informations contenues dans ces mémoires, elles ne sont donc pas suffisantes.

La quasi-totalité des techniques et des outils d'analyses présentés ici ne sont pas génériques. En effet, soit ils utilisent les spécifications de l'application ou du type de fichier ciblé, soit, (et c'est le cas pour la plupart) ils profitent d'une rétro-ingénierie effectuée au préalable. Cette rétro-ingénierie doit être faite au cas par cas, pour chaque technologie ciblée et pour chacune des versions existantes. Il serait possible de produire des outils similaires pour les dumps de mémoire des cartes à puce dans la continuité de *Cardpeek*. En effet, les spécifications des passeports ou des cartes EMV sont publiques. Il est aussi possible d'analyser des applications au cas par cas, par exemple, en collectant de nombreux dumps d'une compagnie de transport et en faisant une rétro-ingénierie spécifique à cette application. De telles méthodes ont un intérêt limité car le nombre d'applications utilisant des cartes à puce est très important et l'analyse de chaque application demande un temps conséquent (plusieurs jours au minimum). Ainsi, cette thèse propose un ensemble de méthodes permettant d'effectuer l'analyse forensique des mémoires non volatiles des cartes à puce indépendamment de leur application d'origine. Ces méthodes ont pour objectif de guider l'enquêteur face au contenu du dump de mémoire plutôt que de retourner des preuves spécifiques. De ce fait, cette thèse répond directement aux challenges posées par la communauté de l'analyse forensique en s'intéressant à de nouveaux supports (mémoire non volatile des cartes à puce), et en proposant des méthodes génériques guidant l'enquêteur humain.

1.4 Spécificités des dumps de mémoire des cartes à puce

Cette section présente deux spécificités des cartes à puce qui facilitent l'élimination des faux positifs générés par le décodage lors du recouvrement des informations dans la mémoire des cartes à puce.

1.4.1 Similarités au sein d'une même application

La figure 1.12 présente 4 dumps de l'application OV-chipkaart qui sont les tickets utilisés dans les transports public néerlandais. La section 1.2.4 indique que les données de la ligne 6 du dump donné en exemple correspondent à la date et l'heure du trajet. Dans ce nouvel exemple, on constate que les données de la ligne 6 de chaque dump

	Dump 1	Dump 2	Dump 3	Dump 4
1.	04B3FBC4	0400FD71	04CCFBBB	04C3307F
2.	22F83380	22F83381	22F83380	02D13E80
3.	694800F0	684800F0	694800F0	6D4800F0
4.	C9CDFFFE	C9CBFFFE	C9CDFFFE	C9C9FFFF
5.	C0003004	C8001004	C0003004	C8001004
6.	CD8CD521	0D8D3220	CD8CD521	0D8DB8A0
7.	069F507F	AE8F6350	FB3F7464	75F34FEA
8.	A98A5318	B1948307	70E99A45	C2469EB1
9.	C8002004	C8002004	C8002004	C8002004
10.	2D8CC910	2D8D3220	2D8CC920	2D8DB8F0
11.	24C8FB5D	249A056B	3504F2E4	5A5BB2C6
12.	DCC5941D	2060C0A1	A0E1E2B5	E199E498
13.	7951631E	3F8721D7	1AB94AAE	74D1AB2D
14.	84FD574D	91AC762B	7BB80E87	54DA1A3D
15.	CCBA5DDB	AFAB796F	AB5BCF09	90CFA9B3
16.	58463DA8	E316699C	243F1032	87B76450

FIGURE 1.12 – 4 tickets de train de l'application OV

correspondent toujours à la date et l'heure du trajet. Cette remarque se généralise pour toutes les informations stockées dans les dumps de l'application OV. Ainsi, une information est toujours stockée au même endroit dans un dump. On observe donc que les dumps de l'application OV partagent la même structure de mémoire : ils contiennent les mêmes champs de données et leurs champs sont stockés au même endroit. Seules les données contenues dans ces champs d'informations varient pour chaque dump.

De manière générale, les cartes se comportant comme des cartes mémoires (Mifare et forfaits de ski), les cartes bancaires, les cartes Calypso et les passeports électroniques partagent la même structure de mémoire lorsqu'elles proviennent du même opérateur (i.e., même ville, même station, même banque, etc.). La structure de la mémoire n'est que partiellement partagée pour ces cartes provenant d'opérateurs différents. Ces cas sont détaillés ci-dessous.

C'est le cas par exemple des forfaits de ski de deux stations différentes ou des cartes Calypso de deux villes différentes. Ces cartes vont stocker les mêmes informations avec les mêmes encodages, mais, pas nécessairement au même endroit dans le dump. Les cartes de transport provenant d'opérateurs différents ne vont pas non plus obligatoirement avoir la même taille d'historique concernant les derniers trajets effectués par l'utilisateur.

Les cartes Moneo et les cartes EMV peuvent contenir des fichiers différents selon leur banque d'origine et le type de carte. Cependant, la plupart des informations sont encapsulées dans des champs de type TLV (Type-Longueur-Valeur) permettant de les identifier. Les enregistrements qui contiennent des informations qui ne sont pas encapsulées des champs EMV ne sont pas nombreux et sont identifiables à l'aide de

leur longueur, c'est le cas des enregistrements contenant l'historique des transactions EMV. Ainsi, il n'est pas difficile de reconstruire des dumps ayant la même structure de mémoire.

Les cartes Vitale de première génération possèdent un fichier de données contenant diverses informations. Toutes ces informations ne sont pas toujours présentes dans toutes les cartes. De plus, elles ne sont pas stockées au même emplacement dans chaque carte. Néanmoins, les informations qui sont stockées sont toujours encodées de la même façon et les données relatives à la carte Vitale de deuxième génération sont encapsulées dans des données TLV.

Les dumps de passeports électroniques de différents pays ne stockent pas nécessairement les mêmes fichiers. Cependant, ils sont facilement identifiables les uns des autres. De plus, certains fichiers ne contiennent pas exactement les mêmes données selon le pays émetteur du passeport. Néanmoins, les encodages utilisés sont les mêmes pour tous les passeports.

Un autre cas problématique survient lorsque l'on traite des informations ayant une longueur variable. Par exemple, le nom du possesseur a une taille qui varie en fonction de la carte traitée. Il existe deux cas possibles : (i) l'espace réservé pour le nom du possesseur est fixe pour tous les dumps de la carte, ainsi l'espace non utilisé est rempli par une valeur par défaut (*padding*), (ii) il existe une (ou plusieurs) valeur(s) d'octet(s) fixée(s) représentant un séparateur entre les différents champs d'information.

Ainsi, même si pour certaines applications il existe des différences, il est possible pour la grande majorité d'entre elles de reconstruire des dumps ayant la même structure de mémoire. La suite de ce travail se base donc sur l'hypothèse que plusieurs dumps appartenant à la même application partagent la même structure de mémoire. On peut noter que, deux dumps provenant de la même carte appartiennent aussi à la même application.

Cette hypothèse peut être exploitée pour réduire le nombre de faux positifs générés par le décodage. Soient D_1 et D_2 deux dumps appartenant à la même application, appliquer une fonction de décodage à un indice donné peut générer une information sur le dump D_1 tout en n'en générant pas sur le dump D_2. Grâce à l'hypothèse, l'information générée sur le dump D_1 est nécessairement un faux positif. Ainsi, analyser simultanément les sorties obtenues par l'application de la fonction de décodage sur plusieurs dumps d'une même application permet d'éliminer ce type de faux positifs.

Il reste toutefois possible que des dumps ne possèdent pas la même structure de mémoire et qu'il soit trop complexe de pouvoir modifier ces dumps pour obtenir des

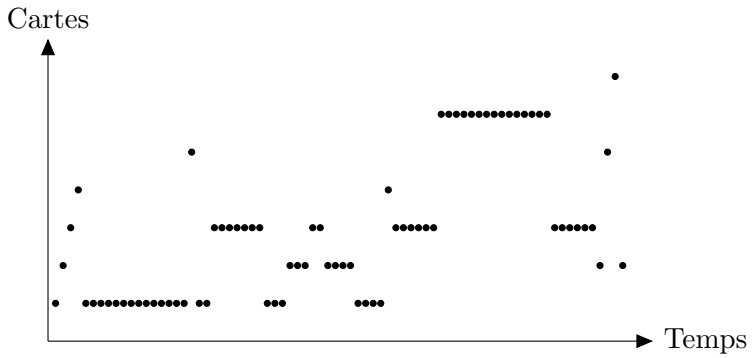


FIGURE 1.13 – Représentation des 75 dumps Calypso en fonction de leur possesseur et de leur date de création

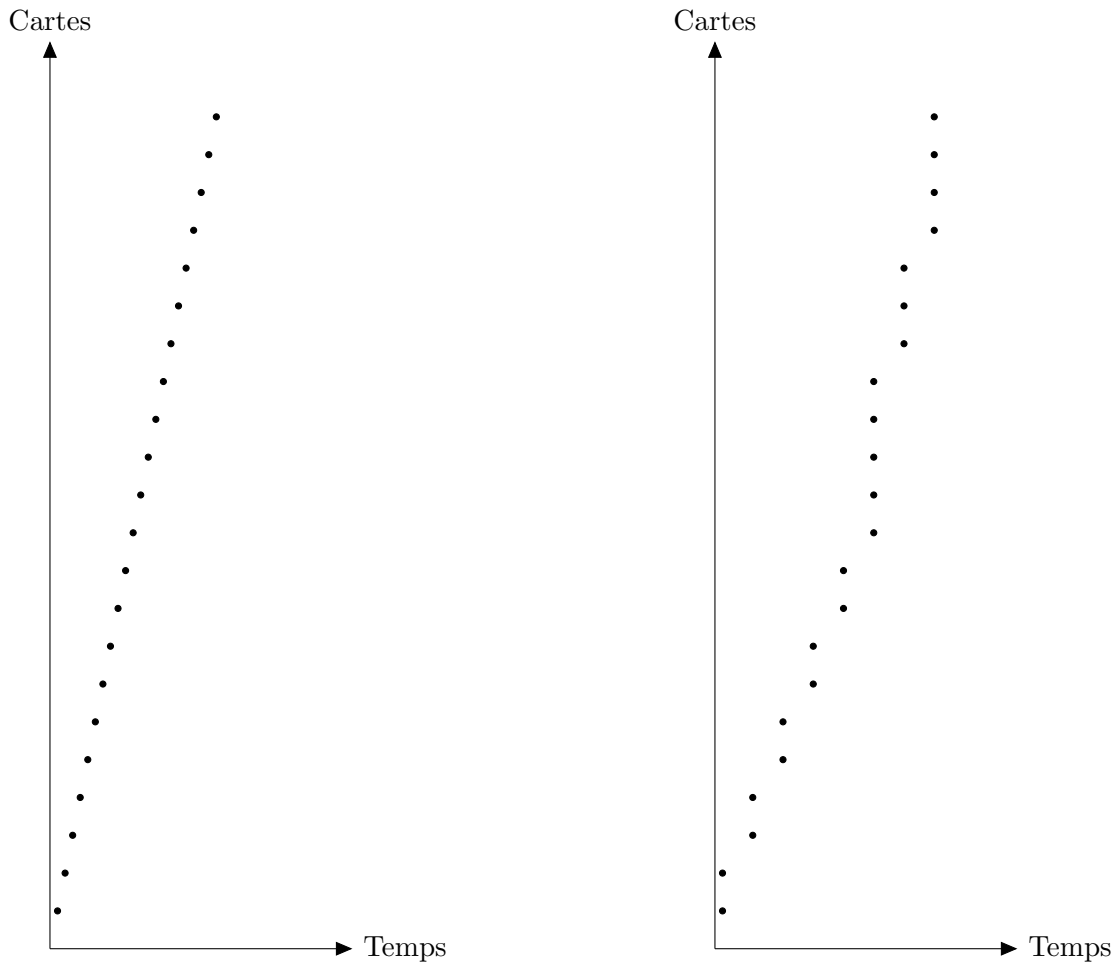
dumps ayant la même structure de mémoire. C'est le cas par exemple des dumps de carte Vitale de première génération. Dans ce cas, la méthode proposée exploitant plusieurs dumps au sein d'une même application ne pourrait pas être utilisée.

1.4.2 Informations contextuelles

Les informations contextuelles regroupent toutes les informations additionnelles qui sont reliées à un dump de mémoire. On peut les considérer comme les méta-données du dump. Ces informations peuvent être liées à la carte, telles que la connaissance de l'application ou la carte d'origine du dump. Elles peuvent aussi être liées à l'utilisation de la carte : la date de dernière utilisation, les dates de début et de fin d'un abonnement, etc. Il existe une dernière catégorie d'informations contextuelles, celles qui sont liées au possesseur de la carte : ses nom, prénom, adresse, etc. Les deux premiers types d'informations permettent de relier les dumps entre eux et des les visualiser sur deux axes représentant la carte d'origine du dump et le temps.

Ainsi, la figure 1.13 représente cette visualisation sur un cas réel. Elle présente 75 des dumps Calypso collectés en fonction de leur date de création et de leur possesseur. Ils proviennent tous du même opérateur (i.e., de la même ville). Chaque dump est représenté par un point sur le graphique. Les dumps proviennent de 7 cartes appartenant à 7 possesseurs différents représentés sur l'axe y (i.e., les dumps affichés sur la même ligne proviennent de la même carte). Notons que, chaque carte a été utilisée dans un transport public, au moins une fois, entre chaque création de dump. L'échelle des abscisse ne respecte pas l'intervalle de temps qui s'est écoulé entre chaque récupération de dump, seulement leur ordre chronologique.

La figure 1.14 représente deux visualisations des informations contextuelles pour l'application OV. Chaque carte OV est un ticket à usage unique et chacun des dumps



(a) Représentation chronologique (b) Granularité quotidienne.

FIGURE 1.14 – Représentation sous forme de grille des 22 dumps OV

de mémoire a été obtenu au moment de l'achat de la carte. La figure 1.14b représente les dumps avec une granularité différente pour le temps, deux dumps obtenus le même jour se voient attribuer la même abscisse. Cette représentation est intéressante lorsque l'on recherche la date d'achat du ticket, en supposant que l'heure et la date soient stockées de manière séparée dans le dump.

Les informations contextuelles peuvent être exploitées afin de définir plus précisément les informations recherchées dans le dump et ainsi éliminer les faux positifs qui ne correspondent pas aux informations contextuelles collectées. Par exemple, des informations à propos de l'utilisation de la carte sont utiles pour la recherche des dates. Toutes ces informations ont aussi été utiles afin d'évaluer la qualité des méthodes proposées dans cette thèse.

Il est aussi à noter que toutes ces informations contextuelles ne doivent pas

nécessairement être précises, par exemple, savoir qu'une carte à été utilisée sur un intervalle d'un mois plutôt que le jour exact, reste une information précieuse. A noter qu'une partie de ces informations contextuelles peut se retrouver imprimée sur le dispositif, par exemple, le nom du propriétaire et la date de validité de la carte sont imprimés sur une carte bancaire.

Chapitre 2

Séparer le bon grain de l'ivraie

Les objets cryptographiques dans les dumps de mémoire compliquent les techniques d'analyse forensique. Une méthode distinguant les données informationnelles des données cryptographiques dans des dumps de mémoire de petite taille est alors proposée. Elle utilise une technique d'apprentissage automatique de caractéristiques calculées à partir de tests statistiques. Expérimentée sur des cartes EMV, des passeports ou encore des cartes Vitale, cette méthode permet de reconnaître correctement plus de 94% *des données cryptographiques et* 98.5% *des données informationnelles.*

Sommaire

2.1 Introduction

Les dumps provenant de mémoire non volatile de cartes à puce peuvent stocker des objets cryptographiques. Ces objets peuvent être des données chiffrées, des données hachées, des clés cryptographiques, des signatures, etc. Aucune information personnelle ne peut être obtenue à partir de ces objets si l'on considère que les algorithmes cryptographiques utilisés sont sûrs. Appliquer la technique de décodage exhaustif sur ces données ne permettra donc pas de retrouver d'informations mais

générera un grand nombre de faux positifs. En plus des objets cryptographiques, les dumps peuvent contenir des données *informationnelles*, ces données représentent toutes les données qui ne sont pas cryptographiques telles que des informations personnelles (texte, date, etc.) et des informations techniques (identifiant, compteur, etc.). Ce chapitre propose donc une méthode qui *sépare le bon grain de l'ivraie*. Cette séparation du grain (des données *informationnelles*) de l'ivraie (des données *cryptographiques*) représente une étape préliminaire dans le processus de recouvrement d'informations.

Les objets cryptographiques présents dans les dumps sont considérés comme étant des données aléatoires. Il est donc envisageable d'utiliser les techniques d'évaluation des générateurs de nombres pseudo-aléatoires (PRNG) pour les identifier dans les dumps. Malheureusement, la taille des dumps considérés ne permet pas d'utiliser directement les outils classiques tels que les tests statistiques de la suite du NIST [34]. Ces outils nécessitent habituellement plusieurs kilo-octets de données pour rendre les tests statistiques pertinents. De plus, les tests statistiques ne peuvent pas être appliqués directement sur le dump entier car ils contiennent différents champs de données qui doivent être analysés séparément. Les tests doivent donc être appliqués sur des séquences de quelques centaines de bits au maximum. Il est aussi envisageable d'utiliser des méthodes localisant les clés cryptographiques dans des dumps de mémoire. Cependant, la plupart de ces techniques sont trop spécifiques à la recherche des clés et ne peuvent pas s'adapter à l'identification des autres objets cryptographiques.

Il est donc proposé une méthode distinguant les données *informationnelles* des données *cryptographiques* dans un dump de mémoire non volatile de carte à puce. Dans la suite de ce chapitre, les données *cryptographiques* sont appelées données de classe C et les données *informationnelles* sont appelées données de classe M. La méthode proposée combine une analyse statistique avec de l'apprentissage automatique. Elle est ensuite améliorée par une analyse multi-dump. Premièrement, des tests statistiques sont appliqués sur des dumps où la *vérité terrain* est connue. La vérité terrain représente la classification théorique des données du dump dans les classes M et C. Les résultats de ces tests sont ensuite analysés avec une technique d'apprentissage automatique appelée *boosting* [35]. Cette technique permet de sélectionner les tests les plus pertinents à appliquer sur les dumps dont la vérité terrain est inconnue. Enfin, les résultats de ces tests statistiques sont améliorés avec l'analyse multi-dump. Cette dernière combine les résultats obtenus sur plusieurs dumps de différentes cartes à puce provenant de la même application.

La méthode proposée atteint un taux de succès élevé : elle obtient plus de 95% de taux de reconnaissance lorsqu'elle est appliquée sur la base de 371 dumps de la

vie réelle du chapitre 1. Ces derniers proviennent de différentes applications telles que les cartes bancaires ou les passeports électroniques.

Le chapitre est structuré de la manière suivante. La section 2.2 décrit les tests statistiques et la méthode pour les appliquer dans le contexte de la thèse. Ensuite, la section 2.3 présente l'exploitation des résultats des tests statistiques à l'aide de l'apprentissage automatique. Cette dernière section présente aussi l'analyse multi-dump permettant d'améliorer les résultats. Enfin, la section 2.4 décrit le protocole d'expérience et fournit les résultats sur des dumps réels.

2.2 Analyse statistique

Cette section explique la difficulté de retrouver les données de classe M dans un dump en utilisant une analyse statistique. Cela est notamment lié au fait que ces informations sont noyées dans une masse de données incluant des séquences de données pseudo-aléatoires générées par des mécanismes cryptographiques. Cette section décrit ensuite la méthode proposée pour appliquer les tests statistiques. Dans l'idéal, les tests devraient être appliqués sur chaque champ de données du dump considéré. Malheureusement, ni l'emplacement ni la taille de ces champs dans le dump ne sont connus. Une méthode consistant à retourner un score statistique bit par bit plutôt que champ par champ est donc proposée.

2.2.1 Identification de clés cryptographiques

Il existe plusieurs techniques permettant de localiser des clés cryptographiques dans un dump de mémoire. Ces techniques ont pour objectif de récupérer la clé (censée être secrète) utilisée par un logiciel du système ciblé.

Les premières méthodes localisant les clés cryptographiques cachées dans des giga-octets de données ont été proposées par [36]. Certaines sont spécifiques aux clés RSA et à leurs propriétés algébriques. Une autre plus générique est basée sur une mesure de l'entropie. Le dump de mémoire est séparé à l'aide d'une fenêtre glissante de 64 octets sur laquelle l'entropie est mesurée. Ensuite, en fonction de cette mesure les données sont considérées comme une clé cryptographique ou non. Cependant, les auteurs ne fournissent pas de détails sur le choix de la taille de la fenêtre glissante ni sur la valeur du seuil à laquelle comparer la mesure de l'entropie pour décider si le bloc fait partie d'une clé cryptographique. De plus, d'autres mesures statistiques que l'entropie peuvent être considérées pour améliorer les résultats. La deuxième méthode localisant des clés cryptographiques dans la mémoire volatile d'un ordinateur proposée par [37] requiert une analyse spécifique de chaque application ciblée. Cette

méthode analyse la structure qui est utilisée par le programme pour stocker la clé cryptographique en mémoire. La troisième méthode proposée par [38] localise les clés AES dans la mémoire volatile d'un ordinateur. Pour cela, elle exploite la dérivation des clés lors du déroulement de l'AES. Parcourant tous les octets du dump de mémoire, la méthode teste si les octets suivants peuvent correspondre à ceux des clés de dérivation en calculant la distance de Hamming entre les différents mots de la clé dérivée. Enfin, les auteurs dans [39] proposent un outil appelé *Interrogate* permettant de localiser les clés cryptographiques dans la mémoire volatile d'un ordinateur. Cet outil implémente les méthodes décrites précédemment et combine même certaines de ces méthodes afin de retrouver aussi des clés utilisées par *SERPENT* ou *Twofish*.

Aucune trace liée à l'exécution ne se retrouve dans la mémoire non volatile des cartes à puce. Pour ces raisons, les techniques [37] et [38] ne peuvent pas être utilisées dans ce contexte. Les objets cryptographiques contenus dans les dumps de mémoire ne sont pas seulement des clés mais peuvent aussi être des hachés, des chiffrés, des signatures, etc. De plus les clés ne sont pas nécessairement stockées de manière contiguë dans la mémoire, c'est le cas dans les cartes EMV. Les techniques proposées dans [36] utilisent des attaques algébriques qui ne fonctionnent que pour les clés RSA. Elles ne répondent donc qu'à une sous partie du problème et ne peuvent fonctionner quand les bits de la clé sont séparés. Quant à la technique utilisant la mesure de l'entropie, rien ne garantit son optimalité concernant la taille de bloc et la mesure utilisée. De plus, la valeur du seuil permettant de prendre la décision n'est pas donnée.

2.2.2 Tests statistiques pour générateurs (pseudo-) aléatoires

Il existe de nombreux tests statistiques pour évaluer la qualité des générateurs aléatoires ou pseudo-aléatoires. Cette évaluation permet de déterminer si ces générateurs sont adaptés à une utilisation cryptographique, on dit alors qu'ils sont cryptographiquement sûrs. La suite de tests statistiques du NIST [34] inclut les tests les plus importants, tout en gardant une redondance faible entre eux ce qui fait d'elle la suite de tests la plus pertinente à l'heure actuelle. Il a donc été décidé de considérer cette suite pour les expériences.

Pour évaluer la qualité d'un générateur, plusieurs tests statistiques doivent être appliqués, chacun analysant une propriété statistique différente. En effet, analyser un générateur avec seulement le test *monobit* ne garantit pas sa robustesse pour un usage cryptographique. Le fait qu'une séquence contienne une proportion équivalente de bits prenant les valeurs 0 et 1 ne garantit pas que ceux-ci soient répartis correctement dans la séquence. Par exemple, la séquence 00000001111111 contient autant de 0 que de 1 et passera donc le test *monobit* avec succès. Par contre, elle ne passera pas le

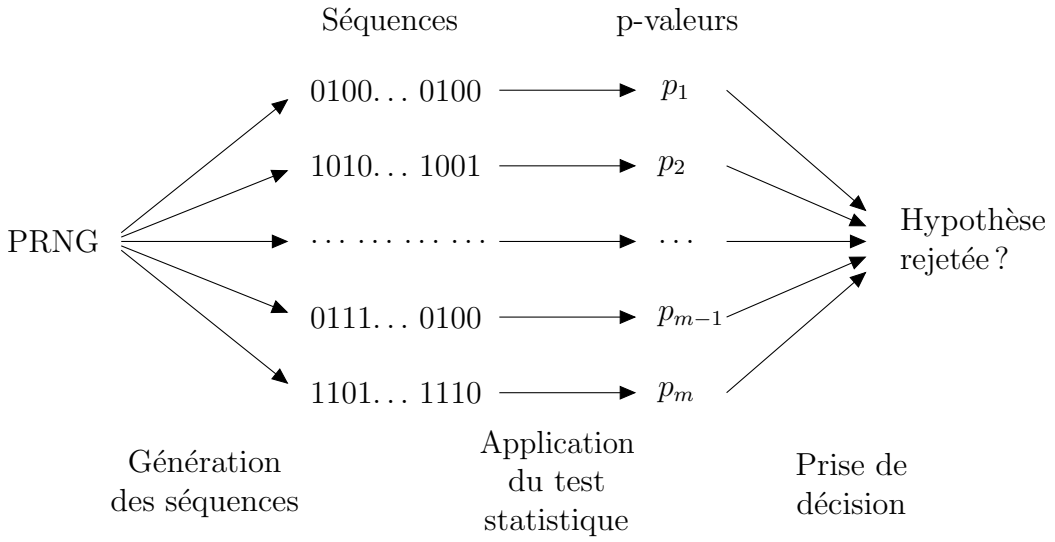


FIGURE 2.1 – Schéma général de l'évaluation d'un générateur pseudo-aléatoire

test du *longest run* qui s'intéresse à la plus longue séquence de bits qui se répètent. Un générateur est donc cryptographiquement sûr si il passe tous les tests avec succès.

Un test statistique a pour but de vérifier une *hypothèse nulle*. Dans le contexte de la thèse, cette hypothèse est *les données sont aléatoires*. Une *p-valeur* représente la force de la preuve contre l'hypothèse nulle. Cette p-valeur est calculée à partir d'une distribution de référence de la propriété statistique testée. L'hypothèse est rejetée si la p-valeur est plus basse que le degré de signification α du test. Pour un usage cryptographique, le NIST recommande d'utiliser des valeurs comme $\alpha = 0,01$ ou $\alpha = 0,001$. Un seuil de $0,01$ signifie qu'il est envisagé qu'une séquence aléatoire sur 100 soit rejetée. Une p-valeur plus élevée que ce seuil (respectivement plus basse) indique que la séquence est considérée comme aléatoire (respectivement non aléatoire) avec une confiance de 99%.

Afin de tester si un générateur (pseudo-)aléatoire est cryptographiquement sûr, un ensemble de m séquences est produit par le générateur à évaluer. Chaque test considéré est appliqué sur chacune des séquences produites. Ainsi, une p-valeur est retournée pour chaque séquence. Le NIST propose ensuite deux méthodes pour prendre la décision sur la qualité du générateur (pseudo)-aléatoire. La première compare chaque p-valeur à un seuil, la deuxième analyse l'uniformité de ces p-valeurs. La figure 2.1 illustre ce schéma d'évaluation.

Plus précisément, pour la première méthode, la décision est prise à partir du taux de séquences ayant passé le test avec succès (i.e., pour lesquelles l'hypothèse n'est pas rejetée). Ce taux est comparé à un seuil calculé en utilisant m et α. Le NIST recommande d'utiliser un nombre de séquences m qui est inversement proportionnel

au degré de signification du test α. Par exemple, pour $\alpha = 0,001$, il est recommandé d'utiliser un ensemble de taille m comprenant au moins 1 000 séquences. Idéalement, de nombreux ensembles indépendants de m séquences doivent être testés, mais le NIST ne fournit pas de valeurs pour ce nombre. La deuxième méthode proposée par le NIST pour approfondir l'analyse des p-valeurs analyse l'uniformité des p-valeurs via un test du χ^2. Pour cette seconde méthode, le NIST conseille d'utiliser un ensemble de séquences de taille m comprenant plus de 55 séquences.

Certains tests tels que le *test monobit*, le *longest runs*, ou le *approximate entropy* peuvent être théoriquement appliqués sur des séquences courtes (> 100 bits). D'autres tests comme le *linear complexity test* ou le *random walk test* nécessitent au moins 10^6 bits de données pour que la p-valeur retournée soit significative.

Le NIST indique qu'il est difficile de déterminer la longueur idéale sur laquelle les tests doivent être appliqués. Néanmoins, il mentionne une longueur utilisée par le document FIPS 140-3, qui est de 20 000 bits. Cependant, cette longueur n'est pas entièrement satisfaisante car certains tests comme le test universel de Maurer nécessite plus de 20 000 bits pour être appliqué. Cela s'explique par le fait que le NIST utilise la distribution asymptotique comme distribution de référence pour les tests. Cette méthode n'est donc pas adaptée pour les séquences courtes. Des détails sur les différentes propriétés statistiques testées ainsi que sur l'interprétation des résultats des tests peuvent être trouvés dans [34].

Pour les séquences courtes, le NIST suggère que les distributions asymptotiques ne sont pas appropriées et devraient être remplacées par les distributions exactes. Toujours selon le NIST, elles sont communément difficiles à calculer. Ainsi, plusieurs contributions [40], [41], et [42] ont introduit des nouveaux tests applicables sur des séquences courtes en donnant leur distribution exacte. Sulak et al. [43] ont aussi suggéré une nouvelle méthode pour évaluer un générateur aléatoire produisant des séquences courtes. Les auteurs ne recommandent pas de valeur particulières pour le nombre de séquences nécessaires ni la longueur des séquences. Cependant, ils illustrent leur méthode en utilisant 10^6 séquences de longueurs variant entre 128, 160 et 256 bits. Malheureusement, bien que ces approches permettent de travailler avec des séquences courtes, elles nécessitent toujours un ensemble contenant un nombre relativement grand de séquences courtes pour évaluer la qualité du générateur.

2.2.3 Les tests statistiques dans le contexte des mémoires de cartes à puce

Les dumps de mémoire de cartes à puce contiennent généralement entre 100 et 40 000 bits de données en fonction de l'application. Un dump contient plusieurs champs

d'informations et chaque champ contient une information (e.g., nom, date, donnée chiffrée, donnée hachée, etc.). La longueur des champs contenant des informations varie généralement entre 1 et 1 024 bits. Ce sont donc des séquences courtes. Il est aussi à noter que l'objectif n'est pas d'évaluer la qualité de l'aléatoire du champ mais de distinguer les données des classes M et C, ce qui impacte le choix du seuil de confiance et de la méthode d'évaluation. En effet, contrairement à l'évaluation d'un PRNG, il n'est pas nécessaire qu'une séquence passe tous les tests avec succès pour décider de sa classe C ou M. De plus, en appliquant les tests statistiques sur des séquences courtes, les p-valeurs retournées par les tests statistiques ne sont peut-être pas toutes autant pertinentes. Il est donc intéressant de déterminer les tests statistiques les plus pertinents. Il est aussi intéressant de combiner les résultats de ces tests afin de décider de la classe de la séquence. Cependant, il n'existe pas de méthode permettant de combiner les p-valeurs de différents tests statistiques dans le cas de l'évaluation de la qualité de l'aléatoire.

Il existe ainsi plusieurs différences avec le contexte des tests statistiques du NIST : (i) les séquences analysées sont bien plus courtes que la longueur mentionnée par le NIST (20 000 bits), (ii) une seule séquence est disponible alors que les techniques proposées par le NIST pour interpréter les p-valeurs retournées nécessitent plusieurs séquences, (iii) le seuil de confiance donné par le NIST doit être re-évalué car l'objectif n'est pas le même, (iv) rien ne garantit qu'il soit judicieux ou nécessaire qu'une séquence passe tous les tests statistiques avec succès pour décider son appartenance à la classe C ou M.

La méthode distinguant les données des classes C et M doit donc remplir les objectifs suivants, (i) établir un moyen de combiner les tests statistiques, (ii) identifier la meilleure combinaison de tests statistiques, (iii) définir le seuil de confiance auquel doivent être comparées les p-valeurs retournées.

2.2.4 Application des tests statistiques sur un dump

Étant donné que les dumps ne peuvent pas être correctement séparés en champs, la classification des données doit être faite bit par bit. Cependant, les tests statistiques ne sont pas applicables sur un seul bit. Une méthodologie qui applique les tests sur des séquences qui se chevauchent est donc proposée. Les p-valeurs obtenues sont ensuite assignées à chaque bit auquel elles sont reliées. Enfin, en utilisant ces p-valeurs, un score est retourné pour chaque bit. La figure 2.2 illustre cette méthode.

Soient D un dump de longueur n bits (les bits sont indicés de 1 à n), ℓ la *longueur* d'une séquence et s le *décalage*, avec $0 < s \leq \ell \leq n$. Le *décalage* représente la distance entre les bits de poids forts de deux séquences testées consécutives. La

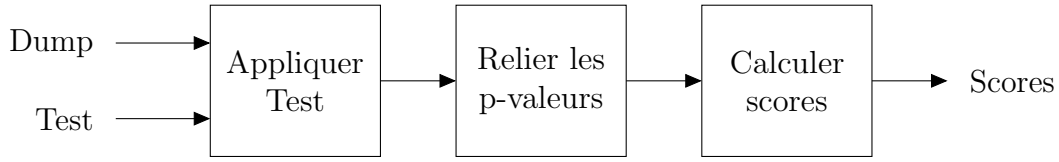


FIGURE 2.2 – Schéma général de l'application d'une caractéristique sur un dump

$i+1$-ème séquence de D est celle allant du bit ayant pour indice $(i \times s + 1)$ au bit d'indice $(i \times s + \ell)$ avec $0 \leq i \leq \lfloor \frac{n-\ell}{s} \rfloor$. Dans le cas où s n'est pas un diviseur de $n - \ell$, les bits allant de l'indice $\lceil \frac{n-\ell}{s} \rceil \times s + \ell - s + 1$ à n ne sont jamais testés, et donc, une dernière séquence allant de l'indice $n - \ell + 1$ à n est testée. Pour chaque séquence testée, le *test statistique* retourne une p-valeur. La figure 2.3 représente l'application d'un test statistique sur un dump.

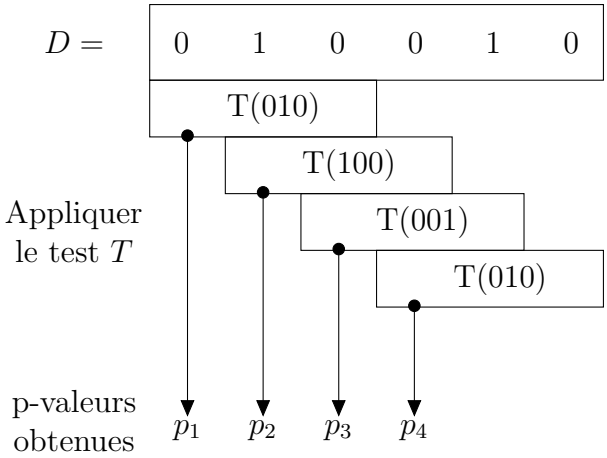


FIGURE 2.3 – Application d'un test statistique T utilisant des séquences de longueur $\ell = 3$ bits avec un décalage $s = 1$ bit, sur un dump D de longueur $n = 6$ bits

En raison de l'usage d'un *décalage* s inférieur ou égal à la *longueur* de la séquence ℓ, certains bits de D sont testés dans plusieurs séquences différentes. Ce nombre de séquences varie entre 1 et $\lceil \frac{\ell}{s} \rceil$ en fonction de la position du bit dans le dump. Chaque séquence testée retourne une p-valeur, tous les bits de D sont donc reliés à un nombre variable de p-valeurs. Cependant, la méthode de classification utilisée dans la section 2.3.4 fonctionne avec un nombre de scores identique pour chaque bit. Une *fonction de score* qui prend des p-valeurs en entrée et génère en sortie un score unique est donc appliquée aux p-valeurs de chaque bit. Cette fonction de score peut par exemple être la moyenne des p-valeurs. La figure 2.4 représente la mise en relation des p-valeurs et le calcul du score pour chaque bit.

Par conséquent, les paramètres précédents : *longueur*, *décalage* et *fonction de score* jouent un rôle important dans la qualité de l'algorithme de classification. De

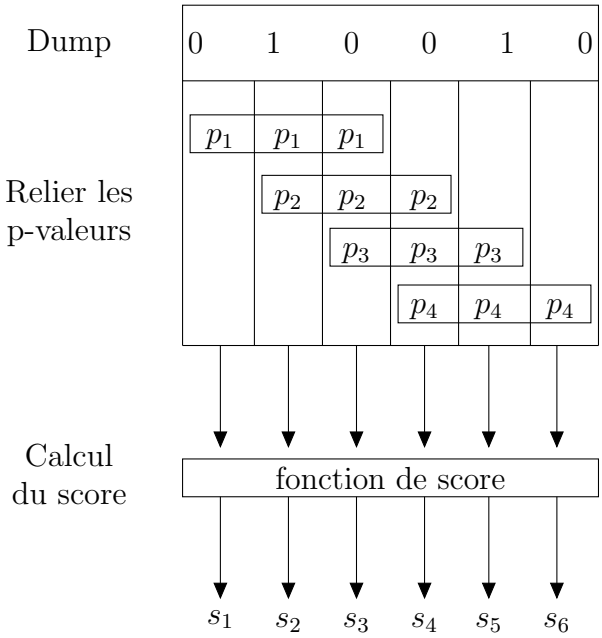
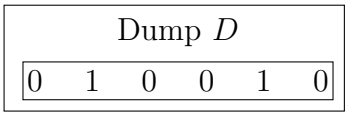


FIGURE 2.4 – Mise en relation des p-valeurs et calcul du score sur un dump D de longueur $n = 6$ bits en appliquant un test statistique T utilisant des séquences de longueur $\ell = 3$ bits avec un décalage $s = 1$ bit

plus, certains tests statistiques nécessitent une *configuration interne*. Par exemple, le *serial test* analysant la proportion de chaque bloc possible de m bits dans la séquence testée, prend m comme configuration interne. Cela conduit à un ensemble $\mathbf{F}$ de X caractéristiques, $\mathbf{F} = \{F_j,\ 1 \leq j \leq X\}$ où chaque caractéristique est définie par un 5-uplet (*test statistique*, *longueur*, *décalage*, *fonction de score*, *configuration interne*). L'application d'une caractéristique F_j sur un dump revient à appliquer le *test statistique* de F_j avec ses paramètres. Cela génère un ensemble de scores $S_j = \{s_i^j,\ 1 \leq i \leq n\}$ où s_i^j est le score assigné par F_j au i-ème bit de D. Appliquer toutes les caractéristiques F_j de $\mathbf{F}$ génère donc un ensemble $\mathbf{S} = \{s_i^j,\ 1 \leq j \leq X,\ 1 \leq i \leq n\}$ comme présenté sur la figure 2.5.

Cette section montre qu'il est possible d'analyser chaque bit d'un dump en utilisant plusieurs tests statistiques paramétrés par un grand nombre de valeurs. Ce chapitre propose donc d'utiliser une méthode d'apprentissage automatique afin de sélectionner les tests statistiques et les paramètres les plus pertinents pour décider si un bit appartient à la classe C ou M.



Appliquer tous les F_j

Indice	1	2	3	4	5	6
Bit	0	1	0	0	1	0
S_1	s_1^1	s_1^2	s_1^3	s_1^4	s_1^5	s_1^6
S_2	s_2^1	s_2^2	s_2^3	s_2^4	s_2^5	s_2^6
.	.	.	.	.	.	.
S_j	s_j^1	s_j^2	s_j^3	s_j^4	s_j^5	s_j^6
.	.	.	.	.	.	.
S_X	s_X^1	s_X^2	s_X^3	s_X^4	s_X^5	s_X^6

FIGURE 2.5 – Ensemble $\mathbf{S} = \{s_i^j,\ 1 \leq j \leq X\}$ des scores obtenus après application de toutes les caractéristiques $F_j \in \mathbf{F}$ sur un dump D d'une longueur de 6 bits

2.3 Apprentissage automatique

À partir des scores obtenus par la méthode d'application des tests statistiques sur un dump, l'objectif est de déterminer pour chaque bit s'il appartient à la classe C ou M. Il est aussi intéressant de connaître les tests statistiques et leurs paramètres les plus utiles pour atteindre cet objectif. Cette section présente donc une technique d'apprentissage automatique d'attributs calculés par les tests statistiques. Elle présente aussi l'analyse multi-dump qui combine les résultats obtenus sur différents dumps d'une même application.

2.3.1 Formalisation

On considère un ensemble d'apprentissage $A = \{(x_i, y_i),\ i \in \{1 \dots n\}\}$ où x_i est un vecteur de caractéristiques décrivant l'objet i et y_i sa classe. L'apprentissage automatique consiste à déterminer de façon automatique une fonction f de prédiction de la classe y à partir d'un vecteur de caractéristiques x connaissant l'ensemble A. La fonction f est appelée classifieur.

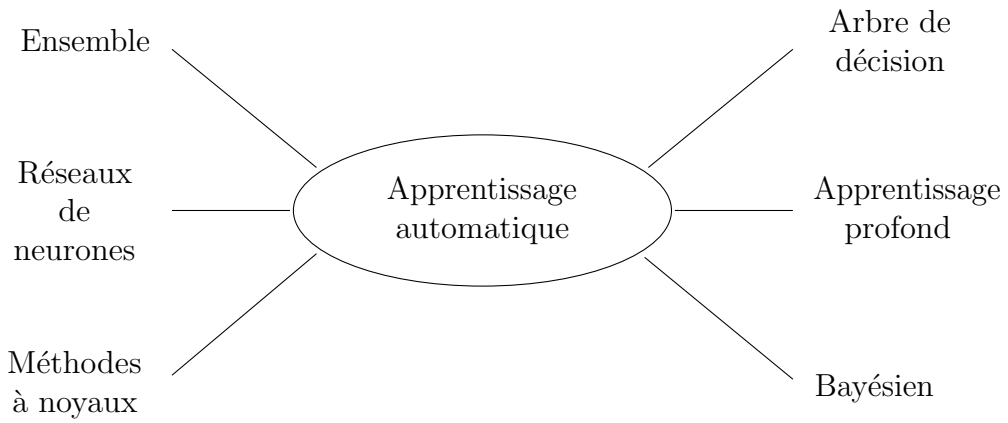


FIGURE 2.6 – Taxonomie de l'apprentissage automatique

Voici un exemple, dans le cas de la classification de courriers malveillants reprenant la technique proposée dans [44]. Les caractéristiques x_i sont les différents mots contenus dans le courrier et les classes sont définies par $y_i \in \{\text{malveillant}, \text{légitime}\}$. On dispose d'un ensemble d'apprentissage $A = \{(x_i, y_i),\ i \in \{1 \dots n\}\}$ composé de n courriers électroniques. Cette base contient à la fois des courriers malveillants et des courriers légitimes. Cette base d'apprentissage permet de remarquer que les courriers malveillants contiennent avec une fréquence plus élevée des mots comme "viagra", "penis", "cash", etc. Il est donc possible d'entraîner un classifieur f prenant en compte la fréquence de ces mots. Ainsi, lorsqu'un nouveau courrier est reçu, les caractéristiques de ce courrier sont calculées. Ensuite, le classifieur f est appliqué pour classer le courrier en tant que *malveillant* ou *légitime*.

2.3.2 Taxonomie de l'apprentissage automatique

La figure 2.6 représente 6 catégories de la taxonomie de l'apprentissage automatique. Il a été choisi de représenter seulement les catégories en lien avec le problème considéré dans ce chapitre. Il existe encore d'autres catégories telles que les algorithmes de réduction de dimension ou de partitionnement de données.

Les réseaux de neurones [45] apprennent des motifs statistiques sur un ensemble d'apprentissage. Ils sont ensuite capables d'effectuer la classification de nouveaux motifs basés sur ces statistiques. Un neurone est composé de différentes entrées sur lesquelles une fonction affine est appliquée pour calculer la sortie. Il reste ensuite à assembler les entrées et les sorties de chaque neurone pour construire le réseau de neurones. C'est lors de l'entraînement que cette architecture est définie. Ces modèles à la base de l'approche par apprentissage profond souffrent d'heuristiques trop importantes pour définir la structure optimale du réseau de neurones. Le perceptron

en est un exemple [45].

L'approche bayésienne est un modèle probabiliste décrivant des liens de cause à effet sur les données. Ce type d'approche basée sur une formalisation mathématique permet également d'évaluer la confiance dans l'affectation dans une classe. Les réseaux bayésiens [46] en sont un exemple.

L'apprentissage par *Arbre de décision* [47] est une méthode basée sur l'utilisation d'un arbre de décision comme modèle de prédiction. Ce type d'approche est plus adaptée à des variables qualitatives ou quantitatives disjonctives (vrai ou faux).

Les méthodes à noyaux ont été introduites pour les problèmes de reconnaissance de formes. La résolution du problème est séparée en deux phases, la première crée une représentation des données dans un espace de dimension élevé. Ensuite, un hyperplan (dans le cas d'une frontière de décision linéaire) séparant les données de manière optimale est recherché. Cette technique évite les minimums locaux qui dégradent les performances des réseaux de neurones. Les machines à vecteur de support (SVM) [48] en sont un exemple.

L'apprentissage profond [49] est une méthode permettant d'obtenir des modèles de calculs composés de plusieurs couches de traitement, ceci afin d'apprendre des représentations sur les données selon plusieurs couches d'abstraction. Cette méthode, très populaire en ce moment, nécessite un large corpus de données annotées (vérité terrain connue). En vision par ordinateur, par exemple, les travaux exploitant ce type d'apprentissage bénéficient de larges bases de données avec plusieurs millions d'images annotées contenant des objets. La différence avec les réseaux de neurones est que l'algorithme d'apprentissage profond sélectionne lui même les caractéristiques les plus discriminantes sur les données. Les réseaux de neurones convolutifs [50] en sont un exemple.

Ensemble [51] représente les méthodes utilisant plusieurs algorithmes d'apprentissage pour obtenir une meilleure prédiction. Ce type d'approche permet de construire des frontières de décision complexes entre les classes à reconnaître. La sélection itérative des classifieurs pertinents permet également de déterminer les caractéristiques importantes pour la reconnaissance. Le boosting [35] est l'une de ces méthodes.

2.3.3 Discussion

L'objectif est de déterminer la fonction de prédiction qui obtient la meilleure reconnaissance entre les données de classes C et M. Il est aussi intéressant d'identifier les tests statistiques qui sont utilisés par cette fonction de prédiction. Cela s'explique par deux raisons, premièrement, il est préférable de ne pas devoir appliquer tout l'ensemble des tests statistiques considérés pour effectuer la classification des bits

d'un dump, deuxièmement, cela permet d'identifier les tests statistiques les plus pertinents à appliquer dans un contexte de séquences courtes.

Les techniques d'apprentissage profond ne sont pas adaptées au contexte car elles nécessitent un ensemble de données trop important pour être appliquées. Les réseaux de neurones sont surtout efficaces sur des données qui ressemblent à celles utilisées lors de l'apprentissage. Or, les dumps de mémoire des cartes à puces sont assez hétérogènes, en particulier à cause des nombreuses applications existantes. Quant aux modèles bayésiens et les méthodes à noyaux, ils ne permettent pas d'identifier les tests statistiques les plus adaptés. Le boosting a été choisi car il répond aux objectifs demandés. En effet, cette méthode approxime la fonction de prédiction voulue tout en retournant les tests statistiques utilisés.

2.3.4 Classification des bits en utilisant des tests statistiques

Pour décider si un bit d'un dump doit être classé comme C ou M, les scores retournés par chaque caractéristique F_j (avec $1 \leq j \leq X$) doivent être comparés à un seuil. Ce processus qui détermine la classe de chaque bit utilisant le score et un seuil prédéfini est appelé un classifieur. À partir d'un dump de longueur n bits, son ensemble de scores S_j obtenu après avoir appliqué la caractéristique F_j et le seuil prédéterminé t, le classifieur C_j calcule la prédiction

$$P_j = \{p_i^j \in \{M,\ C\},\ 1 \leq i \leq n\}$$

pour D. La prédiction de la classe du i-ème bit de D est effectuée de la manière suivante :

$$p_i^j = \begin{cases} C, & \text{si } s_i^j > t \\ M, & \text{sinon} \end{cases}$$

En utilisant les scores retournés par chaque F_j ainsi que la vérité terrain de D, représentée par $G = \{g_i \in \{C, M\},\ 1 \leq i \leq n\}$, un processus d'apprentissage détermine le meilleur seuil t à utiliser. Le processus d'apprentissage décrit dans [52] est utilisé. Ce processus d'apprentissage est un arbre de décision binaire de profondeur 1. Sa complexité est donc en $\mathcal{O}(n)$ dans ce cas présent. Le meilleur seuil est celui qui fournit la classification la plus similaire à la vérité terrain. En d'autres termes, la classification qui maximise le taux de reconnaissance R_j, où R_j est calculé de la manière suivante :

$$R_j = \frac{\sum_i^n r_i^j}{n} \text{ avec } r_i^j = \begin{cases} 1, & \text{si } G_i = p_i^j \\ 0, & \text{sinon} \end{cases}$$

L'application du processus d'apprentissage sur tous les F_j amène à un ensemble de classifieurs $\mathbf{C} = \{C_j,\ 1 \leq j \leq X\}$, où chaque classifieur C_j est relié à une caractéristique F_j.

2.3.5 Boosting de tests statistiques

Le boosting combine plusieurs classifieurs faibles pour créer un classifieur fort. Un classifieur faible est un classifieur qui obtient un taux de reconnaissance légèrement supérieur à 50%. Il doit donc seulement mieux classifier les données qu'un algorithme répondant au hasard le ferait. Un classifieur fort est un classifieur qui obtient un taux de reconnaissance le plus proche possible de 100%. Les scores retournés par l'ensemble de caractéristiques F_j (représentant des tests statistiques paramétrés) peuvent être comparés à un seuil. Ce processus qui compare les scores retournés à un seuil déterminé préalablement se comporte comme un classifieur faible. L'objectif est de combiner ces classifieurs faibles afin de proposer une suite de tests statistiques à appliquer définissant un classifieur fort.

Pour mettre en place cette technique, il est proposé ici d'utiliser l'algorithme AdaBoost qui est la technique de boosting la plus populaire. À partir d'un dump, AdaBoost sélectionne d'abord le meilleur classifieur de l'ensemble $\mathbf{C}$ et l'ajoute au classifieur final $\widehat{C}$. Le meilleur classifieur de l'ensemble $\mathbf{C}$ étant celui qui obtient le meilleur taux de reconnaissance au regard de la vérité terrain du dump. Ensuite, à partir de la classification obtenue par $\widehat{C}$, les bits mal classifiés se voient attribuer un poids plus important. Le boosting sélectionne ensuite le nouveau meilleur classifieur de l'ensemble $\mathbf{C}$ en prenant en compte les poids des bits (ainsi, celui-ci se concentre sur les bits mal classés jusqu'à présent). Il est ajouté au classifieur final $\widehat{C}$. Les poids des bits mal classés sont mis à jour en fonction de la nouvelle classification obtenue par $\widehat{C}$. Cette action est répétée jusqu'à ce que tous les bits du dump soient bien classés, ou alors qu'un nombre prédéfini de classifieurs dans $\widehat{C}$ soit atteint. AdaBoost ne retourne pas le classifieur optimal $\widehat{C}$ car il se comporte comme un algorithme glouton. Néanmoins, il est efficace alors que la recherche naïve du classifieur optimal est calculatoirement impossible à effectuer. En effet, il faudrait comparer le taux de reconnaissance obtenu par chacun des ensembles de k tests statistiques paramétrés parmi un ensemble de X tests statistiques paramétrés avec $1 \leq k \leq X$ et $X \geq 10\,000$. La complexité de AdaBoost est en $\mathcal{O}(|\mathbf{C}|L)$, où L est le complexité de l'algorithme d'apprentissage. Donc, dans ce cas présent, elle est en $\mathcal{O}(|\mathbf{C}|n)$. Le pseudo-code de l'algorithme AdaBoost est fourni dans l'annexe A.

Le classifieur final fourni par le boosting est ensuite utilisé pour distinguer les données des classes C et M dans les dumps où la vérité terrain est inconnue.

2.3.6 Analyse multi-dump

Les dumps appartenant à la même application partagent des similarités sur leurs données. Cependant, la prédiction $\widehat{P}$ obtenue par le classifieur final est faite indépendamment pour chaque dump. Cette analyse multi-dump propose donc de combiner leur classification pour améliorer le taux de reconnaissance.

Comme mentionné dans la section 1.4, les différents champs d'un dump sont localisés au même emplacement dans tous les dumps d'une application. Ainsi, la vérité terrain de tous les dumps d'une application est identique.

À partir d'un ensemble $\mathbf{A} = \{D_k,\ 1 \leq k \leq N\}$ de N dumps de longueur n bits appartenant à la même application, l'analyse multi-dump crée une prédiction P_{multi}. Celle-ci remplace toutes les prédictions du classifieur final. La prédiction P_{multi} est calculée de la manière suivante :

$$P_{multi} = \{Majorité(\widehat{P}_i^1, \widehat{P}_i^2, \dots \widehat{P}_i^N),\ 1 \leq i \leq n\}$$

Avec $\widehat{P}_i^k$ qui représente la prédiction du classifieur final pour le i-ème bit du dump D_k, et *Majorité* qui représente l'application d'un vote majoritaire sur les classes obtenues par les prédictions. La prédiction P_{multi} obtenue atteint un meilleur taux de reconnaissance que ceux des prédictions obtenues indépendamment pour chaque dump de l'ensemble $\mathbf{A}$.

2.4 Expériences : protocole et résultats

Cette section présente les données et les valeurs des caractéristiques utilisées par les expériences de boosting. Le classifieur final obtenu après AdaBoost est appliqué sur des dumps EMV, Vitale, Calypso et des passeports électroniques de la base de dumps présentée dans le chapitre 1. Ce classifieur obtient plus de 94% de taux de reconnaissance sur ces dumps pour les données de classe C et plus de 98% pour les données de classe M. L'analyse multi-dump permet d'atteindre un taux de reconnaissance de 100%.

2.4.1 Génération des données pour la phase d'apprentissage

Pour entraîner les classifieurs, AdaBoost nécessite une quantité suffisante de données appartenant à chaque classe (i.e., classes C et M) de telle sorte qu'elles soient représentatives de toutes les données existantes dans les mémoires des cartes à puces.

Malheureusement, seulement une partie des 371 dumps collectés sont utilisables pour la phase d'apprentissage. En effet, pour la majorité des données, la vérité terrain est inconnue. Pour résoudre ce problème, un dump synthétique contenant des données similaires aux dumps réels a été créé, inspiré des 371 dumps. Ce dump synthétique est modélisé par plusieurs séquences de tailles variables, contenant des données de classe C (données chiffrées ou hachées, clés cryptographiques, etc.) ou des données de classe M (noms, dates, etc.).

Un dump synthétique de 100 000 bits de longueur est créé. Il contient approximativement 65% de séquences représentant des informations de classe M et 35% de séquences de classe C. Les longueurs de séquences de chaque classe sont choisies de manière uniforme entre 80 et 300 bits. Les objets cryptographiques sont générés par divers algorithmes cryptographiques (e.g., RSA, AES, SHA-3) Les sorties de ces algorithmes sont tronquées pour obtenir des séquences de la longueur choisie préalablement. Les informations de classe M incluent des dates avec différents encodages (e.g., ASCII, BCD) représentant différents formats (e.g., YYYY-MM-DDD, DD/MM/YY), ainsi que des compteurs représentant une quantité de temps écoulée depuis une certaine date (e.g., le nombre de secondes depuis le 1er janvier 1970). Ces informations de classe M incluent aussi des informations textuelles (e.g., nom, adresse) ou numériques(e.g., codes postaux, montant de transaction) encodées avec différents encodages.

2.4.2 Caractéristiques considérées

Chaque caractéristique est décrite par 5 paramètres (*test statistique*, *longueur*, *décalage*, *fonction de score*, *configuration interne*) et chaque paramètre peut se voir assigner différentes valeurs. Ainsi, l'ensemble **F** contient approximativement 10 000 caractéristiques différentes.

Test statistique : C'est le test statistique qui est appliqué sur les séquences du dump. Ces tests incluent les tests du NIST, exceptés ceux qui nécessitent plus de 10^6 bits pour être appliqués, ou un pattern spécifique à tester. Cela représente donc 8 tests : *monobit*, *runs*, *block frequency*, *serial*, *discrete fourier transform*, *approximate entropy*, *cumulative sums* et *longest runs*. Ils sont complétés par le test d'*auto-correlation* [53] et les tests adaptés pour les séquences courtes : *TBT* [42] et *saturation point* [40]. Cela représente au total 11 tests.

Longueur : Il a été décidé de borner inférieurement les séquences à 32 bits, car appliquer des tests sur des séquences trop courtes n'est pas pertinent (comme décrit

Tableau 2.1 – Valeurs de configuration interne des tests statistiques

Test	Valeurs
TBT	2, 3, 4, 5, 6, 7
Serial	2, 3, 4, 5, 6, 7, 8
Frequency	2, 3, 4, 5, 6, 7, 8
Approximate entropy	2, 3, 4, 5, 6, 7, 8
Cumulative sum	0, 1
Saturation point	2, 3, 4, 5
Auto-correlation	1 à n/2, avec n la longueur de la séquence testée
Longest runs	8, 10, 14, 16, 24

plus loin). Les différentes longueurs de séquences choisies sont ainsi sélectionnées dans l'ensemble $\{32,\ 48,\ 76,\ 100,\ 128,\ 192,\ 256\}$.

Décalage : 10 décalages différents ont été utilisés, chacun représenté par un pourcentage de la longueur de la séquence testée : $10\%, 20\% \ldots 100\%$. Ce choix est arbitraire mais une étude d'impact des paramètres a montré que cela n'impactait que très légèrement l'efficacité du classifieur obtenu par le boosting.

Fonction de score : Elle représente la méthode qui calcule le score à partir de l'ensemble des p-valeurs de chaque bit. Les différentes fonctions considérées sont : la moyenne arithmétique, le minimum, le maximum et la moyenne géométrique.

Configuration interne : Quand le test nécessite une configuration interne, plusieurs valeurs sont considérées pour cette configuration. Par exemple, pour le test *serial*, la configuration est la taille de bloc dont on teste les fréquences. Le tableau 2.1 présente les valeurs des différentes valeurs de configuration utilisées.

2.4.3 Apprentissage avec AdaBoost

L'algorithme de boosting doit être configuré avec le nombre de classifieurs présents dans le classifieur final. Ce nombre doit être bien choisi en rapport avec le contexte pour empêcher le sur-apprentissage du classifieur final. Ce phénomène apparaît lorsque le classifieur final est trop adapté aux données utilisées pour l'apprentissage. Ce dernier donne un mauvais taux de reconnaissance sur un jeu de données différent de celui utilisé par l'apprentissage. Les expériences ont été effectuées avec un nombre de classifieurs allant de 1 à 50.

Deux dumps synthétiques sont générés, dont un qui représente l'ensemble d'apprentissage et le second celui de validation. Le boosting crée un classifieur final $\widehat{C}$ à

Tableau 2.2 – Tests statistiques et leurs paramètres utilisés par le classifieur final

Test	Longueur	Décalage	Fonction de score	Configuration interne
TBT	128	25	Moy.	4
Longest runs	192	38	Min	24
Auto-correlation	192	57	Min	39
Auto-correlation	192	153	Moy. Géom.	39
Serial	192	192	Moy.	6
Longest runs	256	25	Max	16
TBT	256	76	Moy. Géom.	5
Block frequency	256	128	Min	3
Block frequency	256	128	Max	8
Approximate entropy	256	128	Moy. Géom.	2
Block frequency	256	153	Max	8
Approximate entropy	256	204	Max	4
Block frequency	256	256	Min	2
Block frequency	256	256	Moy.	4

partir du dump d'apprentissage, et, ensuite, il est appliqué sur le dump de validation. Pour déterminer le nombre optimal de classifieurs à utiliser, l'expérience fait varier le nombre de classifieurs dans $\widehat{C}$ sur le dump d'apprentissage. Chaque $\widehat{C}$ obtenu est ensuite appliqué sur le dump de validation. Le classifieur final $\widehat{C}$ qui amène au meilleur taux de reconnaissance sur le dump de validation est sauvegardé.

La figure 2.7 présente le taux de reconnaissance du classifieur final sur les dumps d'apprentissage et de validation en fonction du nombre de classifieurs utilisés. On remarque que le meilleur classifieur sur le dump de validation est celui qui combine les résultats de 14 tests statistiques. Le taux de reconnaissance de ce classifieur final est de $94,14\%$ sur le dump d'apprentissage et de $91,30\%$ sur celui de validation. En utilisant un seul test statistique dans le classifieur final, le meilleur classifieur donne un taux de reconnaissance de $90,7\%$ sur le dump d'apprentissage et un taux de reconnaissance de $88,86\%$ sur celui de validation. En utilisant de plus en plus de tests statistiques dans le classifieur final, on améliore le taux de reconnaissance sur le dump d'apprentissage, par exemple on obtient $98,6\%$ avec 50 tests statistiques. D'un autre côté, le taux de reconnaissance sur le dump de validation ne semble pas s'améliorer pour autant, puisqu'on obtient un taux de $90,88\%$ avec 50 tests. On assiste donc à un phénomène de sur-apprentissage.

Les 14 tests statistiques (et leurs paramètres) de ce classifieur final sont décrits dans le tableau 2.2. Parmi les 14 tests sélectionnés, il n'y en a que 7 différents, ce qui signifie que ces 7 tests sont plus adaptés que les autres utilisés durant l'apprentis-

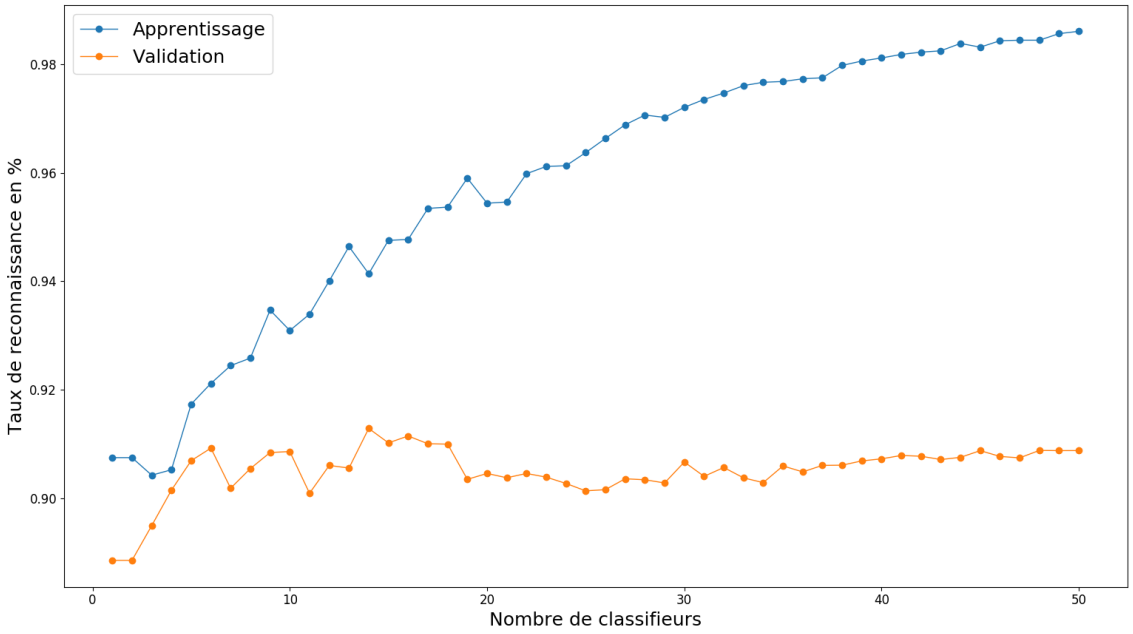


FIGURE 2.7 – Évolution du taux de reconnaissance du classifieur final en fonction du nombre de classifieurs utilisés

sage pour les séquences courtes. On peut aussi remarquer que les tests statistiques sélectionnés par le boosting utilisent des séquences de longueurs 128, 192 et 256 bits, qui sont des tailles assez grandes pour le contexte de ce chapitre. Cela confirme l'intuition qu'appliquer des tests statistiques sur des trop courtes séquences n'est pas efficace. Cependant, le classifieur final est tout de même capable de détecter la classe de certaines séquences ayant une longueur inférieure à 128 bits. Cela peut être expliqué par le fait que chaque bit du dump est testé plusieurs fois dans différentes séquences, lié à la valeur du *décalage*.

L'algorithme de boosting sélectionne les tests statistiques les plus pertinents dans le contexte des séquences courtes appartenant à des dumps de cartes à puce. Il faut noter que, en variant très légèrement le dump d'apprentissage, l'algorithme de boosting retourne un classifieur final différent. Néanmoins, ce classifieur comporte sensiblement les mêmes tests statistiques avec des paramètres similaires. De plus, il fournit des taux de reconnaissances similaires sur le dump de validation.

Les expériences ont été faites avec un programme python en utilisant l'algorithme AdaBoost-SAMME.R [54] de la librairie Scikit-learn [55]. Le calcul des scores représentés sur la figure 2.5 sur un dump de 100 000 bits pour l'ensemble des 10 000 caractéristiques prend plusieurs heures. Les calculs ont été effectués à l'aide de 64 cœurs (4 processeurs AMD Opteron 6282SE cadencés à $2,6$ GHz, chacun possédant 16 cœurs) avec 512 GB de RAM disponibles. L'exécution de l'algorithme d'apprentis-

sage AdaBoost prend, quant à elle, entre quelques minutes (quand $|\widehat{C}| = 1$) et une heure (quand $|\widehat{C}| = 50$). Cet apprentissage est à effectuer qu'une seule fois, les temps obtenus sont donc raisonnables.

2.4.4 Reconnaissance sur des dumps réels

Dans cette section, le classifieur entraîné sur les données synthétiques est utilisé pour classifier les données de classes C et M sur des dumps de mémoire provenant de cartes à puce de la vie réelle. Ce classifieur final est appliqué sur 27 dumps de cartes EMV, 6 dumps de passeports électroniques, 4 dumps de cartes VITALE 2, 3 dumps de cartes VITALE 1, 88 dumps de cartes Calypso et 5 dumps de cartes Moneo. Dans ces cartes, la vérité terrain est connue. En effet, la signification d'une grande partie des données est publiquement connue (EMV, passeports). Aussi, des précédents travaux ont permis de déterminer leur signification (Calypso, Vitale). Cela représente plus de 1 300 000 bits de données avec plus de 180 000 bits de classe C et plus de 1 200 000 bits de classe M. L'application du classifieur final sur ces dumps est très rapide, c'est-à-dire moins de 2 secondes pour les dumps les plus larges (40 000 bits).

Tableau 2.3 – Détection des bits de classe C et M dans les dumps de différentes applications

Application	Dumps	C	Taux	M	Taux.
EMV	27	135 336	93, 89 %	381 344	96, 40 %
Passeport	6	28 912	94, 52 %	40 076	96, 48 %
Vitale2	4	17 360	94, 12 %	253 824	99, 49 %
Vitale1	3	0	–	87 630	99, 69 %
Calypso	88	0	–	420 456	99, 97 %
Moneo	5	0	–	24 128	100, 0 %
Total	133	181 608	94, 01 %	1 207 458	98, 54 %

Le tableau 2.3 montre que la méthode proposée appliquée sur un seul dump fournit de bons résultats. La colonne *Dumps* indique le nombre de dumps sur lesquels l'analyse a été effectuée. Les deux colonnes *Taux* correspondent aux taux de reconnaissance des classes C et M. Un taux de reconnaissance de $94, 01\%$ est obtenu pour les bits de classe C et $98, 54\%$ pour les bits de classe M. On peut remarquer qu'il n'existe pas de variation significative des taux de reconnaissance en fonction de l'application analysée. Ces résultats sont meilleurs que ceux obtenus sur le dump de validation. Cela s'explique par le fait que le dump de validation a été fait pour être représentatif de toutes les données que l'on peut rencontrer dans les dumps de mémoire de cartes à puce. Les expériences de l'évaluation ont été effectuées avec seulement 6 applications,

il est possible que ces applications ne soient pas représentatives des cas les plus difficiles.

La figure 2.8 représente une partie des bits d'un dump d'une carte EMV. Chaque bit est coloré en fonction de la classe qui lui a été attribuée par le classifieur final et de la vérité terrain. La signification des couleurs est la suivante,

— vert : bit de la classe M bien classé ;
— cyan : bit de la classe C bien classé ;
— rouge : bit de la classe M mal classé ;
— violet : bit de la classe C mal classé.

Ainsi, on peut remarquer que les erreurs engendrées par ce classifieur sont souvent localisées à la frontière entre des données de classes différentes ou alors sur des séquences courtes (≤ 100 bits) au sein d'un ensemble de données de classe C ou M relativement grandes ($\geq 1\ 000$ bits). On imagine donc qu'il est possible (avec un post-traitement) de corriger ces erreurs et ainsi obtenir un taux de reconnaissance de 100%. Ce post-traitement est effectué par l'analyse multi-dump.

Tableau 2.4 – Détection des bits de classe C et M dans les dumps de différentes applications en appliquant un classifieur final composé d'un seul test statistique

Application	Dumps	C	Taux	M	Taux.
EMV	27	135 336	$91,42\ \%$	381 344	$98,00\ \%$
Passeport	6	28 912	$93,75\ \%$	40 076	$91,20\ \%$
Vitale2	4	17 360	$93,46\ \%$	253 824	$99,63\ \%$
Vitale1	3	0	–	87 630	$100,0\ \%$
Calypso	88	0	–	420 456	$100,0\ \%$
Moneo	5	0	–	24 128	$100,0\ \%$
Total	133	181 608	$91,98\ \%$	1 207 458	$99,00\ \%$

Le tableau 2.4 présente les résultats de l'application d'un classifieur final comprenant un seul test statistique. Ce classifieur final correspond donc à l'utilisation du meilleur test statistique paramétré. Ce test statistique est le test *approximate entropy* appliqué sur des séquences de longueur 256 bits, avec un décalage de 128 bits entre chaque séquence et 2 comme configuration interne. Les résultats obtenus sur la classe M sont donc améliorés de $0,5\%$ tandis que pour la classe C, ils sont dégradés de 2%. Les résultats sont donc similaires à l'utilisation du classifieur final composés des meilleurs tests statistiques. L'utilisation du boosting n'apporte donc pas un gain très important pour les exemples considérés. Toutefois, il est possible que sur des cas plus complexes, le classifieur final contenant les 14 tests apporte un gain plus conséquent. Néanmoins, ces 2% d'erreurs en plus représentent environ 3 000 bits mal classés.

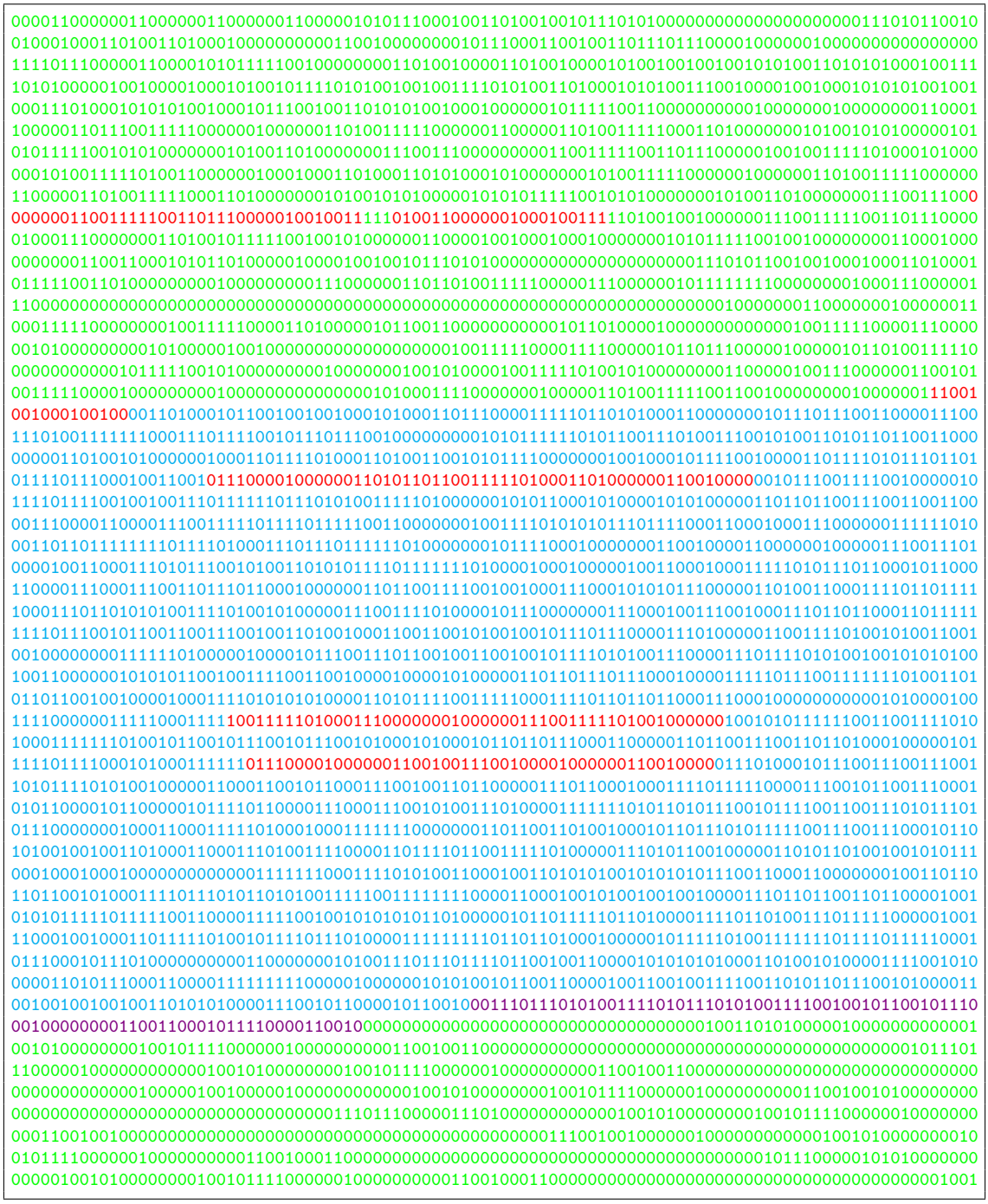


FIGURE 2.8 – Illustration de la reconnaissance d'un dump EMV

Tableau 2.5 – Taux de reconnaissance de l'analyse multi-dump sur différentes applications

Application	Taux	Taux multi-dump
EMV	95, 03%	100, 0%
Vitale 2	98, 90%	100, 0%
Calypso	100, 0%	100, 0%
Moneo	100, 0%	100, 0%

2.4.5 Analyse multi-dump

Ces résultats sont maintenant améliorés en effectuant une analyse multi-dump sur plusieurs dumps provenant de la même application.

Le tableau 2.5 donne les résultats de l'analyse multi-dump pour différentes applications. L'application de l'analyse multi-dump amène un taux de reconnaissance de 100% sur toutes ces applications. La colonne *Taux* correspond à la moyenne des taux de reconnaissances de l'analyse appliquée sur chaque dump séparément. La colonne *Taux multi-dump* correspond au taux de reconnaissance lorsque l'analyse multi-dump est appliquée.

Les taux de reconnaissance de la colonne *Taux* diffèrent légèrement du tableau 2.3 car l'ensemble des données considéré est légèrement différent. En effet, toutes les données de chaque dump ne peuvent pas être fusionnées entre elles. Comme décrit dans le chapitre 1, tous les dumps EMV ne possèdent pas exactement les mêmes champs. Pour les cartes Vitale, certains dumps sont incomplets en raison de restrictions lors de la collecte des données imposées par leur propriétaire respectif.

La figure 2.9 fournit l'évolution du taux de reconnaissance de l'analyse multi-dump en fonction du nombre de dumps pour différents champs de cartes EMV. L'expérience a été réalisée sur les dumps complets sans connaître la vérité terrain. Cette dernière est toutefois utilisée sur la figure 2.9 pour séparer les champs de données et ainsi faciliter la compréhension des résultats.

Soit $nbChamp$ le nombre de champs disponibles pour chaque champ sur lequel l'analyse multi-dump est appliquée. Il aurait fallu tester toutes les combinaisons de k champs possibles parmi $nbChamp$ possible, avec $1 \leq k \leq nbChamp$ pour être rigoureux. Malheureusement, en raison de la combinatoire, cela devient rapidement impossible quand k grandit. Ainsi, seulement une partie des ces sous-ensembles a été choisie pour illustrer l'évolution du taux de reconnaissance.

On remarque que l'analyse multi-dump est très efficace même lorsque l'on utilise peu de dumps. En effet, en utilisant seulement 3 dumps, tous les champs atteignent plus de 98% de taux de reconnaissance. En utilisant 13 dumps, tous les champs ont un taux de reconnaissance de 100%. On peut aussi remarquer qu'il n'existe pas de

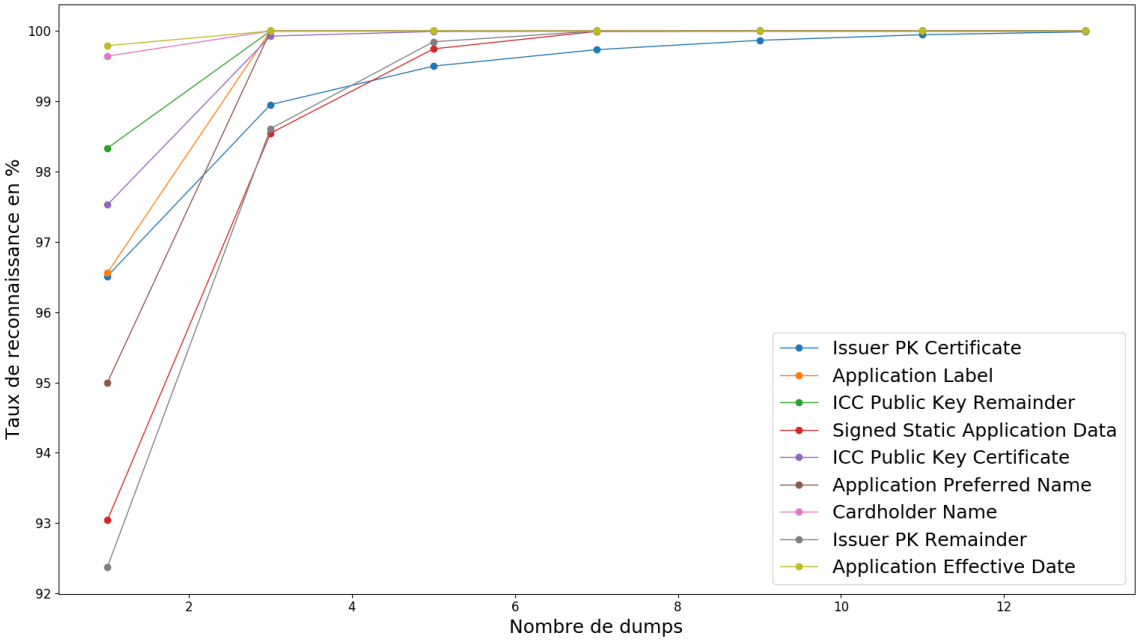


FIGURE 2.9 – Évolution du taux de reconnaissance de l'analyse multi-dump en fonction du nombre de dumps pour certains champs de l'application EMV

disparités entre les résultats des différents champs de l'application.

L'analyse multi-dump obtient ces mêmes résultats de $100,0\%$ lorsqu'elle est appliquée avec le classifieur final composé d'un seul test statistique.

2.5 Conclusion

Ce chapitre montre comment séparer les données représentant des objets *cryptographiques* des données *informationnelles* dans les dumps de mémoire non volatile des cartes à puce. Les objets cryptographiques dans les dumps de mémoire compliquent les techniques d'analyse forensique. En effet, l'application de fonctions de décodage sur des données cryptographiques génère seulement des faux positifs.

Une méthode est alors proposée pour appliquer les tests statistiques sur des dumps de mémoire provenant de cartes à puce. Cette approche utilise une technique d'apprentissage automatique (le boosting) basée sur les résultats de tests statistiques utilisés pour évaluer les générateurs (pseudo)-aléatoires.

Basée sur le même principe de fenêtre glissante que [36], cette méthode optimise les paramètres : taille de séquences, décalage de la fenêtre glissante, mesures utilisées et seuil de décision pour classer les séquences. Cette méthode propose ainsi une nouvelle manière de combiner les tests statistiques et propose aussi une nouvelle technique pour décider la classe de chaque séquence. Elle peut être adaptée à d'autres

contextes. Pour cela, il est nécessaire de disposer de données d'entraînement où la vérité terrain est connue. Ainsi, un classifieur adapté à ce contexte peut être produit par la technique de Boosting.

Il est possible d'utiliser cette nouvelle méthode pour évaluer si un PRNG est cryptographiquement sûr. Cependant, les différences entre les données d'un PRNG biaisé et un PRNG cryptographiquement sûr ne sont peut-être pas suffisamment importantes pour que cette méthode obtienne de bons résultats. Disposer d'une grande quantité de données d'entraînement pourrait éventuellement pallier cette difficulté. Cela permettrait d'appliquer un algorithme de *deep learning* qui serait capable de caractériser de manière précise les propriétés statistiques d'un générateur.

Un taux de reconnaissance aux alentours de 95% est obtenu sur des données réelles provenant de dumps de cartes EMV, Vitale, Calypso, Moneo et des passeports électroniques. Il est aussi préférable d'analyser plusieurs dumps d'une même application simultanément. Cela améliore le taux de reconnaissance jusqu'à 100% pour les différentes applications considérées. Il suffit généralement de combiner seulement 3 dumps de la même application pour obtenir ce résultat de 100%. L'analyse multi-dump est donc très efficace.

L'application de cette méthode sur les dumps dont la vérité terrain est inconnue ainsi que des investigations manuelles ont montré que, en dehors des applications présentées dans ce chapitre (EMV, Vitale, passeport), peu de données cryptographiques étaient finalement présentes dans les autres applications (e.g., tickets de transport, forfait de ski, etc.).

En utilisant un classifieur comprenant un seul test statistique, des résultats semblables sont obtenus. Une diminution du taux de reconnaissance de l'ordre de 3% est observée sur les dumps d'apprentissage et d'entraînement, et une diminution de l'ordre de 2% est observée sur les dumps réels. Le boosting n'apporte donc pas un gain significatif sur les exemples utilisés dans ce chapitre. Toutefois, il n'est pas exclu que le gain soit plus important sur des exemples plus complexes.

Chapitre 3

Recherche d'informations textuelles

Résumé

Ce chapitre introduit une méthode qui retrouve automatiquement les informations textuelles stockées dans les dumps de mémoire des cartes à puce. Partant du fait que la structure et l'encodage des données sont inconnus, la méthode élimine les centaines de faux positifs générés pour chaque dump par le décodage exhaustif. Cette méthode repose sur des statistiques de textes ainsi que sur les spécificités des cartes à puce. Les expériences effectuées sur 371 dumps de cartes à puce de la vie réelle ont automatiquement retrouvé plus de 99% des informations textuelles présentes dans les dumps. En analysant plusieurs dumps de la même application le taux de faux positif est inférieur à $0,6\%$.

Sommaire

3.1 Introduction

Les informations textuelles qui peuvent être retrouvées dans les cartes à puce incluent des noms et prénoms, des adresses postales (pays, ville, rue) ou électroniques, des noms de sociétés, etc.

Ce chapitre introduit la première méthode automatique qui a pour objectif de retrouver les informations textuelles stockées dans des dumps obtenus à partir de

mémoires non volatiles de cartes à puce. La méthode est composée de trois étapes : une analyse lexicale, une analyse syntaxique et une analyse multi-dump.

Cette méthode a été appliquée sans l'analyse multi-dump sur plus de 371 dumps. On retrouve ainsi plus de 99% des informations textuelles stockées dans les dumps et élimine plus de 95% des faux positifs générés par l'analyse lexicale. En appliquant l'analyse multi-dump sur des dumps provenant de 25 applications différentes, le nombre de faux positifs restant est réduit de plus de 85%. Ainsi, $99,4\%$ des faux positifs générés sont éliminés.

Le chapitre est structuré de la manière suivante. Cette section présente l'état de l'art de la recherche d'informations textuelles ainsi que le schéma général de la méthode proposée. La section 3.2 décrit l'analyse lexicale générant les chaînes de caractères à partir des données binaires. La section 3.3 présente ensuite l'analyse syntaxique utilisant un classifieur bayésien pour éliminer les faux positifs. Cette dernière section présente aussi l'analyse multi-dump. Le protocole d'expérience, les métriques utilisées et les résultats d'expériences sont fournis dans la section 3.4.

3.1.1 Travaux connexes

Deux champs de recherche sont connexes au recouvrement de données textuelles, à savoir : la fouille de textes (*text mining*) et la recherche d'informations (*information retrieval*). La fouille de textes [56] est un champ interdisciplinaire combinant la fouille de données et la linguistique informatique. Cela consiste à extraire la connaissance à partir de documents textuels. Par exemple, ces techniques extraient la connaissance en analysant les relations entre les mots et les phrases d'un document. Cette technique n'est pas applicable dans le contexte de la thèse. En effet, les chaînes de caractères traitées sont indépendantes.

Le second champ de recherche précédemment mentionné est la recherche d'informations [57], celle-ci a pour but de retrouver des documents connexes à une information donnée à partir d'une collection de documents textuels. Pour chaque document, un score relié à l'information recherchée est calculé. Ensuite, les documents avec les meilleurs scores sont retournés. La technique fonctionne avec une collection de documents textuels, alors que l'encodage des données provenant de dumps de mémoire de cartes à puce est inconnu. De plus, la nature des informations recherchées est elle aussi inconnue.

Déjà mentionné dans le chapitre 1, il existe une technique proposée par [19] retrouvant (entre autres) des informations textuelles dans des données de masse. Cette technique classe aussi les informations dans diverses catégories : URL, adresses électroniques, mots, etc. Les auteurs fournissent un outil implémentant leur technique :

bulk extractor [20]. Cependant, cet outil ne décode pas le dump avec toutes les fonctions de décodage existantes dans le contexte de la thèse, et n'essaie pas non plus tous les décalages possibles. De plus, cet outil élimine aucun des faux positifs générés. Toutes ces raisons font que cet outil n'est pas suffisant dans le contexte de la thèse.

3.1.2 Schéma général de la méthode

La méthode introduite dans ce chapitre a pour but de retrouver les informations textuelles stockées dans les dumps de mémoire obtenus à partir de cartes à puce. Cette méthode est séparée en trois analyses : lexicale, syntaxique et multi-dump. La figure 3.1 illustre le schéma général de la méthode.

Premièrement, une analyse lexicale génère des chaînes de caractères, en assemblant ceux obtenus après l'application d'une fonction de décodage sur le dump. Si l'on considère que l'on utilise les fonctions de décodage adéquates, l'analyse lexicale génère un ensemble de chaînes de caractères contenant toutes les informations textuelles stockées dans la carte à puce. Malheureusement, cette étape génère aussi un nombre important de faux positifs.

La seconde étape a pour but d'éliminer les faux positifs. Elle consiste en une analyse syntaxique qui vérifie pour chaque chaîne de caractères générée si elle *fait sens* ou non. L'analyse syntaxique utilise un classifieur bayésien étudiant les n-grammes de chaque chaîne de caractères générée afin de décider si elle représente une information ou non. Les n-grammes d'une chaîne de caractères sont toutes les sous-séquences de n caractères contiguës de celle-ci.

La dernière étape est une analyse multi-dump qui utilise plusieurs dumps de la même application dans le but de combiner les résultats obtenus par l'analyse syntaxique sur chaque dump. Si un seul dump est disponible, cette analyse ne peut pas être appliquée et la méthode proposée s'arrête donc à la fin de l'analyse syntaxique.

3.2 Analyse lexicale

L'analyse lexicale est une procédure comportant deux étapes qui décode les données binaires du dump pour générer des caractères, puis les assemble en chaînes de caractères.

3.2.1 Décodage des données binaires

Soit f une fonction de décodage appliquée sur m bits telle que

$$f : \{0,1\}^m \rightarrow \{0,1,\ldots,2^m-1\}.$$

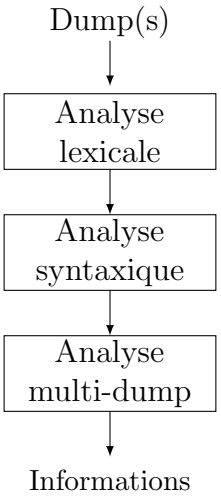


FIGURE 3.1 – Schéma général de la recherche d'informations textuelles dans un dump de mémoire

Par exemple, si la fonction de décodage est l'ASCII, l'entrée est une séquence binaire de longueur 8 bits et la sortie représente le code ASCII du caractère. Toutes les sorties de la fonction ne représentent pas nécessairement un caractère. C'est par exemple le cas avec des valeurs plus grandes que 26 pour l'encodage 5-bit, qui ne représentent pas un caractère. La fonction d'encodage 5-bit est une fonction qui utilise 5 bits pour encoder chaque lettre. Chaque lettre est convertie en un entier représentant l'index de cette dernière dans l'alphabet, ensuite, cet entier est converti sous sa représentation binaire sur 5 bits.

On peut distinguer deux particularités dans ce contexte de recherche d'informations textuelles dans les mémoires de cartes à puce. Les fonctions d'encodage n'utilisent pas forcément un octet entier pour stocker un caractère. De plus, le bit de poids fort de l'information n'est pas forcément aligné sur les octets, ceci implique que la fonction de décodage doit être appliquée sur tous les décalages possibles, i.e., en appliquant f à partir du bit d'indice 0 du dump, puis à partir du bit d'indice 1, etc. Ce décalage s'arrête au bit d'indice $m-1$, car appliquer la fonction f à partir du bit d'indice m correspond à la deuxième sortie de f appliquée à partir du bit d'indice 0.

Soit D un dump de longueur n bits qui est représenté par une séquence binaire $(b_i)_{1\leq i\leq n}$, une fonction f utilisant des séquences de longueur m bits, est appliquée sur D pour générer :

$$\left\{ f\left((b_i)_{\alpha\leq i\leq\beta} \right),\ 0\leq j\leq \left\lfloor \frac{n}{m} \right\rfloor \right\}_s,\quad 0\leq s\leq m-1$$

où $\alpha = jm+s+1$ et $\beta = (j+1)m+s$, et où s est appelé le décalage. On obtient donc s ensembles de sortie.

Dans l'exemple donné ci-dessous, on décode le nom enregistré dans un dump Calypso de la section 1.2.4. La représentation hexadécimale du nom est : `0x282D2CF`. Le dump D testé est donc représenté par la séquence binaire de $n = 28$ bits suivante : $D = 0010100000101101001011001111$. La fonction de décodage est alors appliquée avec $m = 5$ bits et un décalage de $s = 1$ bit : $f(01010)$, $f(00001)$, $f(01101)$, $f(00101)$, $f(10011)$. Elle retourne les entiers : $10, 1, 13, 5, 19$ qui correspondent aux caractères alphabétiques "james".

Il n'a jamais été observé aucun signe diacritique (e.g., accents, cédille et tilde), sur les lettres utilisées pour enregistrer une information dans les dumps collectés. Ainsi, l'alphabet de travail contient les 26 lettres de l'alphabet latin (de "a" à "z", en majuscule ou en minuscule), complétées par les caractères qui peuvent être utilisés pour les URL ou les adresses électroniques (e.g., ".", "@", "_", "/", etc.). Cela implique que l'UTF-8 et les autres jeux de caractères populaires (e.g., ISO-8859-1) donnent le même résultat que la fonction de décodage ASCII. Cela s'explique par le fait que les codes des caractères de l'alphabet de travail sont identiques dans tous ces encodages. Cela réduit fortement l'ensemble des fonctions à appliquer.

3.2.2 Création des chaînes de caractères

Les chaînes de caractères sont construites en assemblant les caractères consécutifs de l'alphabet de travail générés par la fonction de décodage utilisant le même décalage. À partir d'un dump et d'une fonction de décodage, un ensemble de chaînes de caractères est alors construit, où chaque chaîne de caractères est soit une information soit un faux positif.

Cette méthode peut générer des chaînes de caractères contenant une information bruitée. Plus précisément, cela arrive lorsqu'une information stockée dans un dump est proche de données qui, une fois décodées, génèrent des caractères de l'alphabet de travail. Par conséquent, la chaîne de caractères construite contiendra l'information mais aussi du *bruit*. Par construction, les données arbitraires, également appelées *bruit* peuvent seulement être placées avant ou après l'information.

Par exemple, la date de naissance et le nom du possesseur de la carte sont stockés de manière rapprochée dans le dump Calypso fourni par la figure 1.9. La fonction de décodage 5-bit appliquée sur la date de naissance `0x19750710` génère des caractères de l'alphabet de travail. Le nom du possesseur est donc bruité par ces caractères, et la chaîne construite par l'analyse lexicale est "lwjaxprjames" au lieu de "james".

En raison des différents décalages et fonctions de décodage possibles, l'analyse lexicale génère des chaînes de caractères qui sont des informations (possiblement bruitées), mais elle génère aussi un nombre important de chaînes de caractères qui

n'en contiennent pas et sont donc des faux positifs. Comme déjà présenté dans l'introduction, un faux positif apparaît lorsqu'une séquence de bits d'un dump est décodée avec une fonction qui est différente de celle utilisée pour l'encoder. Une analyse supplémentaire est donc nécessaire pour éliminer les faux positifs.

3.3 Analyses syntaxique et multi-dump

L'entrée de l'analyse syntaxique est l'ensemble des chaînes de caractères générées par l'analyse lexicale. L'analyse syntaxique a pour but d'identifier les chaînes de caractères qui ont été décodées avec la bonne fonction et au bon endroit dans le dump. Les chaînes de caractères non éliminées sont ensuite utilisées par l'analyse multi-dump, qui compare les résultats obtenus par l'analyse syntaxique sur différents dumps d'une application.

3.3.1 Information vs non information

Il n'existe pas d'oracle pour décider si une chaîne de caractères a été décodée avec la bonne fonction de décodage. Par conséquent, il est supposé qu'une information textuelle qu'un opérateur a voulu enregistrer dans un dump représente une chaîne de caractères qui fait sens pour un humain. Dans le cas contraire, l'information est soit chiffrée, soit obfusquée, ce qui est en dehors du cadre de cette étude car cela reviendrait à effectuer une cryptanalyse. Sous cette hypothèse de travail, l'objectif de la méthode proposée est d'identifier les chaînes de caractères qui contiennent (au moins) une sous-chaîne faisant sens.

Dans ce qui suit, une chaîne de caractères décodée avec la bonne fonction, de sorte qu'elle contienne une sous-chaîne qui fait sens, est appelée une *information*. Notons qu'une chaîne de caractères peut contenir une sous-chaîne faisant sens alors que la fonction de décodage utilisée durant l'analyse lexicale n'était pas celle employée par l'opérateur pour stocker cette donnée. Dès lors que la fonction de décodage appliquée est différente de celle utilisée par l'opérateur, la chaîne de caractères générée est appelée un *faux positif*. Une chaîne qui n'est pas une information, qui est classifiée par l'analyse syntaxique comme une information est donc un faux positif.

Par exemple, la chaîne de caractères "uuvehicule" contient une sous-chaîne faisant sens. Si la chaîne de caractères a été décodée avec la bonne fonction, alors "uuvehicule" est une information. Si "uuvehicule" avait été obtenue "par chance", alors ce n'est pas une information, bien qu'elle contienne une sous-chaîne faisant sens.

Par la suite, il est montré que l'analyse syntaxique est assez efficace pour identifier les chaînes de caractères qui contiennent une sous-chaîne faisant sens. Cependant,

une chaîne qui n'est pas une information qui contient une sous-chaîne faisant sens, générera certainement un faux positif. L'analyse multi-dump éliminera heureusement la plupart de ces faux positifs.

3.3.2 Analyse syntaxique

Le principe de l'analyse syntaxique est d'évaluer la proximité de chaque chaîne de caractères à analyser avec des mots appartenant à un corpus donné. Cela permet à l'analyse syntaxique de décider si oui ou non la chaîne de caractères (ou une sous-chaîne qu'elle contient) fait sens pour un humain. Il est détaillé ci-dessous le double mode opératoire de cette analyse syntaxique, qui est basée sur une approche par dictionnaire, ainsi qu'une approche plus robuste basée sur un classifieur bayésien.

Une étape préliminaire de l'analyse syntaxique consiste donc à élaborer un large dictionnaire de toutes les informations textuelles potentielles. La première opération consistant à vérifier si les chaînes de caractères analysées appartiennent à ce dictionnaire.

Cependant, il est impossible, en pratique, de créer un dictionnaire exhaustif de noms propres et communs. En particulier, des nouveaux noms sont régulièrement créés, bien qu'ils gardent des règles de construction usuelles. Ce dictionnaire peut tout de même être complété par des informations contextuelles, par exemple, des informations imprimées sur la carte à puce (e.g., nom ou date) ou bien une information à propos du possesseur (e.g., nom, adresse, entreprise). De plus, les informations textuelles peuvent être comprises dans une chaîne de caractères bruitée en raison du fonctionnement de l'analyse lexicale. Il est donc proposé de compléter cette analyse par dictionnaire à l'aide d'un classifieur bayésien pour décider si la chaîne de caractères est une information ou non.

Le classifieur bayésien calcule deux scores : le premier score représente la confiance que l'on a sur l'hypothèse que la chaîne de caractères soit une information, et le deuxième score la confiance sur l'hypothèse que la chaîne de caractères ne soit pas une information. Le classifieur compare ensuite ces deux scores pour décider si la chaîne de caractères est une information ou non.

3.3.3 Classifieur bayésien pour les informations textuelles

Dans le contexte de recouvrement des informations textuelles dans un dump, un classifieur naïf bayésien assigne une classe à chaque chaîne générée par l'analyse lexicale. Les classes sont I (information) et $\bar{\text{I}}$ (non information). Chaque chaîne est caractérisée par un vecteur $\vec{x} = \langle x_1, x_2, x_3 \ldots, x_n \rangle$ qui représente n caractéristiques

indépendantes. Ici, les caractéristiques x_i représentent la présence de chaque n-gramme de la chaîne à classer. Par exemple, si "ab" est la chaîne à classer, le vecteur de caractéristiques qui représente la chaîne est $\vec{x} =$ ("ab", "a", "b").

Un classifieur naïf bayésien est une technique d'apprentissage automatique qui utilise le théorème de Bayes avec des hypothèses d'indépendances fortes entre les différentes caractéristiques. Ce classifieur considère que toutes les caractéristiques de l'instance contribuent indépendamment à la probabilité d'appartenir à une classe. Cette hypothèse est faite sans regarder les possibles corrélations entre les caractéristiques. C'est la raison pour laquelle le classifieur est considéré comme naïf. D'un autre côté, Zhang [58] a montré que même lorsque les caractéristiques ne sont pas indépendantes, sous certaines conditions, les classifieurs naïfs Bayésiens obtiennent des résultats optimaux. Ces conditions sont que les dépendances des différentes caractéristiques s'annulent les unes les autres.

Le classifieur naïf bayésien assigne à chaque chaîne de caractères et pour chaque classe c la probabilité $\Pr(c \mid \vec{x})$ que la chaîne de caractères représentée par $\vec{x}$ appartienne à la classe c. En utilisant le théorème de Bayes, on obtient :

$$\Pr(c \mid \vec{x}) = \frac{\Pr(\vec{x} \mid c)\ \Pr(c)}{\Pr(\vec{x})}.$$

En utilisant l'hypothèse d'indépendance entre les caractéristiques x_i des classes c, les probabilités conditionnelles $\Pr(\vec{x} \mid c)$ peuvent être réécrites comme suit :

$$\Pr(\vec{x} \mid c) = \prod_{i=1}^{n} \Pr(x_i \mid c).$$

Par exemple, la probabilité que la chaîne de caractères "ab" appartienne à la classe I est :

$$\Pr(\mathrm{I} \mid \vec{x}) = \frac{\Pr(\text{“ab”} \mid \mathrm{I})\ \Pr(\text{“b”} \mid \mathrm{I})\ \Pr(\text{“a”} \mid \mathrm{I})\ \Pr(\mathrm{I})}{\Pr(\vec{x})}.$$

Pour chaque chaîne de caractères représentée par $\vec{x}$, la classe c assignée est celle qui a la plus haute probabilité $\Pr(\vec{x} \mid c)$. La valeur $\Pr(\vec{x})$ ne dépend pas de la classe c. Donc, pour déterminer la classe c qui a la plus haute probabilité $\Pr(\vec{x} \mid c)$, le dénominateur $\Pr(\vec{x})$ peut être ignoré durant le calcul.

La probabilité conditionnelle $\Pr(x_i \mid \mathrm{I})$ représente la probabilité qu'une information contienne le n-gramme x_i. Ces dernières sont calculées en utilisant un corpus d'apprentissage, qui est le même dictionnaire que celui qui est utilisé pour la première étape de l'analyse syntaxique. Pour les calculer, on considère que tous les mots du corpus appartiennent à la classe I, les autres mots appartiennent à la classe $\bar{\mathrm{I}}$. Les

deux ensembles de mots des classes I et $\bar{\text{I}}$ ont besoin d'être finis dans le but de calculer les probabilités, donc une longueur maximale $\ell_{\max}$ pour les mots testés est fixée.

La valeur Pr(I) représente la probabilité qu'une chaîne de caractères appartienne à la classe I. Elle dépend de la source des données (EMV, Calypso, etc.) et de la fonction de décodage utilisée pour générer les chaînes de caractères. Cette valeur peut être définie expérimentalement. Lorsque les données nécessaires pour les expériences ne sont pas disponibles, la valeur $\Pr(\text{I}) = 0,5$ peut être utilisée par défaut. Les expériences ont montré qu'en utilisant des valeurs différentes de celle qui est optimale pour Pr(I), le taux de reconnaissance n'est diminué que de $0,5\%$ au maximum.

Les détails des calculs des probabilités précédentes sont fournis ci-dessous. La longueur de la caractéristique x_i (i.e., le nombre de caractères du n-gramme) est notée ℓ_{x_i}, $\mathbf{C}$ représente le nombre de mots dans le corpus d'apprentissage et $\mathbf{C}_{x_i}$ représente le nombre de mots contenant le n-gramme x_i dans le corpus. La valeur suivante $\mathbf{S}$ représente le nombre de chaînes de caractères de longueur 1 à $\ell_{\max}$ caractères.

$$\mathbf{S} = \sum_{i=1}^{\ell_{\max}} 26^i$$

Ainsi, $\overline{\mathbf{C}} = \mathbf{S} - \mathbf{C}$, avec $\overline{\mathbf{C}}$ représentant le nombre de chaînes de caractères qui ne sont pas dans le corpus. La valeur suivante $\mathbf{S}_{x_i}$ représente le nombre de chaînes de caractères contenant le ℓ_{x_i}-gramme x_i.

$$\mathbf{S}_{x_i} = \sum_{\ell=\ell_{x_i}}^{\ell_{\max}} \mathbf{S}_{x_i}^{\ell}$$

Enfin, la valeur suivante $\mathbf{S}_{x_i}^{\ell}$ représente le nombre de chaînes de caractères de ℓ lettres contenant un ℓ_{x_i}-gramme.

$$\mathbf{S}_{x_i}^{\ell} = (\ell - \ell_{x_i} + 1) * 26^{(\ell - \ell_{x_i})}$$

Ainsi, cette probabilité $\Pr(x_i \mid \text{I})$ représente la probabilité d'avoir le ℓ_{x_i}-gramme x_i dans la chaîne de caractères sachant que cette dernière est une information.

$$\Pr(x_i \mid \text{I}) = \frac{\mathbf{C}_{x_i}}{\mathbf{C}},$$

Aussi, on peut calculer $\Pr(x_i \mid \bar{\text{I}})$, la probabilité d'avoir le ℓ_{x_i}-gramme x_i dans la chaîne de caractères sachant que cette dernière n'est pas une information.

$$\Pr(x_i \mid \bar{\text{I}}) = \frac{\overline{\mathbf{C}}_{x_i}}{\overline{\mathbf{C}}} = \frac{\mathbf{S}_{x_i} - \mathbf{C}_{x_i}}{\mathbf{S} \ - \ \mathbf{C}}$$

Certains événements anormaux peuvent survenir et doivent être pris en considération. Ici, un événement anormal est un ℓ_{x_i}-gramme x_i qui appartient à une chaîne

de caractères testée (représentée par $\vec{x}$) mais qui n'apparaît pas dans le corpus d'apprentissage. Le numérateur de $\Pr(\vec{x} \mid \mathrm{I})$ est un produit. Ainsi, si une des composantes est nulle, alors, le résultat final du produit sera nul. Cela a pour conséquence que si une caractéristique de la chaîne de caractères testée n'est pas dans le corpus de la classe I, alors, la probabilité $\Pr(\vec{x} \mid \mathrm{I})$ est nulle. On ne veut pas que ce soit le cas car une information peut se retrouver dans une chaîne de caractères qui est bruitée, et, dont un des n-gramme de ce bruit n'appartient pas au corpus d'apprentissage. Ainsi, lorsqu'un n-gramme n'apparaît pas dans le corpus, il est ignoré dans le calcul du score. C'est donc un cas d'indécision, et cette solution est équivalent à assigner la même probabilité à $\Pr(x_i \mid \mathrm{I})$ et $\Pr(x_i \mid \bar{\mathrm{I}})$ dans le calcul.

Lorsqu'une chaîne de caractères construite contient un caractère qui n'est pas dans l'alphabet latin (e.g., "@"), la chaîne est séparée en plusieurs sous-chaînes contenant seulement des caractères latins. Si au moins une des sous-chaînes est considérée comme une information par le classifieur, alors, la chaîne complète est classée comme une information.

3.3.4 Analyse multi-dump

Cette section présente l'analyse multi-dump qui permet de supprimer tous les faux positifs dans la plupart des cas. Pour les cas les plus difficiles, une partie importante des faux positifs sera tout de même supprimée. Les dumps extraits de cartes provenant de la même application (e.g., Calypso) possèdent des similarités dans la structure de leur mémoire. On propose d'exploiter cette spécificité en combinant les classifications obtenues par l'analyse syntaxique et ainsi améliorer significativement le taux de reconnaissance du classifieur.

Comme expliqué précédemment, les données stockées au même indice dans chaque dump de l'application sont encodées avec la même fonction de décodage et représentent le même type d'information. Donc, les chaînes de caractères générées à cet indice par l'analyse lexicale appartiennent toutes à la même classe I ou $\bar{\mathrm{I}}$.

En pratique, toutes les valeurs d'un champ sur les différents dumps d'une application ne sont pas de la même longueur. Par exemple, le nom de deux personnes différentes a peu de chance d'être de la même longueur. Donc, ils ne vont pas commencer ou finir au même indice dans le dump. Ainsi, toutes les chaînes de caractères qui ont au moins un indice en commun avec une autre chaîne de caractères provenant d'un dump de la même application sont mises en relation.

Les classes obtenues après l'analyse syntaxique sur ces chaînes liées entre elles peuvent être combinées dans le but de prendre une décision commune. Cette décision

commune peut ainsi éliminer les faux positifs générés à un indice donné dans les cas suivants :

— Au moins une des chaînes de caractères générées à cet indice a été classifiée comme n'étant pas une information par l'analyse syntaxique.
— Il existe un dump qui n'a pas généré de chaîne de caractères à cet indice.
— Toutes les chaînes de caractères générées à cet indice sont identiques.

Pour le premier cas, il a été choisi qu'une seule chaîne de caractères classifiée comme n'étant pas une information par l'analyse syntaxique était une condition suffisante pour que toutes les chaînes de caractères générées à cet emplacement le soient aussi. Un autre choix possible aurait été un vote majoritaire. Mais, comme il sera montré par la suite, l'analyse syntaxique retrouve plus de 99% des informations alors qu'elle est légèrement moins efficace pour éliminer les faux positifs. Une élimination stricte des faux positifs a donc été privilégiée. Pour le deuxième cas, ne pas générer de chaînes de caractères à un certain emplacement sur tous les dumps signifie que les chaînes de caractères obtenues sur certains dumps étaient dues au hasard. Le dernier cas n'est valable que lorsqu'on s'intéresse à des informations liées au possesseur de la carte. En effet, des chaînes de caractères liées à l'application seront identiques pour tous les dumps. Ainsi, l'analyse multi-dump peut être appliquée à deux niveaux. Une première fois avec ces trois cas d'élimination. Elle retrouvera ainsi les informations personnelles des possesseurs. Une seconde fois, avec seulement les deux premiers cas d'élimination. Elle retrouvera toutes les chaînes de caractères mais pourra aussi générer des faux positifs supplémentaires.

Soit $\mathbf{D} = \{D_i,\ 1 \leq i \leq N\}$ un ensemble de N dumps provenant de la même application. À partir d'une fonction de décodage et d'un décalage pour appliquer la fonction, l'application de l'analyse lexicale sur chaque dump D_i génère un ensemble de t chaînes de caractères $\{w_k^i,\ 1 \leq k \leq t\}$. Soit Classe$(w_k^i)$ la classe obtenue par l'analyse syntaxique (I ou $\overline{\text{I}}$) de la chaîne de caractères w_k^i.

Soient les t ensembles de chaînes de caractères suivants :

$$\widehat{W_k} = \{w_k^i,\ 1 \leq i \leq N\}_{1 \leq k \leq t}.$$

Chaque ensemble $\widehat{W_k}$ représente un ensemble de N séquences liées. Pour un indice k donné, toutes les séquences de $\widehat{W_k}$ ont été générées à partir du même indice dans le dump. Ou, tout du moins, ils ont en commun au moins un indice d'un de leurs bits. Dans le cas où les données génèrent des chaînes de caractères pour seulement une partie des dumps considérés à un indice donné, les séquences w_k^i peuvent être de longueur 0.

L'analyse multi-dump va donc assigner une classe à chaque ensemble $\widehat{W_k}$. Cette classe sera I si les trois assertions `Assert1, Assert2, Assert3` sont vérifiées, $\overline{I}$

sinon. Chacune des assertions correspond à un des cas d'élimination des faux positifs décrits précédemment.

$$\texttt{Assert1.}\ \left|\{\text{Classe}(w_k^i) = \bar{\text{I}}\}_{1\leq i\leq N}\right| < 1$$
$$\texttt{Assert2.}\ \left|\{\left|(w_k^i)\right| = 0\}_{1\leq i\leq N}\right| == 0$$
$$\texttt{Assert3.}\ \left|\text{Unique}(\widehat{W_k})\right| > 1$$

où Unique($\widehat{W_k}$) représente l'ensemble des valeurs différentes de l'ensemble $\widehat{W_k}$.

3.4 Validation expérimentale

3.4.1 Protocole

Dans le but de valider l'efficacité de la méthode proposée, elle est expérimentée sur des dumps réels. Elle est appliquée sur un ensemble de chaînes de caractères générées par l'analyse lexicale.

Les fonctions de décodage qui sont appliquées sur les dumps provenant de cartes à puce de la vie réelle sont la fonction de décodage ASCII sur 7 bits (ASCII-7), ASCII sur 8 bits (ASCII-8), le décodage 5-bit (5-bit), le décodage 5-bit utilisant 8 bits pour stocker l'information (5-bit-8), l'encodage base64 ainsi que l'encodage *Unix to Unix* (UU) utilisant 6 bits.

Dans le but d'évaluer l'efficacité de la méthode proposée, la vérité terrain, i.e., la classe théorique de chaque chaîne de caractères générée par l'analyse lexicale doit être connue. Une étape préliminaire manuelle est effectuée pour assigner une classe – information ou non information – à chaque chaîne de caractères générée par l'analyse lexicale. Un humain peut difficilement classer (avec une confiance suffisante) les chaînes de caractères qui contiennent plus de 5 lettres. Par conséquent, seulement les chaînes de caractères qui contiennent moins de 5 caractères sont gardées pour évaluer la méthode proposée. Cela ne signifie cependant pas que la méthode proposée n'est pas capable d'assigner une classe aux chaînes de caractères qui sont plus courtes que 5 caractères.

La vérité terrain a été créée en exploitant (i) des spécifications, comme pour les cartes EMV [59] et les passeports électroniques [60] ; (ii) des précédentes analyses ad hoc, comme pour les cartes de transport [61, 30] et les forfaits de ski [10] ; (iii) des outils interprétant les données des cartes à puce, tels que Cardpeek [2] et l'application du lecteur de carte d'identité électronique belge [62].

Considérant toutes les fonctions de décodage et les possibles décalages, l'analyse lexicale génère 190 093 chaînes de caractères comme présenté dans le tableau 3.1 où

les colonnes I et $\overline{I}$ représentent la vérité terrain. L'analyse lexicale génère donc 1 133 chaînes de caractères qui sont des informations, ce qui représente $0,5\%$ des chaînes générées au total.

Tableau 3.1 – Chaînes de caractères générées par l'analyse lexicale

Décodage	Chaînes	I	$\overline{I}$
ASCII-8	1 017	421	596
ASCII-7	3 462	0	3 462
5-bit	58 001	171	57 830
5-bit-8	53 772	533	53 239
UU	2 872	8	2 864
Base64	70 969	0	70 969
Total	190 093	1 133	188 960

3.4.2 Métriques d'évaluation

Soit S un ensemble de t chaînes de caractères $S = \{s_i,\ 1 \leq i \leq t\}$ et C un ensemble de classes $C = \{\mathrm{I}, \bar{\mathrm{I}}\}$, ℓ_i une étiquette qui est assignée à chaque chaîne, L un ensemble de t étiquettes représentant la vérité terrain tel que

$$L = \{\ell_i = \mathrm{Class}(s_i) \in C,\ 1 \leq i \leq t\}.$$

L'ensemble S contient t_{I} informations et $t_{\bar{\mathrm{I}}}$ qui ne sont pas des informations avec $t_{\mathrm{I}} + t_{\bar{\mathrm{I}}} = t$.

On définit l'ensemble $P = \{p_i,\ 1 \leq i \leq t\}$ représentant les prédictions de l'analyse syntaxique où :

$$p_i = \begin{cases} \mathrm{I}, & \text{si } \Pr(\vec{x}_i \mid \mathrm{I}) > \Pr(\vec{x}_i \mid \bar{\mathrm{I}}) \\ \bar{\mathrm{I}}, & \text{sinon;} \end{cases}$$

où $\vec{x}_i$ représente les caractéristiques de chaque chaîne de caractères s_i. Soit V le vecteur qui représente le succès de la prédiction : $V = \{v_i,\ 1 \leq i \leq t\}$ où :

$$v_i = \begin{cases} 1, & \text{si } p_i = \ell_i \\ 0, & \text{sinon.} \end{cases}$$

Soit V^c le vecteur représentant le succès de la prédiction pour une classe c donnée, $V^c = \{v_i^c,\ 1 \leq i \leq t\}$ où chaque v_i^c est calculé de la façon suivante :

$$v_i^c = \begin{cases} 1, & \text{si } p_i = \ell_i = c \\ 0, & \text{sinon.} \end{cases}$$

Soit E^c un vecteur d'erreur de classe $E^c = \{e_i^c,\ 1 \leq i \leq t\}$ représentant l'erreur pour une classe donnée c où chaque e_i^c est calculé comme suit :

$$e_i^c = \begin{cases} 1, & \text{si } p_i \neq \ell_i \text{ et } \ell_i = c \\ 0, & \text{sinon.} \end{cases}$$

Ainsi,

$$\forall i,\ 1 \leq i \leq t,\ \sum_{c \in \{\mathrm{I}, \bar{\mathrm{I}}\}} (v_i^c + e_i^c) = 1.$$

Le taux de reconnaissance de l'analyse syntaxique pour toutes les classes est calculé de la manière suivante :

$$RR = \frac{\sum_{i=1}^{t} v_i}{t}.$$

Le taux de reconnaissance de l'analyse syntaxique pour une classe donnée c est :

$$RR_c = \frac{\sum_{i=1}^{t} v_i^c}{t_c}.$$

Le taux de reconnaissance équilibré représentant la moyenne des taux de reconnaissances de chaque classe est calculé comme suit :

$$BRR = \frac{\sum_{c \in C} RR_c}{\mid C \mid}.$$

Notons que le BRR est préféré au RR quand la proportion de chaînes de caractères qui sont des informations diffère significativement de la proportion de celles qui ne sont pas des informations dans l'ensemble des chaînes générées par l'analyse lexicale.

Enfin, la précision représente la proportion d'information (selon la vérité terrain) parmi les chaînes de caractères classées comme étant des informations par la méthode proposée. La précision est donc calculée comme suit :

$$PR = \frac{\sum_{i=1}^{t} v_i^{\mathrm{I}}}{\sum_{i=1}^{t} (v_i^{\mathrm{I}} + e_i^{\bar{\mathrm{I}}})}.$$

Une précision de 50% signifie donc que la moitié des chaînes de caractères classées comme étant des informations par la méthode sont des faux positifs. Une précision de 50% couplée à un taux de reconnaissance des informations proche de 100% représente donc un bon résultat.

3.4.3 L'efficacité des classifieurs bayésiens

L'efficacité de l'analyse syntaxique est testée avec un ensemble de chaînes de caractères qui n'appartient pas au corpus d'apprentissage. Cette approche veut démontrer que la méthode proposée peut reconnaître des nouveaux mots. Le taux de succès de l'analyse syntaxique est comparée à celle par dictionnaire uniquement.

Données

On construit un ensemble S contenant 10 000 chaînes de caractères avec 5 000 qui sont des informations et 5 000 qui ne sont pas des informations. L'ensemble contient des chaînes des deux classes dans le but d'évaluer le taux de reconnaissance des informations mais aussi la quantité des faux positifs générés.

Au lieu de créer 5 000 nouveaux mots pour vérifier qu'ils sont effectivement bien reconnus par l'analyse syntaxique, on utilise une approche qui ne modifie pas artificiellement le langage. Les 5 000 chaînes de caractères représentant des informations sont choisies aléatoirement dans le corpus d'apprentissage. Ensuite, il est important de supprimer temporairement ces 5 000 chaînes du corpus pour empêcher une reconnaissance triviale de l'analyse par dictionnaire.

Le corpus d'apprentissage utilisé pour la classe I est une liste de 400 000 mots distincts provenant de deux sources différentes. La première source est un dictionnaire français contenant 150 000 mots (provenant de *John The Ripper* [63]). La seconde source contient plusieurs listes de noms et prénoms, représentant 250 000 mots (provenant principalement de l'*US Census* [64]). D'autres corpus ont été testés, sans amener de meilleurs résultats. Par exemple, l'utilisation d'un dictionnaire anglais de *John The Ripper* en plus de ces deux premières sources n'améliore pas le taux de reconnaissance final des expériences.

Les chaînes de caractères qui ne sont pas des informations devraient représenter les chaînes de caractères qui sont générées en appliquant une fonction de décodage sur des données encodées avec une fonction d'encodage différente. Il est impossible en pratique de créer un tel modèle qui soit représentatif de ces non informations provenant des cartes à puces. Ainsi, ces chaînes de caractères sont considérées comme étant générées par une source aléatoire uniforme. Plus précisément, chaque chaîne de caractères est générée de la manière suivante : une longueur ℓ est choisie de manière aléatoire et uniforme entre 4 et 15, ensuite ℓ lettres de l'alphabet sont choisies elles aussi aléatoirement et sont concaténées pour produire une chaîne de caractères. Si cette chaîne construite appartient au corpus d'apprentissage, alors elle est jetée, et une nouvelle est construite.

Analyse basée sur un dictionnaire

La première étape de l'analyse syntaxique vérifie si la chaîne de caractères contient un mot du dictionnaire ou non. Utiliser des mots d'une longueur trop courte génère beaucoup de faux positifs, et utiliser des mots d'une longueur trop longue aura pour conséquence de manquer des informations. Les expériences ont montré que la longueur optimale à utiliser est d'au moins 5 caractères (5 inclue) comme présenté sur la figure 3.2.

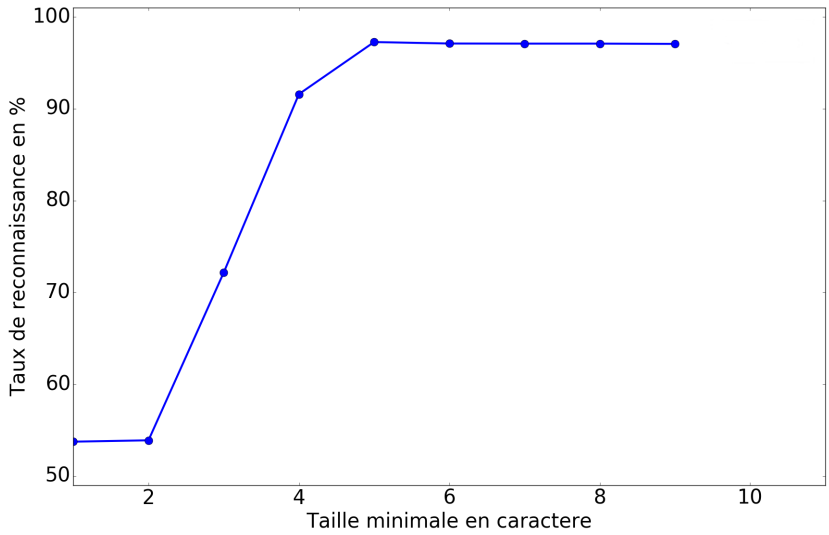


FIGURE 3.2 – Évolution du taux de reconnaissance de l'analyse syntaxique en fonction de la taille minimale des mots utilisés pour l'analyse par dictionnaire

Le tableau 3.2 présente les résultats de l'utilisation seule d'une analyse dictionnaire appliquée sur l'ensemble S. Cette analyse reconnaît plus de 76% des informations et ne génère aucun faux positif. Les chaînes de caractères de la classe $\bar{\text{I}}$ reconnues comme I sont les faux positifs. Dans la table, RR_c représente le taux de reconnaissance de la classe c, RR représente la moyenne des taux de reconnaissances des classes I et $\bar{\text{I}}$.

Tableau 3.2 – Résultat de l'analyse par dictionnaire appliquée sur S

RR_{I}	$RR_{\bar{\text{I}}}$	RR
$76,30\%$	$100,0\%$	$88,15\%$

Ajout du classifieur bayésien

Le tableau 3.3 présente les résultats de l'application de l'analyse syntaxique sur l'ensemble S. Les expériences ont été réalisées avec 10 ensembles S différents. La pro-

babilité Pr(I) utilisée est ici $\Pr(\mathrm{I}) = 0,5$. L'analyse atteint un taux de reconnaissance plus élevé que 98% en moyenne pour chaque classe et avec un intervalle de confiance inférieur à $0,1\%$ en utilisant un niveau de confiance de 95%. Le taux de reconnaissance est amélioré de plus de 20% comparé à l'analyse précédente utilisant seulement le dictionnaire. D'un autre côté, cela génère des faux positifs, mais ils représentent seulement $1,5\%$ du nombre total de chaînes qui ne sont pas des informations qu'il y avait à classer.

Tableau 3.3 – Résultats de l'analyse par dictionnaire combinée avec le classifieur bayésien appliquée sur l'ensemble S

RR_{I}	$RR_{\overline{\mathrm{I}}}$	RR
$98,8\% \pm 0,1$	$98.5\% \pm 0,1$	$98,7\% \pm 0,1$

3.4.4 Résultats d'expériences de l'analyse syntaxique

L'analyse syntaxique est appliquée sur les chaînes de caractères générées par l'analyse lexicale sur des dumps provenant de cartes à puce réelles. Notons qu'ici l'analyse multi-dump n'est pas encore appliquée. La probabilité Pr(I) utilisée pour le classifieur bayésien ici est $0,5\%$, ce qui correspond à la proportion de chaînes de caractères étant des informations parmi les chaînes générées. Les résultats sont présentés dans le tableau 3.4.

Tableau 3.4 – Taux de succès de l'analyse syntaxique

Décodage	$\%RR_{\mathrm{I}}$	$\%RR_{\overline{\mathrm{I}}}$	%BRR	%PR
ASCII-8	99,76	93,12	96,44	91,11
ASCII-7	NA	97,40	NA	NA
5-bit	100,0	96,73	98,37	8,30
5-bit-8	98,69	96,9	97,79	24,13
UU	100,0	98,29	99,14	14,03
Base64	NA	94,94	NA	NA
Moyenne	99,29	96,13	97,71	13,33

En utilisant la fonction de décodage 5-bit, l'analyse syntaxique reconnaît 26 informations différentes, représentant des noms et prénoms provenant des dumps des cartes Calypso et des dumps de carte Vitale. Les faux positifs générés représentent 1 888 chaînes de caractères, alors que l'analyse lexicale avait généré 58 001 chaînes de caractères soit $3,25\%$.

En moyenne, appliquer l'analyse lexicale avec la fonction de décodage 5-bit sur un seul dump Calypso génère 83 chaînes de caractères. L'analyse syntaxique classe 4

chaînes de caractères comme étant des informations. Seules 2 d'entre elles représentent réellement des informations textuelles qui sont le nom et le prénom du possesseur de la carte, les 2 autres sont des faux positifs.

En utilisant la fonction de décodage ASCII-8, 134 chaînes de caractères différentes sont classées comme étant des informations, alors qu'il y en avait 135 à retrouver. Cela représente donc plus de 99% de taux de reconnaissance. La seule information qui n'est pas reconnue est un nom contenant des n-grammes qui sont rares tel que "uza". Il y a seulement 41 faux positifs qui sont générés parmi les 1 017 chaînes de caractères générées par l'analyse lexicale, soit $4,0\%$. Les informations retrouvées sont des noms de famille, des prénoms, des adresses postales ou électroniques et des chaînes de caractères liées à l'application (e.g., "repository", "signature", etc.)

En moyenne, appliquer l'analyse lexicale avec la fonction de décodage ASCII-8 sur un dump EMV génère une dizaine de chaînes de caractères dont seulement 4 sont différentes. Dans la plupart des cas, l'analyse syntaxique ne génère aucun faux positif. Les informations retrouvées sont le nom du possesseur et son prénom, ainsi que quelques informations à propos de l'application : "transaction", "mastercard", "debit", etc.

En utilisant la fonction de décodage UU, l'analyse syntaxique reconnaît 8 informations différentes représentant des adresses postales dans la carte Vitale. Les faux positifs générés représentent au total 49 chaînes de caractères parmi les 2 872 qui ont été générées par l'analyse lexicale, soit $1,71\%$.

La fonction de décodage 5-bit-8 génère au moins les mêmes chaînes de caractères que la fonction de décodage ASCII-8 en raison de sa définition. Le taux de reconnaissance est plus bas que celui de l'ASCII-8 car les chaînes de caractères générées sont plus souvent bruitées. En effet, comparé à l'ASCII-8 une proportion plus importante des entrées génère un caractère. Appliquer l'analyse syntaxique génère 1 654 faux positifs parmi les 53 772 chaînes de caractères générées par l'analyse lexicale, soit $3,07\%$.

Il n'y a pas d'informations qui sont stockées avec la fonction d'encodage ASCII-7 dans les dumps considérés. L'analyse syntaxique génère 90 faux positifs parmi les 3 462 chaînes de caractères générées par l'analyse lexicale, soit $2,60\%$.

Il n'y a pas non plus d'informations qui sont encodées à l'aide de base64 dans les dumps considérés. L'analyse syntaxique génère 2 188 faux positifs parmi les 70 969 chaînes de caractères générées par l'analyse lexicale, soit $3,08\%$.

L'analyse lexicale génère 190 093 chaînes de caractères, l'analyse syntaxique en élimine 181 654 qui ne sont pas des informations. Il reste donc 8 439 chaînes de caractères qui sont considérées comme des informations, 1 125 d'entre elles sont des

informations (selon la vérité terrain), les 7 314 autres sont des faux positifs.

On peut noter que certaines fonctions (ASCII-7, base64) de décodage ne génèrent aucune information sur l'ensemble de dumps considéré. Néanmoins, il est intéressant de voir combien de faux positifs elles génèrent. La plupart des faux positifs sont générés par les fonctions de décodage où la proportion de caractères alphabétiques parmi les sorties possibles est importante telles que les encodages 5-bit, 5-bit-8 et base64. La plupart des faux positifs sont plus courts que 7 caractères tels que "cogra", "darat", "ecran", etc. Ces mots courts sont fait de n-grammes communément utilisés par les mots des dictionnaires de langues ou même des mots de certains dictionnaires comme le mot "ecran". Les faux positifs les plus longs sont composés de sous séquences du corpus d'apprentissage comme par exemple "tqqimainsqdlkjn" qui contient le mot "mains".

La plupart de ces faux positifs sont ensuite éliminées par l'analyse multi-dump. Le tableau 3.5 effectue un bilan des informations retrouvées dans les 371 dumps.

Tableau 3.5 – Informations retrouvées en utilisant la méthode proposée sur les 371 dumps

Application	Champs d'informations et chaînes de caractères retrouvées
Carte d'accès d'hôpital	<nom de famille>, <prénom>
Carte de membre UCL	<nom de famille>, <prénom>, universite, catholique, louvain, officer, policy, security
Calypso	<nom de famille>, <prénom>
Passeport	<nom de famille>, <prénom>, <Lieu de naissance>, <Lieu de création>, Belgium, kingdom, affaires, service, federal, foreign, internal authenticate
Cartes de salons professionnels	<nom de famille>, <prénoms>, <adresse mail>, <adresse de l'entreprise>, <nom de l'entreprise>, <nom du salon>, France
Cartes EMV	<nom de famille>, <prénom>, VISA, Mastercard, transaction
Carte Vitale	<nom de famille>, <prénom>, <noms des enfants> <prénoms des enfants>, Vitale, health, photo, datauser
Carte eID belge	<nom de famille>, <prénom>, <Lieu de naissance>, <Lieu de création>, Belgium, repository, signature, authentication, citizen
NFC action launcher	quitter, android, bureau

3.4.5 Résultats d'expériences de l'analyse multi-dump

Pour évaluer l'efficacité de l'analyse multi-dump, les 371 dumps sont regroupés en différents ensembles selon l'application à laquelle ils appartiennent (e.g, EMV, passeports, Calypso, etc.). Plus de 50 ensembles sont ainsi construits. Chaque ensemble qui contient moins de 3 dumps est ignoré. On construit ainsi 25 ensembles qui respectent cette contrainte. Ces 25 ensembles totalisent un nombre de 287 dumps.

Les analyses lexicales et syntaxiques sont d'abord appliquées sur ces 25 ensembles de dumps, puis l'analyse multi-dump est effectuée. Les résultats sont fournis dans le tableau 3.6 pour l'analyse syntaxique et le tableau 3.7 pour l'analyse multi-dump.

Tableau 3.6 – Résultats de l'analyse syntaxique appliquée sur les 287 dumps sélectionnés

Fonction	$\%RR_{\mathrm{I}}$	$\%RR_{\bar{\mathrm{I}}}$	%BRR	%PR
ASCII-8	$100,0$	$99,00$	$99,50$	$98,30$
ASCII-7	NA	$98,28$	NA	NA
5-bit	$100,0$	$96,82$	$98,42$	$21,63$
5-bit-8	$100,0$	$96,44$	$98,22$	$10,71$
UU	NA	$97,88$	NA	NA
Base64	NA	$94,25$	NA	NA
Moyenne	$100,0$	$95,80$	$97,90$	$13,24$

Notons que le tableau 3.4 diffère légèrement du tableau 3.6 car le tableau 3.4 représente l'analyse syntaxique appliquée sur les 371 dumps alors que le tableau 3.6 représente l'application de cette dernière sur les 287 dumps sélectionnés pour l'analyse multi-dump.

L'analyse syntaxique génère 2 358 faux positifs lorsqu'elle est appliquée sur les 287 dumps sélectionnés. Toutes les informations sont retrouvées mais la précision est de $13,24\%$. C'est-à-dire que parmi les chaînes de caractères considérées comme des informations par l'analyse syntaxique, $86,76\%$ sont en fait des faux positifs.

Le tableau 3.7 présente les résultats de l'analyse multi-dump en vérifiant les trois assertions. L'application de l'analyse multi-dump réduit le nombre de faux positifs de 2 358 à seulement 334 faux positifs, ce qui correspond à moins de 2 faux positifs par dump en moyenne. La précision s'améliore ainsi, atteignant $51,87\%$. Cela signifie que parmi les chaînes de caractères considérées comme des informations par l'analyse, moins d'une sur deux est un faux positif. La plupart des faux positifs générés par cette analyse sont dus aux zones de données qui sont partiellement identiques dans tous les dumps de la même application. Ce qui fait que les chaînes de caractères générées sur ces données posséderont des n-grammes en commun. Si ces n-grammes

sont fréquents dans les mots des dictionnaires considérés alors l'analyse multi-dump ne peut pas améliorer les résultats de l'analyse syntaxique.

Le vote majoritaire a aussi été testé pour l'assertion `Assert1`, mais la précision ainsi obtenue est inférieure de 10% de celle présente ici.

Tableau 3.7 – Résultats de l'analyse multi-dump appliquée sur les 287 dumps sélectionnés

Fonction	$\%RR_{\mathrm{I}}$	$\%RR_{\bar{\mathrm{I}}}$	%BRR	%PR
ASCII-8	100, 0	100, 0	100, 0	100, 0
ASCII-7	NA	100, 0	NA	NA
5-bit	100, 0	99, 90	99, 95	89, 53
5-bit-8	100, 0	99, 27	99, 64	55, 43
UU	NA	100, 0	NA	NA
Base64	NA	99, 05	NA	NA
Moyenne	100, 0	99, 42	99, 71	51, 87

3.4.6 Résultats d'expériences ignorant les données cryptographiques

Dans cette section, les résultats du chapitre 2 sont utilisés. Avant d'appliquer l'analyse lexicale sur un dump, les tests statistiques retenus par la méthode sont appliqués. Ensuite, à l'aide du classifieur entraîné par le boosting les données du dump sont classifiées dans les classes *cryptographique* et *faisant sens*. Ainsi, les chaînes de caractères contenant seulement des bits classés comme *cryptographique* par le classifieur sont ignorés. Il est ainsi espéré une réduction du nombre de faux positifs générés par l'analyse lexicale.

Tableau 3.8 – Comparaison des résultats avec et sans la prise en compte des données cryptographiques

	Chaînes	Faux positifs	Informations retrouvées
Avec données crypto.	190 093	7 314	1 125
Sans données crypto.	131 290	5 532	1 120

Le tableau 3.8 montre la différence des résultats lorsqu'on ignore les chaînes de caractères provenant des données cryptographiques. La première colonne indique le nombre de chaînes de caractères générées par l'analyse lexicale, la deuxième colonne donne le nombre de faux positifs restants après application de l'analyse syntaxique, et la dernière colonne fournit le nombre d'informations retrouvées par l'analyse syntaxique.

Il y a 58 803 chaînes de caractères générées en moins. Cela représente une réduction de $30,93\%$ du nombre de chaînes de caractères. Malheureusement, parmi ces chaînes de caractères ignorées, 5 n'auraient pas dû l'être car elles représentent des informations. Cela est expliqué par le fait que la technique de classification proposée génère quelques faux négatifs. Néanmoins, ces 5 informations ignorées ne représentent que $0,44\%$ des informations de ces cartes. Ainsi, $99,29\%$ des informations sont tout de même retrouvées en ignorant les données cryptographiques. Le résultat le plus intéressant est la diminution du nombre de faux positifs, il est réduit de 1 782, soit une réduction de $24,36\%$.

Lorsque l'analyse multi-dump est combinée avec cette analyse des données cryptographiques, les résultats sont identiques à ceux utilisant l'analyse multi-dump seulement. En fait, l'analyse multi-dump est déjà très efficace sur l'élimination des faux positifs contenus dans les zones contenant des données cryptographiques.

3.5 Conclusion

Dans ce chapitre, une méthode en trois étapes est proposée pour retrouver les informations textuelles dans les dumps de mémoire provenant de cartes à puce. La première étape est une analyse lexicale qui génère des chaînes de caractères à partir des données binaires. La seconde étape est une analyse syntaxique qui élimine les chaînes ne faisant pas sens. Cette étape utilise une analyse basée sur un dictionnaire ainsi qu'un classifieur bayésien étudiant les n-grammes de la chaîne de caractères pour calculer des scores et décider si elle fait sens ou non. Enfin, une analyse multi-dump exploite une spécificité des cartes à puce en combinant les résultats de l'analyse syntaxique sur plusieurs dumps d'une même application.

Cette méthode en trois étapes atteint un taux de succès élevé. Des expériences ont été effectuées sur plus de 371 dumps de la vie réelle. Elles montrent que la méthode peut automatiquement retrouver plus de 99% des informations textuelles disponibles dans un seul dump tout en gardant un taux de faux positif aussi faible que $5,5\%$. Lorsqu'au moins 2 dumps de la même application sont disponibles, alors l'analyse multi-dump peut être appliquée, ce qui réduit le taux de faux positifs à $0,6\%$ sans diminuer le taux de reconnaissance des informations. Pour illustrer son efficacité, l'analyse multi-dump génère moins de 2 faux positifs par dump et retrouve toutes les informations qui y sont stockées lorsqu'elle est appliquée sur les 25 applications.

Chapitre 4

Recherche des dates

Résumé

Ce chapitre introduit une méthode qui retrouve automatiquement les dates stockées dans les dumps de mémoire des cartes à puce. Étant donné que la structure et l'encodage des données sont inconnus, cette méthode exploite les spécificités des cartes à puces pour éliminer les faux positifs. Elle utilise une analyse multi-dump enrichie par des informations contextuelles. Les expériences effectuées sur 371 dumps de cartes à puce ont révélé que la méthode permet de retrouver automatiquement les dates présentes dans les dumps tout en générant très peu de faux positifs.

Sommaire

4.1 Introduction

Les dates qui peuvent typiquement être retrouvées dans les cartes à puce incluent : les dates de naissance (e.g., celle du possesseur de la carte), de création, celles liées à un abonnement ou aux dernières actions effectuées avec la carte.

La méthode proposée dans le chapitre 3 ne peut être appliquée pour retrouver des dates. En effet, l'élimination des faux positifs repose sur l'analyse des n-grammes des chaînes de caractères générées. Ainsi, les chaînes de caractères trop éloignées des

mots du dictionnaires sont éliminées. Elles représentent les chaînes de caractères qui ne semblent pas avoir de signification. Dans le cas des dates, outre le fait que les dates ne peuvent pas être découpées en n-grammes, il semble difficile de dire si une date a du sens ou non en dehors de son contexte.

Ce chapitre introduit une méthode automatique qui retrouve les dates stockées dans les dumps obtenus à partir de mémoires non volatiles de cartes à puce. La méthode est composée de trois étapes : décodage, analyse multi-dump et analyse contextuelle. La première étape décode les données binaires du dump pour générer des dates. La seconde étape est une analyse multi-dump éliminant une partie des faux positifs générés en analysant plusieurs dumps de la même application. La dernière étape est une analyse contextuelle éliminant les dates ne respectant pas les informations contextuelles collectées (e.g., informations imprimées sur le ticket). À noter que la deuxième étape ne génère aucun faux négatif (i.e., elle ne peut pas supprimer par erreur une date).

Il existe de nombreuses fonctions d'encodage pour stocker les dates en mémoire. Cette fonction peut être un compteur représentant une quantité de temps depuis une certaine date, telle que le nombre de secondes écoulées depuis le 1er janvier 1970. L'unité de temps et la date de départ du compteur peuvent varier. Il existe donc, en théorie, un nombre infini de fonctions de décodage possible. L'application de ces nombreuses fonctions de décodage sur un dump génère des centaines ou des milliers de dates selon la taille du dump. Des heuristiques simples telles qu'ignorer les séquences contenant seulement des bits à 0 ne réduisent que très légèrement la complexité du problème. Une méthode automatique éliminant ces faux positifs est donc nécessaire.

Appliquée sur des dumps de la vie réelle, la méthode proposée retrouve les dates dans plusieurs applications différentes tout en générant très peu de faux positifs. Cette méthode retrouve, entre autres, la date de naissance du possesseur et la date de création des documents officiels (e.g., passeport électronique ou carte d'identité électronique), les dates des transactions dans les cartes bancaires, et les dates des derniers voyages effectués dans les cartes de transport.

Le chapitre est structuré de la manière suivante. La section 4.2 décrit la méthode proposée générant les dates à partir des données binaires et éliminant les faux positifs. Les résultats expérimentaux et la validation de la méthode sur les 371 dumps sont présentés en section 4.3.

4.2 Méthode proposée

Cette section présente la méthode en trois étapes retrouvant les dates stockées dans les dumps. La méthode exploite les spécificités des mémoires des cartes à puce pour éliminer les faux positifs.

4.2.1 Schéma général

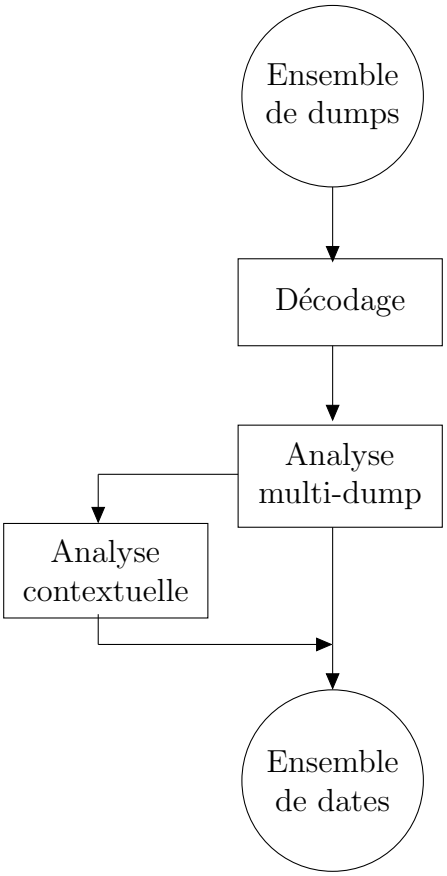


FIGURE 4.1 – Schéma général de la recherche de dates dans un dump de mémoire

Dans un dump de mémoire, chaque bit peut-être le bit de poids fort de la séquence binaire encodant une date. Le couple (indice du bit, fonction de décodage) est alors appelé un *candidat*. Lors du décodage du dump, chaque candidat génère une date. Une méthode est donc proposée pour éliminer tous les candidats qui ne sont finalement pas utilisés par l'opérateur pour stocker une date. Un tel candidat génère donc un *faux positif* par dump lors du décodage. Par extension, ce candidat est aussi appelé *faux positif*.

La méthode est séparée en trois étapes. La première consiste à appliquer toutes les fonctions de décodage potentielles sur les dumps d'une même application, générant une date par candidat et par dump. La seconde étape exploite les relations entre les dumps d'une même application avec une analyse multi-dump, éliminant une partie des faux positifs générés. La troisième étape exploite les informations contextuelles, elle élimine les candidats ne respectant pas les contraintes exprimées par les informations contextuelles collectées. Cette dernière étape peut être appliquée plusieurs fois sur une même application avec des contraintes différentes selon le type de date que l'on recherche (e.g., date de trajet, date de naissance). Cette méthode retourne les candidats non éliminés après toutes les analyses. Les résultats exploités par un enquêteur sont les dates générées sur chaque dump lors du décodage de ces candidats. Dans la suite de ce chapitre, l'opérateur humain qui analyse les résultats est appelé *enquêteur*.

La figure 4.1 illustre la méthode proposée. Il existe deux chemins amenant à l'ensemble des dates retournées, car l'analyse contextuelle est facultative, il est possible de n'avoir aucune information contextuelle à propos des dumps.

4.2.2 Décoder un dump

Soient m un entier et f une fonction de décodage telle que

$$f : \{0,1\}^m \rightarrow \ \mathcal{D} \cup \{0,\ 1,\ -1\}.$$

Soit $\mathcal{D}$ l'ensemble des dates existantes du calendrier grégorien. La valeur 0 (resp. 1) est générée par l'entrée 0^m (resp. 1^m). Ces sorties peuvent correspondre à des données non utilisées au moment de la création du dump. Par exemple, des informations à propos du troisième voyage qui n'aurait pas encore été effectué dans un dump Calypso (cf la figure 1.9).

La valeur -1 représente les sorties non conformes avec le calendrier grégorien, par exemple, en utilisant la fonction de décodage ASCII, la sortie peut contenir des lettres au lieu de ne contenir que des chiffres.

Soient $D_1, \ldots, D_N$ les dumps de l'application $\mathcal{A}$. Chaque dump D_i est un dump de longueur n bits, qui est représenté par la séquence binaire$(b_k^i)_{1 \leq k \leq n}$. Appliquer la fonction de décodage f sur une séquence binaire du dump D_i avec $1 \leq i \leq N$, en utilisant l'indice j comme bit de poids fort de la séquence avec $1 \leq j \leq n - m + 1$ génère une date $d_{i,j}^f$:

$$d_{i,j}^f = f\left(\left(b_k^i\right)_{j \leq k < j+m}\right).$$

Maintenant, plutôt que d'analyser les ensembles des sorties générées sur chaque dump D_i de l'application, on se concentre sur les sorties générées par chaque candidat (f, j) sur tous les dumps de l'application $\mathcal{A}$. Avec j qui est l'indice auquel est appliquée la fonction de décodage f. Cet ensemble des sorties est représenté par $\{d^f_{i,j}\}$ avec $1 \leq i \leq N$. Il est généré pour chaque fonction de décodage f considérée et chaque indice j possible $1 \leq j \leq n - m + 1$. Le tableau 4.1 illustre les résultats de l'application de deux fonctions de décodage sur une application contenant N dumps de longueur n bits.

Tableau 4.1 – Résultats obtenus après le décodage des dumps d'une application avec deux fonctions de décodage

		Sorties		
Fonction	Candidat	D_1	...	D_N
f_1	$(f_1, 1)$	$d^{f_1}_{1,1}$	...	$d^{f_1}_{1,N}$
	$(f_1, 2)$	$d^{f_1}_{2,1}$		$d^{f_1}_{2,N}$
	...	...		...
	$(f_1, n-m+1)$	$d^{f_1}_{n-m+1,1}$		$d^{f_1}_{n-m+1,N}$
f_2	$(f_2, 1)$	$d^{f_2}_{1,1}$	...	$d^{f_2}_{1,N}$
	$(f_2, 2)$	$d^{f_2}_{2,1}$		$d^{f_2}_{2,N}$
	...	...		...
	$(f_2, n-m+1)$	$d^{f_2}_{n-m+1,1}$		$d^{f_2}_{n-m+1,N}$

4.2.3 Analyse multi-dump

Cette section exploite les relations existantes entre plusieurs dumps provenant de la même application. Considérant un ensemble de valeurs $\{d^f_{i,j}\}$ avec f une fonction de décodage, $1 \leq i \leq N$ et $1 \leq j \leq n - m + 1$ obtenu par la phase de décodage, l'analyse multi-dump élimine une partie des faux positifs générés par le décodage.

La section 1.4.1 présente les relations entre les dumps provenant de la même application. Ces relations impliquent qu'un candidat qui génère des dates respectant une des deux assertions suivantes est un faux positif, et est donc éliminé.

$$\texttt{Assert1}.\ \exists\ i \in \{1 \dots N\}\ t.q.\ d^f_{i,j}\ =\ -1$$

$$\texttt{Assert2}.\ \forall\ i \in \{1 \dots N\},\ d^f_{i,j}\ \in\ \{0,\ 1\}$$

Les assertions doivent être vérifiées dans cet ordre : `Assert1` puis `Assert2`. La première assertion `Assert1` élimine les candidats générant au moins une sortie qui n'est ni une date, ni 0 ni 1. La seconde assertion `Assert2` élimine les candidats générant seulement des 0 et des 1 pour toutes les sorties. Les sorties 0 et 1 sont autorisées pour un nombre de candidat strictement inférieur à N. Ces candidats représentent généralement des zones de mémoires non utilisées par l'opérateur. Les sorties 0 et 1 sont autorisées pour un candidat, car ils peuvent correspondre à un champ contenant des informations liées à un événement qui n'est pas encore survenu (e.g., un trajet). Remarquons que l'analyse multi-dump ne peut pas générer de faux négatifs, c'est-à-dire qu'elle ne peut pas supprimer un candidat qui serait effectivement utilisé par l'opérateur pour enregistrer une information.

4.2.4 Analyse contextuelle

Cette section décrit comment les informations contextuelles liées aux dumps sont exploitées dans la méthode proposée. Les contraintes sur les dates générées par les candidats sont exprimées à partir des informations contextuelles. Lors du décodage les candidats générant des dates qui ne respectent pas ces contraintes sont éliminés. Pour une question de simplicité, l'analyse est séparée en deux filtres différents. La première contrainte élimine les candidats générant des dates qui n'appartiennent pas à un intervalle de temps défini et la deuxième élimine les candidats générant des dates qui ne respectent pas les relations existantes entre les dumps de l'application.

Pour retrouver un champ (e.g., une date de naissance), plusieurs contraintes peuvent être exprimées simultanément à partir des informations contextuelles. Aussi, dans le but de retrouver plusieurs champs de l'application, l'analyse contextuelle peut être appliquée plusieurs fois avec des contraintes différentes.

Filtre par intervalle de temps

Certaines informations contextuelles reliées à l'utilisation de la carte ou à son possesseur peuvent être connues : la date d'utilisation de la carte lors d'un transport public, d'un paiement, d'un nouvel abonnement, etc. Ces informations contextuelles peuvent être exploitées pour définir un intervalle de temps, ou, au moins, une borne inférieure ou supérieure pour filtrer les dates. Pour chaque sorte d'information contextuelle, une borne *DateMin* et une borne *DateMax* peuvent être assignées à chaque dump de l'application : $[D\mathrm{Min}_i, D\mathrm{Max}_i]$.

Ainsi, tous les candidats (f, j) générant des dates qui ne respectent pas l'assertion suivante sont éliminés.

$$\texttt{Assert3}.\ \forall\ i \in \{1 \dots N\},\ D\mathrm{Min}_i\ \leq\ d^f_{i,j}\ \leq\ D\mathrm{Max}_i$$

Filtre par relation entre les dumps

Une autre façon d'exploiter les informations contextuelles et de définir une relation entre les dumps est la suivante : un enquêteur peut savoir qu'une carte donnée a été utilisée deux jours consécutifs dans un transport public. Cela peut aussi être utile pour rechercher des dates reliées au possesseur ou à la carte, quand on sait de quelle carte le dump provient (e.g., pour rechercher la date de naissance ou la date de création).

Pour chaque sorte d'information contextuelle, une relation peut être établie entre chaque paire de dumps de l'application. Cette relation est définie par un (ou plusieurs) opérateur(s) incluant (mais non limité à) : $<$, $>$, $\leq$, $\geq$, $=$, $\neq$.

Ainsi, tous les candidats (f, j) qui ne sont pas conformes avec l'assertion suivante sont éliminés.

$$\texttt{Assert4}.\ \forall\ i_1, i_2 \in \{1 \dots N\};\ d^f_{i_1,j}\ \texttt{OP}_{i_1,i_2}\ d^f_{i_2,j}$$

où $\texttt{OP}_{i_1,i_2}$ définit la relation entre les dumps D_{i_1} et D_{i_2}, où $1 \leq i_1 < i_2 \leq N$.

4.3 Validation expérimentale

Cette section définit le protocole expérimental utilisé et présente les résultats obtenus avec la méthode proposée.

4.3.1 Protocole expérimental

Catégories de fonctions de décodage

Il existe plusieurs façons d'encoder une date, une séparation en deux catégories est proposée : *compteur* et *format*. Les fonctions de décodage de type *compteur* encodent le temps écoulé depuis une certaine date. L'unité de temps peut être, par exemple, le nombre de jours ou de secondes. Les fonctions de décodage de type *format* sont une représentation d'une date du calendrier grégorien sous forme d'une suite de chiffres, où chaque chiffre est stocké en utilisant un encodage tel que BCD ou ASCII. Remarquons que le type *format* peut être vu comme trois *compteurs*, l'un comptant le nombre de jours écoulés depuis le début du mois, l'autre comptant le nombre de mois écoulés depuis le début de l'année, et le dernier comptant le nombre d'années écoulées depuis J.C.

Cette séparation en deux catégories est faite car ces deux types de fonction de décodage ne génèrent pas le même nombre de dates quand ils sont appliqués sur un dump. Par conséquent, ils ne donnent pas le même nombre de faux positifs. En fait, le type *format*, en utilisant l'ASCII et BCD dans le but de représenter une date, a un nombre important d'entrées qui ne génèrent pas des suites de chiffres et par conséquent qui ne génèrent pas de date. Le type *format* `YYYY-MM-DD` utilisant l'ASCII prend comme entrée pour le décodage une séquence binaire de longueur $m = 64$. Dans ce cas, il y a donc $\{0,1\}^m >> |\mathcal{D}|$. Lorsque l'on s'intéresse à l'ASCII utilisant 8 bits, seulement 10 sorties parmi les 256 possibles sont des chiffres. De plus, les suites de chiffres générées ne sont pas nécessairement conformes au calendrier grégorien, par exemple, le 37/01/1995. D'un autre côté, toutes les entrées des fonctions de type *compteur* génèrent des dates conformes avec le calendrier grégorien.

Pour les expériences, les sorties des fonctions de décodage sont redéfinies en autorisant seulement les dates du calendrier grégorien qui sont postérieures au 01/01/1900 et antérieures au 01/01/2050. Cette borne inférieure (resp. supérieure) est choisie car il y a rarement des gens qui sont nés avant (resp. après) et rarement des événements liés à la carte qui ont eu lieu avant (resp. après). Ces bornes ont été choisies de manière arbitraire et peuvent être modifiées selon le contexte. Les sorties qui ne respectent pas ces bornes se voient assigner la valeur -1.

Fonctions de décodage

Une sélection de 25 fonctions de décodage a été effectuée. Ces fonctions sont appliquées sur les 371 dumps afin de valider la méthode proposée. Ces fonctions de décodage sont donc séparées en deux catégories : *compteur* et *format*. Le tableau 4.2 décrit toutes les fonctions de type *compteur* où m est la taille en bits de l'entrée de la fonction et tableau 4.3 décrit toutes les fonctions de type *format* qui ont été utilisées pour les expériences.

Tableau 4.2 – 7 fonctions de décodage de type *compteur*

Date de départ	Unité de temps	Taille de l'entrée (m)
01/01/1990	jours	14, 15
01/01/1997	jours	13, 14, 15
01/01/1997	secondes	31, 32

Pour les *compteurs*, trois dates de départ sont considérées : 01/01/1990 et 01/01/1997 sont considérées en raison de leur présence dans les cartes de transport, et 01/01/1970 est le temps UNIX. Les quantités de bits utilisés pour encoder les dates sont choisies pour représenter des dates jusqu'en 2034 (resp. 2079) pour les

Tableau 4.3 – 18 fonctions de décodage de type *format*

Format	Encodage
DDMMYY	ASCII, BCD, BCD-8
DDMMYYYY	ASCII, BCD, BCD-8
MMDDYY	ASCII, BCD, BCD-8
MMDDYYYY	ASCII, BCD, BCD-8
YYMMDD	ASCII, BCD, BCD-8
YYYYMMDD	ASCII, BCD, BCD-8

fonctions de décodage dont la date de départ est 01/01/1990 utilisant 14 bits (resp. 15 bits). Jusqu'à 2019 (resp. 2041 et 2086) pour les fonctions de décodage dont la date de départ est le 01/01/1997 utilisant 13 (resp. 14 et 15 bits). Jusqu'à 2038 (resp. 2106) pour les fonctions de décodage dont la date de départ est 01/01/1970 utilisant 31 (resp. 32 bits).

Pour les dates de type *format*, il existe plusieurs manières d'encoder chaque chiffre représentant le jour, le mois, ou l'année. ASCII, BCD et BCD-8 sont considérés pour les expériences. BCD-8 est l'encodage BCD complété jusqu'à 8 bits avec 4 bits à zéro. Cet encodage est considéré car un opérateur pourrait forcer les données à être alignées sur l'octet.

Grille Calypso

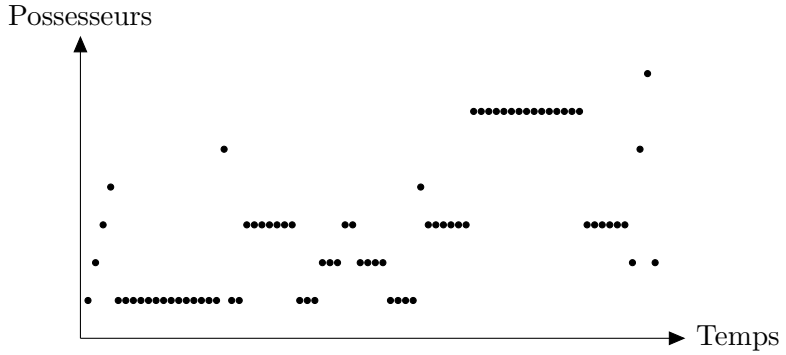


FIGURE 4.2 – Représentation sous forme de grille des 75 dumps Calypso

La figure 4.2 présente les 75 dumps Calypso en fonction de leur date de création et de leur possesseur. L'échelle de temps est seulement chronologique. Ils proviennent tous du même opérateur (i.e., de la même ville) et sont utilisés dans les expériences décrites dans la suite de la thèse. Chaque dump est représenté par un point sur le graphique. Les dumps proviennent de 7 cartes appartenant à 7 possesseurs différents représentés sur l'axe y (i.e., les dumps affichés sur la même ligne proviennent de la

même carte). Chaque carte a été utilisée dans un transport public, au moins une fois, entre chaque création de dump. Cela représente un scénario crédible dans le cas d'un enquêteur actif qui peut contrôler les informations contextuelles liées aux dumps. L'enquêteur décide alors de la carte avec laquelle il veut effectuer une action ainsi que le moment de celle-ci.

4.3.2 Phase de décodage

Dans les sections suivantes, les candidats restants après avoir appliquÃl' les analyses peuvent être séparés en deux catégories. Les candidats réellement utilisés par les opérateurs pour stocker l'information mais aussi ceux représentant les faux positifs que la méthode proposée n'a pas été capable d'éliminer.

Le tableau 4.4 fournit les résultats de l'application de la phase de décodage sur un dump aléatoire et sur deux dumps d'applications de la vie réelle. Les dumps aléatoires contiennent 5 000 bits et leurs données proviennent de `/dev/urandom` d'un système UNIX. Dans un dump aléatoire, les dates restantes sont nécessairement des faux positifs. Décoder un dump aléatoire génère 13 805 dates en moyenne par dump (9 de type *format* et 13 796 de type *compteur*). Les dumps Calypso proviennent de cartes d'abonnement de transport public et contiennent environ 5 000 bits. Décoder un dump Calypso génère en moyenne 5 555 candidats (70 *formats* et 5 485 *compteurs*). OV-chipkaart est le système de ticket de transport public utilisé aux Pays-Bas. Le dump de mémoire d'un ticket OV contient 512 bits. Décoder un dump OV génère en moyenne 1 379 candidats (11 *formats* et 1 368 *compteurs*).

Après la phase de décodage, des milliers de candidats sont générés, ce qui est ingérable pour un enquêteur. Avoir seulement une dizaine de faux positifs par application pourrait être gérable pour un enquêteur.

Tableau 4.4 – Nombre de dates par dump générées par la phase de décodage

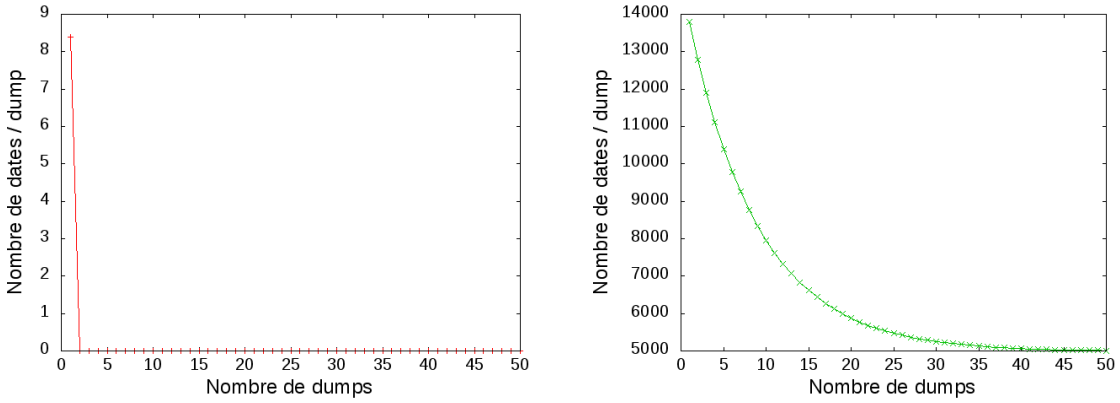
Application	Nombre de dates / dump
Aléatoire	13 805
Calypso	5 555
OV	1 379

4.3.3 Analyse multi-dump

Cette section fournit des résultats sur l'application de l'analyse multi-dump sur des dumps aléatoires, et deux applications de la vie réelle. Les résultats présentés sont l'évolution du nombre de dates en fonction du nombre de dumps utilisés. À

cause de la combinatoire du problème, analyser tous les sous-ensembles de k dumps parmi un ensemble de n dumps est impossible à effectuer en pratique. Ainsi, des sous-ensembles sont tirés aléatoirement pour faire les expériences.

Aléatoire



(a) Dates de type *format*.

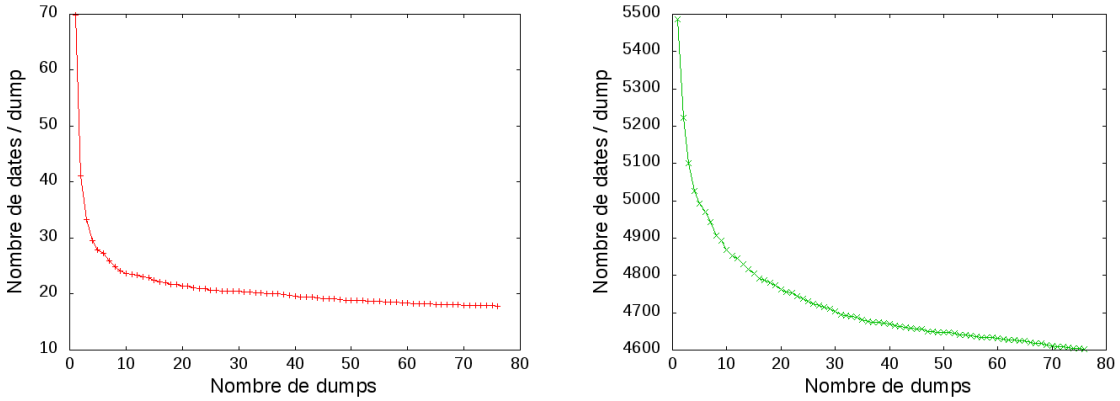
(b) Dates de type *compteur*.

FIGURE 4.3 – Nombre de dates par dump après application de l'analyse multi-dump sur des dumps aléatoires

Les graphiques de la figure 4.3 montrent le nombre de dates restantes par dump en fonction du nombre de dumps utilisé pour effectuer l'analyse multi-dump sur des dumps aléatoires. Appliquer l'analyse multi-dump sur 50 dumps aléatoires génère 5,010 dates par dump en moyenne (toutes de type *compteur*) représentant une réduction de ce nombre de 64% comparé à l'utilisation d'un seul dump. En utilisant seulement 2 dumps, toutes les dates de type *format* sont éliminées.

On remarque que, plus le nombre de dumps est élevé, moins le nombre de dates est réduit quand on ajoute un nouveau dump à l'analyse multi-dump. En effet, en utilisant 20 dumps, il reste 5 881 dates de type *compteur* par dump en moyenne (57% de réduction), et en utilisant 40 dumps il reste 5 065 dates par dump (63% de réduction). Cela s'explique par le fait que la quantité d'information ajoutée par chaque dump supplémentaire est de plus en plus faible.

Il est à noter que, quelques fonctions de décodage (e.g., un *compteur* utilisant 14 bits comptant le nombre de jours écoulés depuis le 01/01/1990) génèrent toujours une date qui est dans le calendrier grégorien. Donc, certains candidats ne peuvent pas être éliminés en utilisant seulement l'analyse multi-dump.



(a) Dates de type *format*.

(b) Dates de type *compteur*.

FIGURE 4.4 – Nombre de dates par dump après l'analyse multi-dump sur Calypso

Calypso

Les graphiques de la figure 4.4 montrent le nombre de dates restantes par dump en fonction du nombre de dumps utilisés pour appliquer l'analyse multi-dump sur des dumps provenant de l'application Calypso.

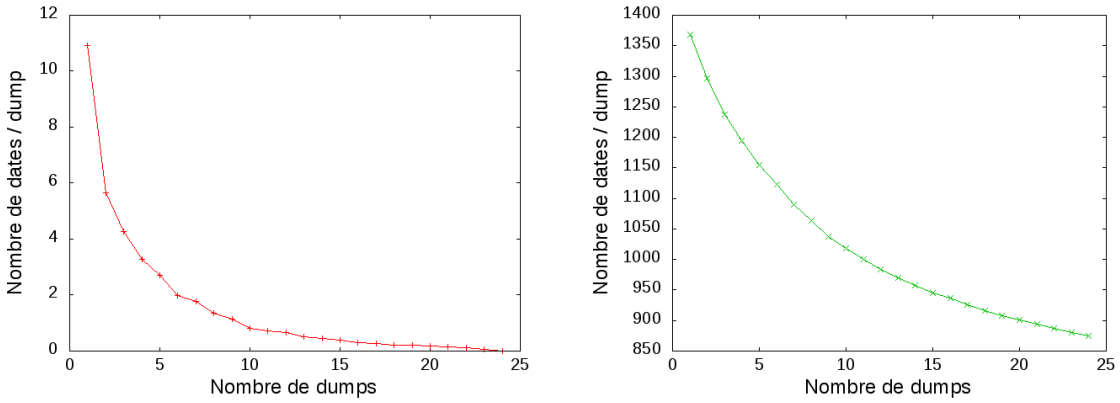
Appliquer l'analyse multi-dump avec 75 dumps Calypso génère 4 620 candidats (18 *formats* et 4 602 *compteurs*), représentant une réduction de 17% (75% pour *format* et 16% pour *compteur*) par rapport à l'utilisation d'un seul dump. En fait, les dates de type *format* restantes représentent des données qui sont identiques dans tous les dumps de l'application. Il est très peu probable de retrouver une information liée à un événement dans de telles données. L'impact d'ajouter un dump quand le nombre de dumps considérés est déjà supérieur à 10 ne réduit que légèrement le nombre de dates par dump. En effet, l'analyse atteint déjà une réduction du nombre de dates de 12% en utilisant 10 dumps et atteint 17% avec 75 dumps.

OV-chipkaart

Les graphiques de la figure 4.5 montrent le nombre de dates restantes par dump en fonction du nombre de dumps utilisés pour appliquer l'analyse multi-dump sur des dumps appartenant à l'application OV. Appliquer l'analyse multi-dump avec 24 dumps OV génère 875 dates (toutes de type *compteur*) représentant une réduction de 37%.

On remarque le même comportement que sur les deux analyses précédentes, le nombre de dates est plus réduit quand on ajoute un dump lorsque le nombre de dumps considérés est bas. Le taux de réduction du nombre de dates sur l'application OV en utilisant 20 dumps est supérieur (environ 35%) comparé à celui sur l'application

Calypso (environ 10%). Cela s'explique par le fait que les données dans l'application OV diffèrent plus entre chaque dump que pour l'application Calypso.



(a) Dates de type *format*.

(b) Dates de type *compteur*.

FIGURE 4.5 – Nombre de dates par dump après l'analyse multi-dump sur OV

Bilan

L'analyse multi-dump n'est pas suffisante pour retrouver des dates dans les applications de la vie réelle. Il y a trop de faux positifs restants : plusieurs milliers pour Calypso, plusieurs centaines pour OV. Néanmoins, cette analyse peut être appliquée sans la connaissance d'information contextuelle et les faux positifs sont tout de même réduits par 16% pour Calypso et par 37% pour OV.

4.3.4 Analyse contextuelle

Cette section fournit des résultats sur l'application de l'analyse contextuelle sur les 371 dumps. Seule l'assertion `Assert3` filtrant les dates par intervalle est utilisée pour éliminer les faux positifs dans les expériences suivantes. L'assertion `Assert4` filtrant les dates selon les relations entre les dumps sera utilisée par la suite.

Statistiques générales

La figure 4.6 montre le nombre de dates obtenues après l'application du filtre par intervalle sur un seul dump de plusieurs applications, en utilisant différents intervalles. Pour réduire le temps de calcul, tous les intervalles possibles ne sont pas testés, des intervalles sont ainsi choisis de manière aléatoire. La borne inférieure de l'intervalle est choisie aléatoirement dans l'intervalle des dates allant du 01/01/2000 au 01/01/2015 et la borne supérieure est fixée en ajoutant un nombre de jours à

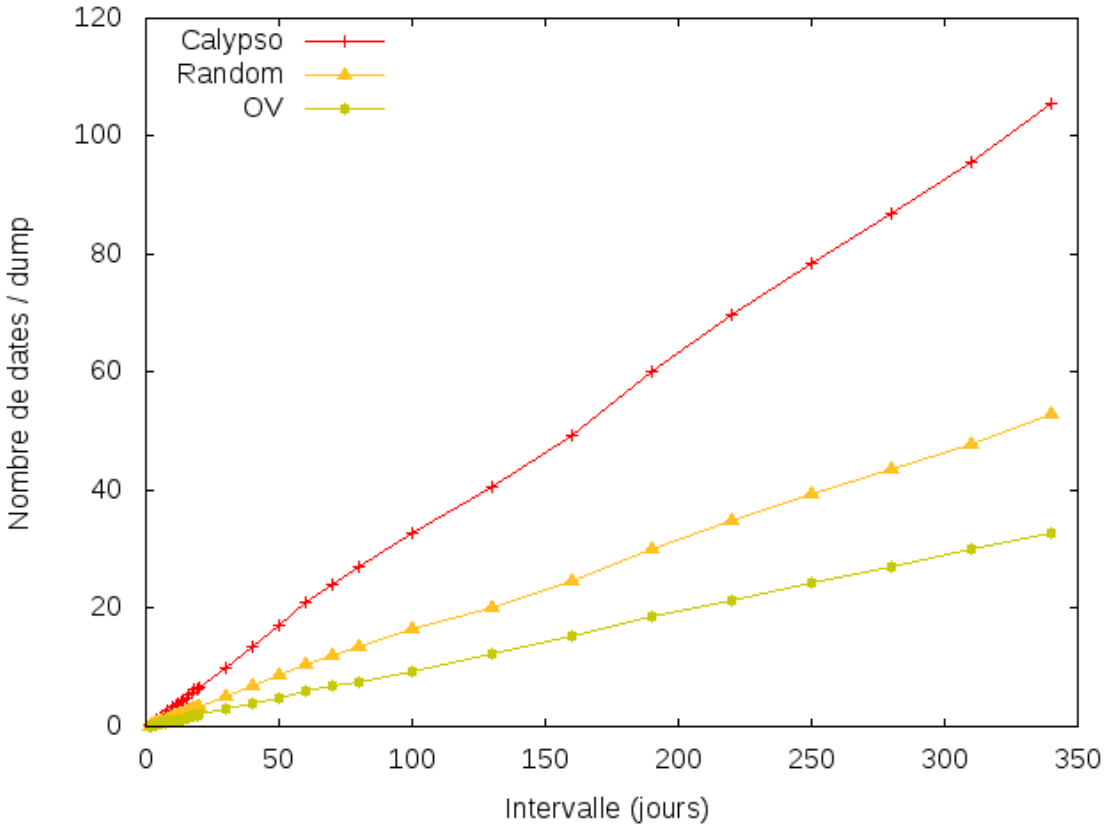


FIGURE 4.6 – Nombre de dates par dump après l'application de l'analyse contextuelle

cette borne inférieure. Cet intervalle ainsi construit peut représenter l'ensemble des dates intéressantes lorsque l'on recherche un événement particulier. Pour les dumps aléatoires, en moyenne, le nombre de dates restantes par dump est linéaire en fonction de la taille de l'intervalle. Le coefficient directeur est le nombre de dates avec un intervalle d'un jour. En moyenne, autoriser un intervalle d'un an génère plus de 100 dates sur un dump Calypso, et environ 30 dates sur un dump OV. Autoriser un intervalle d'un jour génère $0,23$ dates sur un dump Calypso, $0,11$ pour un dump OV, et $0,07$ pour un forfait de ski. Cela signifie qu'une date sur 5 du calendrier grégorien se retrouve dans l'ensemble des dates obtenues par le décodage d'un dump Calypso.

Bien que les faux positifs soient fortement réduits, l'analyse contextuelle appliquée sur un seul dump n'est pas suffisante pour que les résultats soient facilement exploitables par un enquêteur.

Calypso

La figure 4.7 montre le nombre de dates restantes par dump en utilisant différents intervalles pour l'assertion `Assert3` de l'analyse contextuelle, en fonction du nombre

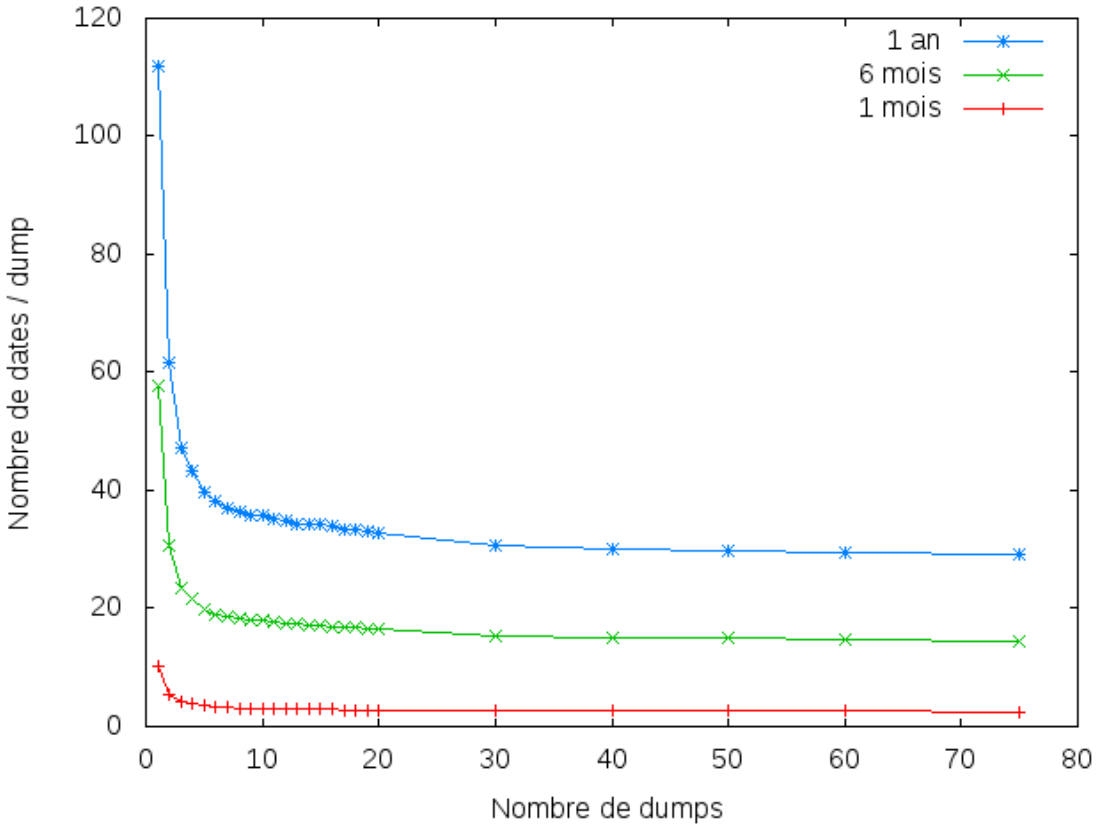


FIGURE 4.7 – Nombre de dates par dump après l'application de l'analyse multi-dump et l'analyse contextuelle sur Calypso

de dumps utilisés pour l'analyse multi-dump. La méthode est très efficace en utilisant jusqu'à $5 - 10$ dumps pour les différents intervalles, divisant par 3 le nombre de dates générées (comparé à l'utilisation d'un seul dump). Par exemple, en utilisant un intervalle d'un an, il reste seulement 40 candidats en utilisant 5 dumps, alors qu'il restait 112 candidats avec seulement 1 dump. Cette quantité de faux positifs, bien qu'un peu élevée, est un résultat intéressant pour un enquêteur disposant de peu d'information contextuelle. L'exploitation de l'assertion `Assert4` pourrait être appliquée pour encore réduire les faux positifs.

OV-chipkaart

En utilisant différents intervalles pour le filtre de l'analyse contextuelle, la figure 4.8 montre le nombre de dates restantes par dump en fonction du nombre de dumps utilisés pour l'analyse multi-dump. Utiliser seulement 2 dumps avec l'analyse contextuelle génère un nombre de dates par dump qui est très faible rendant le résultat exploitable pour un enquêteur. Plus précisément, avec seulement 2 dumps, il

y a moins de 1 date par dump avec un intervalle de 1 mois, et seulement 7 dates par dump pour un intervalle de 1 an. Pour obtenir, en moyenne, moins de 1 date par dump avec un intervalle de 1 an, seulement 20 dumps sont nécessaires.

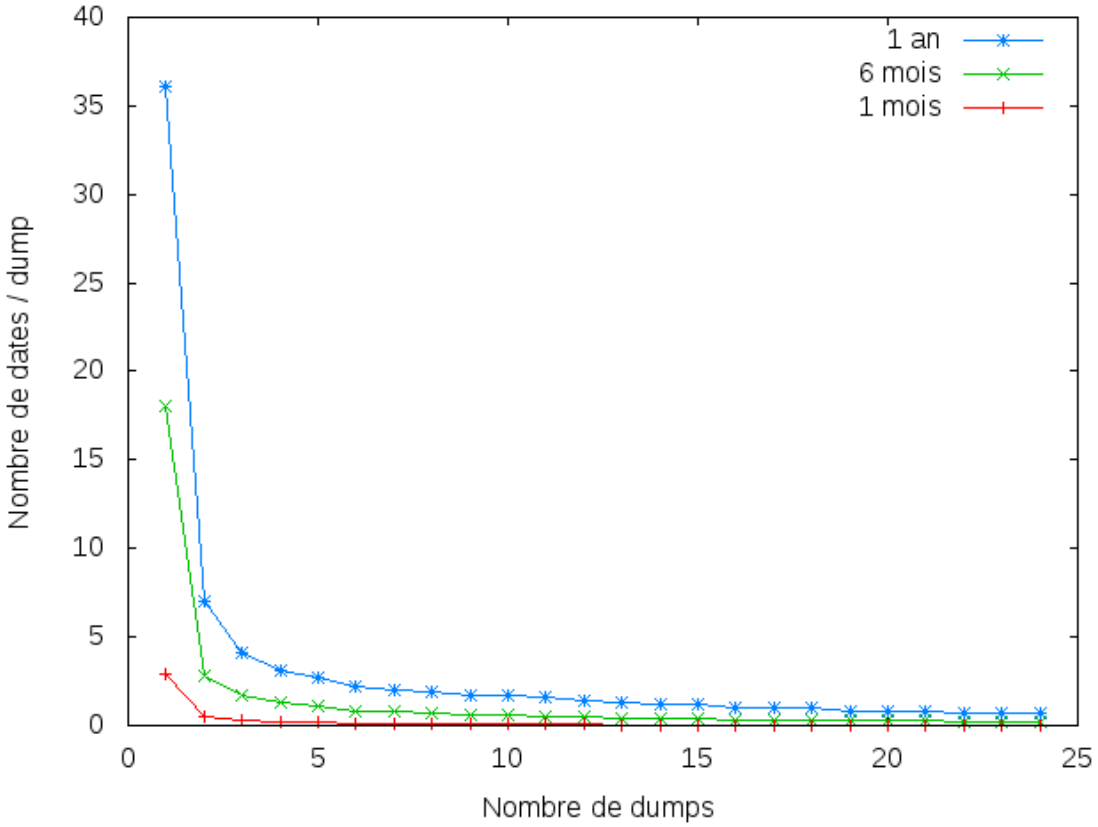


FIGURE 4.8 – Nombre de dates par dump après l'application de l'analyse multi-dump et l'analyse contextuelle sur OV

4.3.5 Scénarios réels

Cette section fournit des résultats sur des scénarios réels, recherchant la date de naissance et la date de voyage dans les dumps Calypso, ou recherchant la date d'achat du ticket pour l'application OV. L'analyse multi-dump et les deux assertions de l'analyse contextuelle sont utilisées dans les expériences suivantes. Les résultats obtenus ne comportent que très peu de faux positifs et sont donc exploitables facilement par un enquêteur.

Calypso : la date de naissance

En recherchant la date de naissance du possesseur dans les dumps et en appliquant toutes les analyses, il reste 189 candidats. Ce nombre de candidats est élevé pour un enquêteur. Mais, pour encoder une date de naissance, il est beaucoup plus probable d'avoir une date de type *format* qu'un *compteur*. Un compteur démarrant en 1970, 1990, ou encore 1997 ne pourrait pas encoder les dates de naissance d'une part importante de la population. En éliminant les fonctions de type *compteur*, il reste seulement 2 candidats. Ces 2 candidats sont vraiment utilisés par l'opérateur pour encoder la date de naissance du possesseur dans le dump. L'analyse n'a donc généré aucun faux positif.

Calypso : la date de voyage

La recherche de la date de voyage dans les dumps Calypso en appliquant l'analyse multi-dump et l'assertion `Assert3` de l'analyse contextuelle génère quelques faux positifs. En autorisant seulement un intervalle de un jour (i.e., seulement le jour de la date du trajet), appliquer la méthode avec les 75 dumps retourne la date de voyage sans faux positif. Avec seulement 10 dumps, il y a 4 dates par dumps qui sont générées en moyenne. Parmi elles, 3 correspondent aux 3 derniers trajets effectués avec la carte, et la dernière est un faux positif. En autorisant un intervalle de 1 mois et en utilisant tous les dumps, moins de 4 dates sont retournées en moyenne par dump, représentant les 3 derniers trajets, et un faux positif.

Autoriser un intervalle de 1 an avec les 75 dump génère en moyenne 50 faux positifs par dump. Par contre, en ajoutant l'assertion `Assert4`, il reste seulement 12 faux positifs. Un enquêteur retrouverait donc la date de trajet sans difficulté.

OV : la date d'achat

En recherchant la date d'achat dans les 24 dumps OV, et en appliquant toutes les analyses, avec un intervalle de 1 an génère 1 candidat. Ce candidat est effectivement utilisé par l'opérateur pour stocker la date d'achat. Donc, aucun faux positif n'est généré. En réduisant le nombre de dumps à seulement 2, l'analyse génère moins de 10 faux positifs par dump en moyenne. En utilisant toujours un intervalle d'un an, et sans exploiter l'assertion `Assert4`, l'analyse génère un faux positif par dump. Un enquêteur retrouverait donc la date d'achat sans difficultés.

Tableau 4.5 – Liste des dates retrouvées après application de la méthode sur des applications de la vie réelle

Application	Champs de type date retrouvé
Calypso	<Date de naissance du possesseur>, <Date de création>, <1-5 dates des derniers voyages>,
Tickets de transport	<Date de validité>, <Date d'achat> , <Date d'utilisation>
Forfaits de ski	<Période de validité>, <Date de la dernière remontée mécanique>
Carte Vitale	<Date de naissance du porteur>, <Date de naissance des enfants>, <Date de début de maladie>
Passeport	<Date de naissance du porteur>, <Date de création>, <Date de validité>
EMV	<1-100 Dates de transaction>, <Date de validité>

4.3.6 Résumé des résultats

Le tableau 4.5 fournit les dates que l'on a récupérées dans les différentes applications des 371 dumps. Les cartes Calypso peuvent contenir des dates à propos de la création, la date de naissance du porteur et les dates des derniers trajets effectués. Le nombre de ces dates varie entre 1 et 5 selon l'opérateur. Les tickets de transport de différentes villes, peuvent stocker la date de validité, d'achat et d'utilisation. Les forfaits de ski stockent généralement la date de validité et de la dernière remontée mécanique utilisée. La carte Vitale enregistre la date de naissance du porteur, les dates de naissance des éventuels enfants, les dates de début d'événements médicaux le cas échéant (e.g., maladies de longue durée). Les passeports biométriques stockent la date de naissance du porteur, la date de création et la période de validité du passeport. Les cartes bancaires (EMV) stockent la date de validité de la carte et les dates des dernières transactions effectuées avec les cartes. Ce nombre pouvant aller jusqu'à plus de 100 transactions selon les opérateurs.

4.4 Conclusion

Dans ce chapitre, une méthode est donc proposée pour retrouver les dates dans les dumps de mémoire des cartes à puce. La méthode proposée est composée de 3 étapes : décodage, analyse multi-dump et analyse contextuelle. La première étape consiste à appliquer de nombreuses fonctions de décodage sur les données binaires afin de générer des dates. Cette étape génère aussi des faux positifs. Ensuite, l'analyse multi-

dump exploite les similarités entre les dumps de la même application pour éliminer une partie des faux positifs. Enfin, l'analyse contextuelle exploite les informations contextuelles qui peuvent être collectées, telles que des informations imprimées sur la carte, afin de supprimer les faux positifs restants. Par exemple, l'analyse contextuelle peut ainsi rechercher des dates dans un intervalle de temps donné. L'élimination des données cryptographiques n'a pas été effectuée dans ce chapitre car les exemples fournis afin d'illustrer la méthode ne contiennent pas de données cryptographiques.

Cette méthode réduit fortement le nombre de faux positifs générés par la phase de décodage. Elle va même jusqu'à tous les éliminer dans certains scénarios réels (e.g., date de naissance ou de trajet dans les cartes Calypso). Elle permet de retrouver des dates dans plusieurs applications différentes telles que des cartes de transport, des forfaits de ski, la carte Vitale, etc.

La méthode proposée peut être adaptée pour retrouver d'autres données telles que des numéros de cartes de crédits, l'âge du porteur, etc. Les différences reposent sur les fonctions de décodage à appliquer, et les informations contextuelles à exploiter.

Conclusion

L'analyse forensique des mémoires EEPROM de cartes à puce a été étudiée dans cette thèse. Après avoir décrit le contexte et les difficultés liées à ces analyses, les dumps mémoires considérés ont été définis ainsi que leur méthode d'extraction. Ensuite, l'état de l'art des différentes techniques d'analyse forensique existantes a été fait, montrant ainsi leurs limites dans le contexte de la thèse. Le manuscrit s'articule autour de trois contributions qui permettent de traiter l'analyse forensique des mémoires EEPPROM des cartes à puce : la séparation des données cryptographiques de celles contenant des informations dans un dump de mémoire, la recherche automatique d'informations textuelles ainsi que la recherche automatique de dates. Chacune de ces contributions est soutenue par des expériences effectuées sur 371 dumps provenant d'applications de la vie réelle. L'analyse de ces résultats met en évidence que l'exploitation des spécificités des cartes à puce à travers l'analyse multi-dump améliore significativement les résultats obtenus.

Contributions

La première contribution sépare les données *cryptographiques* des données *informationnelles* dans un dump. Ainsi, les zones de données où ne pourront pas être retrouvées des informations sont identifiées. Cette méthode utilise une combinaison de tests statistiques dont les paramètres ont été choisis grâce à une technique d'apprentissage automatique : le *boosting*. Appliquée sur des données de la vie réelle (Calypso, EMV, Vitale, passeport électronique, ...), $94,0\%$ des données *cryptographiques* et $98,5\%$ des données *informationnelles* sont reconnues. Appliquer l'analyse multi-dump permet d'obtenir $100,0\%$ de taux de reconnaissance sur ces mêmes applications de la vie réelle.

La deuxième contribution de cette thèse est une méthode pour retrouver automatiquement les informations textuelles stockées dans la mémoire. Pour cela, une méthode en trois étapes est proposée : une analyse lexicale, une analyse syntaxique

puis une analyse multi-dump. La première étape consiste à générer des chaînes de caractères à partir des données du dump. La seconde étape analyse les n-grammes de chaque chaîne de caractères et les compare à ceux d'un corpus d'apprentissage. Les chaînes de caractères jugées ainsi trop éloignées du corpus d'apprentissage sont éliminées. Ensuite, les chaînes de caractères provenant de plusieurs dumps de la même application sont analysées simultanément afin d'améliorer les résultats. La méthode proposée est capable de retrouver plus de 99% des informations textuelles stockées dans les dumps réels. La précision obtenue est de l'ordre de 52%, ce qui signifie que parmi les chaînes de caractères considérées comme étant des informations, moins de la moitié d'entre elles sont des faux positifs. La méthode proposée est ainsi capable de retrouver des noms, prénoms, adresses postales et électroniques dans les applications les plus populaires comme : les cartes bancaires, les cartes de transport, les passeports électroniques, etc.

La troisième contribution de cette thèse est une méthode pour retrouver automatiquement les dates enregistrées dans la mémoire. La première étape consiste à décoder les données du dump afin de générer des dates. Ensuite, la technique exploite les spécificités des mémoires des cartes à puce pour éliminer les nombreux faux positifs. Plusieurs dumps de la même application sont ainsi analysés simultanément. Aussi, les informations contextuelles liées à la carte et son utilisation sont utilisées afin d'appliquer des contraintes supplémentaires sur les dates générées. La méthode proposée retrouve les informations importantes telles que des dates de naissance ou des dates d'utilisation de la carte lors d'un paiement ou d'un transport. Des scénarios sur des applications de la vie réelle illustrant l'efficacité de la méthode proposée sont aussi présentés.

Perspectives

Durant cette thèse, les méthodes proposées ont seulement été testées sur les mémoires EEPROM de cartes à puce. Il semble possible de les appliquer dans de nombreux autres contextes tels que l'analyse de binaire, les traces réseaux, ou encore les applications mobiles. Les outils existants pour effectuer des investigations numériques sur ces type de données utilisent, soit, les spécifications des applications, soit, dans la plupart des cas, les résultats d'une rétro-ingénierie préalablement effectuée couplée à une analyse dynamique de l'application ciblée afin de retrouver les informations. De plus, lorsque les données sont altérées ou incomplètes, ces outils sont souvent inefficaces, alors qu'une part importante des informations est potentiellement présente.

Traces réseaux L'analyse de traces réseaux peut être effectuée dans plusieurs

contextes : diagnostiquer un problème de performance réseau, caractériser le trafic généré par une nouvelle application, détection d'attaques par déni de service, détection d'intrusion, etc. Mais aussi dans le but d'analyser le contenu de trames réseaux capturées lors d'une enquête judiciaire. Dans ce dernier cas, il est possible d'appliquer une partie des méthodes proposées sur des captures réseaux. Lorsque le(s) protocole(s) de communication utilisé(s) au sein de la capture réseau sont connus, il existe des outils tels que Scapy [65] ou Wireshark [66] permettant d'analyser efficacement ces captures réseaux. Par exemple, Wireshark permet de visualiser les différents champs de chaque couche réseau. Dans le cas où le protocole de communication utilisé au sein de la capture réseau est inconnu, il est possible d'effectuer une tentative de rétro-ingénierie à l'aide d'un outil appelé Netzob [67]. Cet outil va analyser le *vocabulaire* du protocole représentant le format des messages échangés, ainsi que sa *grammaire* représentant les séquences de messages valides selon le protocole. Enfin, il va simuler une exécution du protocole afin de vérifier que les informations précédemment inférées sont correctes. Les méthodes proposées vont quant à elles se concentrer sur la charge utile (*payload*) des paquets plutôt que sur le protocole. Cela peut être utile lorsque l'encodage utilisé pour la charge utile est inconnu ou que le protocole est inconnu. Dans le cas où le protocole est connu ou que l'analyse effectuée par Netzob permet de retrouver les informations du protocole, les méthodes proposées peuvent être appliquées seulement sur la charge utile des paquets. Dans le cas contraire, l'intégralité de la capture réseau devra être analysée par la méthode proposée. Appliquées sur la charge utile, les méthodes proposées sont capables de détecter si des informations cryptographiques telles que des clés sont échangées ou si les communications sont chiffrées. Dans le cas d'une communication en claire, la méthode proposée serait capable de retrouver des informations textuelles dans cet échange sans avoir besoin de connaître les spécifications du protocole. Bien qu'applicable, il parait moins évident de tirer profit de la méthode proposée permettant de retrouver des dates. En effet, les charges utiles des paquets sont peu susceptibles de contenir des dates. Quant à l'analyse multi-dump, son principe est applicable pour les traces réseaux. En effet, il est possible de rejouer certaines traces réseaux, ou de stimuler celui-ci afin d'obtenir de nouvelles traces et ainsi obtenir plusieurs traces réseaux de la même application.

Binaires Il est aussi possible d'appliquer une partie des méthodes proposées sur des fichiers binaires. Bien qu'il existe de nombreux outils pour analyser le programme et ses données (e.g., IDA [68], radare [69], etc.), les méthodes proposées peuvent compléter certaines de ces analyses. Par exemple, la méthode distinguant les données *cryptographiques* peut permettre de retrouver des clés stockées dans le binaire. Cette méthode se veut plus générique que les méthodes recherchant spécifiquement des clés

RSA ou AES [37, 38] car la méthode est indépendante du type de clé recherchée. Elle est aussi plus optimisée que la méthode mesurant seulement l'entropie [36] car elle ne se limite pas à cette seule mesure et cherches les meilleurs paramètres pour la taille de la fenêtre, le décalage, ainsi que la manière de combiner les différentes mesures utilisées. La méthode recherchant des informations textuelles peut être capable de retrouver des symboles ou des ressources stockées dans le binaire. Le principe de l'analyse multi-dump pourrait être envisagé dans le cas où l'on analyserait différentes versions d'un binaire, plus particulièrement pour les différentes versions d'une librairie.

Données des applications mobiles Lors d'une analyse forensique d'un téléphone, les données contenant des informations sont reliées à des applications, par exemple, la téléphonie, la messagerie, le courrier électronique, les réseaux sociaux, etc. Les outils disponibles pour effectuer ces analyses traitent au cas par cas (e.g., [28]). Si l'application à analyser fait partie des applications connues de l'outil alors des informations sont retournées, sinon aucune information n'est retournée. Ces informations peuvent être partielles ou complètes, selon l'efficacité de l'outil. Dans le cas où l'application analysée n'est pas connue de l'outil ou bien que l'analyse effectuée par l'outil est incomplète, il est alors intéressant d'appliquer les méthodes proposées dans la thèse.

Dans ce manuscrit, le cas d'un enquêteur passif est considéré. Un enquêteur représente toute personne cherchant à interpréter les données contenues dans les cartes à puce. Dans ces conditions, les dumps à analyser ne sont pas choisis par l'enquêteur. Il est toutefois possible lors de l'analyse d'un ou plusieurs dumps d'une application de récupérer des nouveaux dumps de manière active. Dans ce cas, l'enquêteur maîtrise les informations contextuelles liées aux nouveaux dumps collectés. Il est ainsi possible d'élaborer une stratégie qui optimise le nombre de dumps à collecter pour éliminer les faux positifs de manière efficace. Il peut exploiter le fait que les informations soient stockées au même endroit et avec la même fonction d'encodage. Ainsi, une fois qu'il a retrouvé les informations enregistrées dans un dump où il connaît les informations contextuelles, il peut récupérer ces mêmes informations dans les dumps où il ne connaissait pas la vérité terrain.

D'autres informations que celles recherchées (les noms et les dates) sont enregistrées dans les mémoires EEPROM des cartes à puce. Par exemple, des identifiants de ligne de bus, de station, des codes d'applications, etc. Ces informations semblent plus compliquées à retrouver. Dans le cas d'utilisation d'identifiants, il est peu probable de pouvoir décoder les données avec une technique de décodage exhaustive. Seule l'utilisation d'informations contextuelles détaillées et exactes combinées à une analyse

multi-dump pourrait permettre de retrouver ces identifiants et de les interpréter.

Ainsi, les contributions de cette thèse suivent bien les recommandations de la communauté de l'analyse forensique actuelle en s'intéressant à de nouveaux supports numériques très peu étudiées, à savoir, la mémoire non volatile des cartes à puces. Aussi, les méthodes proposées se veulent génériques, elles sont indépendantes des applications analysées. Ainsi, elles permettent d'assister l'enquêteur face à des données inconnues.

Publications de l'auteur

Conférences internationales avec comité de lecture et avec actes

1. **Gougeon, T.**, Barbier, M., Lacharme, P., Avoine, G., & Rosenberger, C. (2016, June). Memory Carving in Embedded Devices : Separate the Wheat from the Chaff. In International Conference on Applied Cryptography and Network Security (ACNS'16) (pp. 592-608). Guildford, United Kingdom, June 19-22, 2017. Springer International Publishing.
2. **Gougeon, T.**, Barbier, M., Lacharme, P., Avoine, G., & Rosenberger, C. (2017, August). Memory carving can finally unveil your embedded personal data. In International Conference on Availabity, Reliability and Security (ARES'17) (9 pages). Reggio Calabria, Italy, August 29-September 01, 2017. ACM.

Articles en soumissions

1. **Gougeon, T.**, Barbier, M., Lacharme, P., Avoine, G., & Rosenberger, C. Extracting dates from smart cards memory.
2. **Gougeon, T.**, Lacharme, P. How to break CaptchaStar.

Autres publications

1. **Gougeon, T.**, Avoine, G. Investigation numérique dans votre porte-feuille. MISC Magazine. Hors série numéro 15. La sécurité des objets connectés.

Présentations et séminaires

1. Séminaire Dromadaire de l'équipe EMSEC, 14/06/2016, IRISA, Rennes, France.
2. Workshop CRYPTACUS (COST Cryptanalysis of Ubiquitous Computing Systems), 14-15/03/2017, Sutomore, Montenegro.
3. Séminaire Dromadaire de l'équipe EMSEC, 07/07/2017, IRISA, Rennes, France.
4. Séminaire sécurité des systèmes électroniques embarqués, Septembre 2017, IRISA Rennes.

Bibliographie

[1] Joshua I. James and Pavel Gladyshev. Challenges with automation in digital forensic investigations. *arXiv preprint arXiv :1303.4498*, 2013. [cité p. 6]

[2] Alain Pannetrat. Cardpeek. http://pannetrat.com/Cardpeek/, dernière visite : 05/2017. [cité p. 10, 26, 72]

[3] Adam Laurie. Rfidiot. http://rfidiot.org/, dernière visite : 05/2017. [cité p. 11]

[4] NXP. NFC Research Lab. https://play.google.com/store/apps/details?id=at.mroland.android.apps.nfctaginfo&hl=fr, dernière visite : 05/2017. [cité p. 11]

[5] NXP. NFC TagInfo by NXP. https://play.google.com/store/apps/details?id=com.nxp.taginfolite&hl=fr, dernière visite : 05/2017. [cité p. 11]

[6] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling Mifare classic. In *European Symposium on Research in Computer Security (ESORICS)*, pages 97–114, Malaga, Spain, 2008. Springer. [cité p. 14]

[7] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D Garcia. A practical attack on the Mifare classic. In *International Conference on Smart Card Research and Advanced Applications (CARDIS)*, pages 267–282, London, United Kingdom, 2008. Springer. [cité p. 14]

[8] Nethemba Core Team. mfoc. https://github.com/nfc-tools/mfoc, dernière visite : 05/2017. [cité p. 14]

[9] Andrei Costin. mfcuk. https://github.com/nfc-tools/mfcuk, dernière visite : 05/2017. [cité p. 14]

[10] Wolfgang Rankl and Wolfgang Effing. *Smart card handbook*. John Wiley & Sons, 2004. [cité p. 15, 72]

[11] Information Security Group from University College London. emvlab. https://www.emvlab.org/tlvutils/, dernière visite : 05/2017. [cité p. 16]

[12] Service public. Quelles informations contient la carte Vitale ? https://www.service-public.fr/particuliers/vosdroits/F21742, dernière visite : 05/2017. [cité p. 20]

[13] legifrance. Arrêté du 14 mars 2007 relatif aux spécifications physiques et logiques de la carte d'assurance maladie et aux données contenues dans cette carte , 2007. https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000461627&fastPos=1&fastReqId=1733260077&categorieLien=id, dernière visite : 05/2017. [cité p. 20]

[14] Simson L Garfinkel. Digital forensics research : The next 10 years. *digital investigation*, 7 :S64–S73, 2010. [cité p. 22, 23]

[15] Nicole Beebe. Digital forensic research : The good, the bad and the unaddressed. *Advances in digital forensics V*, pages 17–36, 2009. [cité p. 23]

[16] Rainer Poisel and Simon Tjoa. A comprehensive literature review of file carving. In *Availability, Reliability and Security (ARES)*, pages 475–484, Regensburg, Germany, September 2013. IEEE. [cité p. 23]

[17] General public. Foremost. http://foremost.sourceforge.net, dernière visite : 05/2017. [cité p. 23]

[18] Grenier, Christophe. Photorec. http://www.cgsecurity.org/wiki/PhotoRec, dernière visite : 05/2017. [cité p. 23]

[19] Simson L Garfinkel. Digital media triage with bulk data analysis and bulk_extractor. *Computers & Security*, 32 :56–72, 2013. [cité p. 24, 62]

[20] Digital Corpora. Bulk_extractor. http://www.forensicswiki.org/wiki/Bulk_extractor, dernière visite : 05/2017. [cité p. 24, 63]

[21] Volatility Foundation. Volatility, 2013. https://github.com/volatilityfoundation/volatility. [cité p. 25]

[22] Rekall Team. Rekall. https://github.com/google/rekall, dernière visite : 05/2017. [cité p. 25]

[23] Nick L Petroni, Aaron Walters, Timothy Fraser, and William A Arbaugh. Fatkit : A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, 3(4) :197–210, 2006. [cité p. 25]

[24] TRACIP. TRACIP IEF. https://www.tracip.fr/internet-evidence-finder.html, dernière visite : 05/2017. [cité p. 25]

[25] Magnet Forensics. Magnet IEF. https://www.magnetforensics.com/magnet-ief, dernière visite : 05/2017. [cité p. 25]

[26] Aditya Mahajan, MS Dahiya, and HP Sanghvi. Forensic analysis of instant messenger applications on Android devices. *arXiv preprint arXiv :1304.4915*, 2013. [cité p. 25]

[27] Farhood Norouzizadeh Dezfouli, Ali Dehghantanha, Brett Eterovic-Soric, and Kim-Kwang Raymond Choo. Investigating social networking applications on smartphones detecting Facebook, Twitter, Linkedin and Google+ artefacts on Android and iOS platforms. *Australian journal of forensic sciences*, 48(4) :469–488, 2016. [cité p. 25]

[28] Cellebrite. Cellebrite - Mobile Forensics Solutions. http://www.cellebrite.com/Mobile-Forensics/Solutions, dernière visite : 05/2017. [cité p. 25, 106]

[29] SecureView. Susteen. http://www.secureview.us/, dernière visite : 05/2017. [cité p. 25]

[30] Gildas Avoine, Luca Calderoni, Jonathan Delvaux, Dario Maio, and Paolo Palmieri. Passengers information in public transport and privacy : Can anonymous tickets prevent tracking ? *International Journal of Information Management*, 34(5) :682–688, 2014. [cité p. 26, 72]

[31] Jean-Louis Lanet, Guillaume Bouffard, Rokia Lamrani, Ranim Chakra, Afef Mestiri, Mohammed Monsif, and Abdellatif Fandi. Memory forensics of a java card dump. In *Smart Card Research and Advanced Applications (CARDIS)*, pages 3–17. Springer, Paris, France, 2014. [cité p. 26]

[32] Ton Van Deursen, Sjouke Mauw, and Saša Radomirović. mCarve : Carving attributed dump sets. In *USENIX Security Symposium*, pages 107–121, San Francisco, California, USA, August 2011. USENIX Association Berkeley. Tool available at http://satoss.uni.lu/software/ccarve/. [cité p. 26]

[33] Patrick Gueulle. Cloner la carte vitale 2 : enfantin ?, 2011. http://www.acbm.com/inedits/securite-vitale2.html. [cité p. 27]

[34] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for random and pseudorandom number

generators for cryptographic applications. Technical report, DTIC Document, April 2010. [cité p. 36, 38, 40]

[35] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5) :771–780, 1999. [cité p. 36, 46]

[36] Adi Shamir and Nicko Van Someren. Playing "hide and seek" with stored keys. In *Financial cryptography*, pages 118–124, Anguilla, British West Indies, 1999. Springer. [cité p. 37, 38, 58, 106]

[37] Torbjörn Pettersson. Cryptographic key recovery from linux memory dumps, 2007. Chaos Communication Camp (CCC). [cité p. 37, 38, 106]

[38] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember : cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5) :91–98, 2009. [cité p. 38, 106]

[39] Carsten Maartmann-Moe, Steffen E Thorkildsen, and André Årnes. The persistence of memory : Forensic identification and extraction of cryptographic keys. *digital investigation*, 6 :S132–S140, 2009. [cité p. 38]

[40] Fatih Sulak. A new statistical randomness test : Saturation point test. *International Journal of Information Security Science*, 2(3) :81–85, 2013. [cité p. 40, 50]

[41] Ali Doğanaksoy, Cağdaş Çalık, Fatih Sulak, and M Sönmez Turan. New randomness tests using random walk. In *National Cryptology Symposium II*, 2006. [cité p. 40]

[42] Pedro María Alcover, Antonio Guillamón, and María del Carmen Ruiz. A new randomness test for bit sequences. *Informatica*, 24(3) :339–356, 2013. [cité p. 40, 50]

[43] Fatih Sulak, Ali Doğanaksoy, Barış Ege, and Onur Koçak. Evaluation of randomness test results for short sequences. In *Sequences and Their Applications (SETA)*, pages 309–319. Springer, Paris, France, 2010. [cité p. 40]

[44] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization : Papers from the 1998 workshop*, volume 62, pages 98–105. AAAI Press, 1998. [cité p. 45]

[45] Simon Haykin and Neural Network. *Neural Networks : A Comprehensive Foundation*. Prentice Hall PTR, 2004. [cité p. 45, 46]

[46] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3) :131–163, 1997. [cité p. 46]

[47] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *International Conference on Machine Learning (ICML)*, volume 99, pages 124–133, San Francisco, California, USA, 1999. Morgan Kaufmann Publishers Inc. [cité p. 46]

[48] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3) :293–300, 1999. [cité p. 46]

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553) :436–444, 2015. [cité p. 46]

[50] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, Massachusetts, USA, June 2015. IEEE. [cité p. 46]

[51] Yong Liu and Xin Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10) :1399–1404, 1999. [cité p. 46]

[52] Jiang Su and Harry Zhang. A fast decision tree learning algorithm. In *Proceedings of the 21st national conference on Artificial intelligence (AAAI)*, volume 1, pages 500–505, Boston, Massachusetts, USA, July 2006. AAAI Press. [cité p. 47]

[53] Donald Ervin Knuth. *The art of computer programming : Seminumerical Algorithms*, volume 2. Addison-Wesley, 1997. [cité p. 50]

[54] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and Its Interface*, 2(3) :349–360, 2009. [cité p. 53]

[55] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn : Machine learning in python. *The Journal of Machine Learning Research*, 12 :2825–2830, 2011. [cité p. 53]

[56] Ah-Hwee Tan. Text mining : The state of the art and the challenges. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, volume 8, pages 65–70. Springer, 1999. [cité p. 62]

[57] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5) :513–523, 1988. [cité p. 62]

[58] Harry Zhang. The optimality of naive bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 562–568, Miami Beach, Florida, USA, January 2004. AAAI Press. [cité p. 68]

[59] EMVCo. EMV Specification. https://www.emvco.com/specifications.aspx, dernière visite : 05/2017. [cité p. 72]

[60] ICAO. Doc 93003, machine readable travel documents, 2016. http://www.icao.int/publications/Documents/9303_p3_cons_en.pdf. [cité p. 72]

[61] Céline Delforge. La carte mobib mise à nu : le ministre Smet doit s'expliquer, 2009. http://ecolo.be/?article1065. [cité p. 72]

[62] Belgium government. eidReader. http://eid.belgium.be/en/using_your_eid, dernière visite : 05/2017. [cité p. 72]

[63] Openwall Project. John the ripper. http://www.openwall.com/john/, dernière visite : 05/2017. [cité p. 75]

[64] US Census. Frequently occurring surnames from the census 2000, 2000. http://www.census.gov/topics/population/genealogy/data/2000_surnames.html. [cité p. 75]

[65] SecDev. Scapy. http://www.secdev.org/projects/scapy/, dernière visite : 05/2017. [cité p. 105]

[66] Wireshark Foundation. Wireshark. https://www.wireshark.org/, dernière visite : 05/2017. [cité p. 105]

[67] Netzob. Netzob. https://github.com/netzob/netzob, dernière visite : 05/2017. [cité p. 105]

[68] Hex-Rays SA. IDA. https://www.hex-rays.com/products/ida/support/download.shtml, dernière visite : 05/2017. [cité p. 105]

[69] Pancake. Radare2. http://www.radare.org/r/, dernière visite : 05/2017. [cité p. 105]

[70] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013. [cité p. 121]

Glossaire

Calypso Calypso est un standard utilisé pour la billetterie dans le monde entier. 13

Carte à puce Carte comportant un circuit intégré pouvant réaliser des calculs et stocker des informations. 8

Données informationnelles Les données informationnelles sont toutes les données non-cryptographiques d'un dump de mémoire, elles incluent les données personnelles (noms, dates, etc.) et les données techniques (identifiant, compteur, etc.).. 36

Dump de mémoire Contenu de la mémoire EEPROM d'une carte à puce. 7

EEPROM Mémoire morte effaçable électriquement et programmable. 7

EMV Europay Mastercard Visa est le standard international des cartes de paiement. 13

Faux positif Un faux positif apparaît lorsqu'une fonction de décodage est appliquée sur des données ayant été encodées avec une fonction différente. 6

Multi-dump Analyse simultanée de plusieurs dumps de mémoire provenant de la même application. 26

Précision d'un classifieur Proportion d'information (selon la vérité terrain) parmi les chaînes de caractères classées comme des informations par le classifieur. 74

Annexe

Annexe A

Algorithme AdaBoost

Soient (x_1, y_1), ..., (x_m, y_m) avec $x_i \in \mathcal{X}$ et $y_i \in \{-1, +1\}$
Initialisation : $D_1(i) = 1/m$ pour $i = 1, \ldots, m$
pour $t = 1, \ldots, T$ **faire**
Entraîner le classifieur faible en utilisant D_t
Obtenir le classifieur faible $h_t : \mathcal{X} \to \{-1, +1\}$
Objectif : sélectionner h_t avec l'erreur ϵ_t la plus faible

$$\epsilon_t = \mathbf{Pr}_{i \sim D_t}[h_t(x_i) \neq y_i]$$

Choisir $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Mettre à jour les distributions
pour $i = 1, \ldots, m$ **faire**

$$D_{t+1}(i) = \frac{D_t(i)\exp\left(-\alpha_t y_i h_t\left(x_i\right)\right)}{Z_t}$$

avec Z_t le facteur de normalisation (choisi tel que D_{t+1} soit une distribution.
fin pour
fin pour
Retourner l'hypothèse finale

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

FIGURE A.1 – L'algorithme de Boosting de AdaBoost

La figure A.1 fournit le pseudo-code de l'algorithme de AdaBoost qui provient de l'article de R. Schapire [70]. Soient m exemples d'apprentissage dont on connaît la vérité terrain, (x_1, y_1), ..., (x_m, y_m) où les x_i sont définis dans un domaine $\mathcal{X}$,

et les labels représentant la vérité terrain $y_i \in \{-1, +1\}$. À chaque tour de boucle $t = 1, \ldots, T$, une distribution D_t est calculée sur les m exemples d'entraînements. Ensuite, un algorithme d'apprentissage faible est appliqué afin d'obtenir un classifieur faible $h_t : \mathcal{X} \to \{-1, +1\}$, où l'objectif de cet algorithme d'apprentissage faible est d'obtenir un classifieur minimisant l'erreur ϵ_t relative au domaine D_t. Le classifieur final H calcule le signe d'une combinaison pondérée des classifieurs faibles, chaque classifieur étant pondéré par un poids α_t.

Table des figures

Liste des tableaux

Dans notre monde toujours plus connecté, les cartes à puce sont impliquées quotidiennement dans nos activités, que ce soit pour le paiement, le transport, le contrôle d'accès ou encore la santé. Ces cartes contiennent des informations personnelles liées aux faits et gestes de leur possesseur. Le besoin d'interpréter les données contenues dans les mémoires de ces cartes n'a jamais été aussi important. Cependant, sans les spécifications de l'application, il est difficile de connaître quelles informations sont stockées dans la carte, leur emplacement précis, ou encore l'encodage utilisé.

L'objectif de cette thèse est de proposer une méthode qui retrouve les informations stockées dans les mémoires non volatile des cartes à puce. Ces informations peuvent être des dates (e.g., date de naissance, date d'un événement) ou des informations textuelles (e.g., nom, adresse). Pour retrouver ces informations, un décodage exhaustif des données à l'aide de différentes fonctions de décodage est possible. Malheureusement, cette technique génère de nombreux faux positifs. Un faux positif apparaît lorsqu'une fonction de décodage est appliquée sur des données qui ont été encodées avec une fonction différente. Cette thèse s'appuie alors sur trois contributions exploitant les spécificités des cartes à puce pour éliminer ces faux positifs. La première contribution identifie les objets cryptographiques dans les mémoires non volatiles des cartes à puce afin de ne pas effectuer le décodage sur ces données. Les deux autres contributions retrouvent respectivement des informations textuelles et des dates dans ces mémoires. Afin de valider ces méthodes, elles sont chacune appliquées sur 371 mémoires de cartes à puce de la vie réelle.

Memory carving of smart cards memories

In our increasingly connected world, smart cards are involved in any everyday activity, and they gather and record plenty of personal data. The need to interpret the raw data of smart card memory has never been stronger. However, without the knowledge of the specifications, it is difficult to retrieve what are the information stored, their location, and the encoding used to store them.

The objective of this thesis is to propose a method retrieving the stored information in the non-volatile memory of smart cards. This information include dates (e.g., birth date or event date) and textual information (e.g., name, address). In order to retrieve these information, it is possible to perform an exhaustive decoding of the data with several decoding functions. Unfortunately, this technique generates a lot of false positives. Indeed, a false positive occurs when a decoding function is applied to data that have been encoded with another function. This thesis proposes three contributions exploiting smart cards specificities to eliminate the false positives. The first contribution identifies cryptographic material in these non-volatile memories in order to prevent the false positives generated by the decoding of these cryptographic objects. The two others contributions retrieve respectively textual information and dates in these memories. In order to validate these methods, they are applied on 371 memory dumps of real-life smart cards.

Spécialité Informatique et applications

Laboratoire GREYC - UMR CNRS 6072 - Normandie Université - Unicaen - Ensicaen
6 Boulevard du Maréchal Juin - 14050 CAEN CEDEX