Memory Carving in Embedded Devices: Separate the Wheat from the Chaff

Thomas Gougeon[1(✉)], Morgan Barbier[1], Patrick Lacharme[1], Gildas Avoine[2,3], and Christophe Rosenberger[1]

[1] Normandie Univ, ENSICAEN, UNICAEN, CNRS, GREYC, 14000 Caen, France
thomas.gougeon@ensicaen.fr
[2] INSA Rennes, IRISA UMR 6074, Rennes, France
[3] Institut Universitaire de France, Paris, France

Abstract. This paper investigates memory carving techniques for embedded devices. Given that cryptographic material in memory dumps makes carving techniques inefficient, we introduce a methodology to distinguish meaningful information from cryptographic material in small-sized memory dumps. The proposed methodology uses an adaptive boosting technique with statistical tests. Experimented on EMV cards, the methodology recognized 92% of meaningful information and 98 % of cryptographic material.

Keywords: Forensics · Memory carving · Randomness · Embedded devices · Smartcards · Privacy

1 Introduction

Embedded devices usually gather and store personal data about the behaviours of their holders. They are typically low-cost devices including (but not limited to) credit cards, mass transportation passes, electronic passports, keyless entry and start systems, and ski passes. They usually gather and store a lot of personal data, for example an electronic passport contains the identity and the picture of its holder [2], a mass transportation pass may store the last trips of its holder [4], a ski pass may also contain the location of the ski lifts the skier used [21], an EMV card records the last payments done by the customer [8], a car ignition key in recent vehicle contains plenty of information about the car and the behaviour of the driver, including the monthly fuel consumption, the external temperature during the last trip, and the average engine speed. In most cases, the personal data contained in these devices are accessible without requiring any authentication, and can be obtained using, for example, the ISO/IEC 7816 interface or by sniffing a genuine communication between the device and a reader.

Interpreting the meaning of the captured raw data is hard when the system specifications are not available. However, such a task is important today when investigations must be carried out. It can be to find digital evidence for example

DOI: 10.1007/978-3-319-39555-5_32

in connection with criminal investigations – when information related to a suspect is stored in a device – or to verify that a system complies with the national privacy regulations.

A large body of literature exists in the field of memory forensics. Many off-the-shelf tools exist, too. The analyses typically focus on hard drives [18] and volatile memories [3]. Analyses of hard drives are typically based on file carving, i.e. a technique that consists in searching for files in the considered data. The main difficulty is the file fragmentation in the system. File carving is consequently performed using machine learning techniques, the entropy of the blocks, or the file headers and footers. The technique targets specific file formats, e.g., PDF, ZIP [5], or file systems such as NTFS [26]. Analyses of volatile memories consist in searching for special strings or signatures, interpreting internal kernel structures, or enumerating and correlating all page frames, in order to retrieve running and terminated processes, open ports, sockets, hidden data, etc.

The analysis of the non-volatile memory of an embedded device differs from classical memory forensics techniques for several reasons. (i) First of all, the memory typically consists of a few kilobits only. (ii) The data available in these devices are poorly structured: in most cases, there are no file headers, sentinels, or field separators. (iii) Home-made encoding systems are commonly used in practice to save memory or to naively hide information. (iv) Performing a bit-by-bit copy of the memory is rarely possible because the only way to access the memory is to use the application program interface (API) or to eavesdrop a genuine communication. This means that the captured data is not necessarily a perfect copy of the memory.

A naive technique to interpret data retrieved from embedded devices (called a dump) consists in applying several encoding functions to the dumps until retrieving the correct one for each information stored. Due to the nature of the dumps, there is unfortunately no oracle that can efficiently determine whether the decoding of the information is correct. As a consequence, the technique outputs many false positives that renders it unusable in practice. Most existing contributions on the memory carving problem for embedded devices consider ad-hoc, hand-made analyses, e.g., for retrieving keys hidden in an EEPROM [6].

There exist few techniques designed for an automatic analysis of embedded devices. A seminal work, though, is due to Ton Van Deursen et al. [25], who investigated the memory carving problem for sets of memory dumps, and applied it to public transportation cards. It is worth noting that they obtained the memory dumps using the API of the cards, meaning that there is no guarantee that the dumps are indeed bit-by-bit copy of the memory. The authors aimed to singulate the memory data fields using the concept of commonalities and dissimilarities applied to a dump set. A commonality occurs for a given bit position if the value of the bit is the same for all the dumps of a given set, whereas a dissimilarity occurs otherwise. Using these commonalities and dissimilarities, as well as contextual information (as data printed on the coupon), the technique deduces the data fields. Once the data fields are singulated, a manual investigation is needed to retrieve the encoding function. The authors applied their technique to the

E-Go System (the public transportation card in Luxembourg) and retrieved a dozen of fields, e.g., the date and time of the last validation. Their work does not provide an automatic interpretation of the data and it requires contextual information to complete the analysis. Another work related to ours is due to Jean-Louis Lanet et al. [14], who investigated the reverse engineering of EEP-ROM in Java Cards. They aim to retrieve the location of the source code and data related to the language(package, class, instance . . .). The index of coincidence [10] is used to locate the source code. This approach is not very efficient, though. Still worse, in our case, real-life dumps are generally generated using several (unknown) encoding functions. This makes the calculation of the index of coincidence meaningless. To retrieve the data related to the language [14] uses a pattern matching technique applied to the headers (or metadata), which differ for each type of data. Unfortunately, there is neither header nor metadata in our dumps.

Given the difficulty to retrieve personal data from the memory dump of an embedded device, this work focuses on a narrower problem that consists in distinguishing meaningful information (encoded with ASCII, BCD, etc.) from cryptographic material (ciphered data, hash value, secret key, etc.). The rationale behind this restriction is that cryptographic materials generate many false positives and no personal information can be obtained from these values, assuming the algorithms used to create the materials are cryptographically secure. As a consequence, we introduce a technique that *separates the wheat from the chaff*, namely a preliminary step in the forensics process that distinguishes meaningful information from cryptographic materials, considered as random data. Unfortunately, the size of the considered dumps does not allow to naively use classical tools (e.g., NIST's statistical tests [19]) that usually require several kilobytes of data to make the statistical tests relevant. Moreover, the tests cannot be directly applied to the data because the considered dumps contain data fields, which must be analysed separately. For the same reason, techniques for locating cryptographic keys hidden in gigabytes of sparse data, proposed by [20] and based on the entropy computation, are not possible on such dumps.

This paper introduces a statistical and automatic recognition technique that distinguishes meaningful information from cryptographic material, obtained from non-volatile memory dumps of embedded devices. The technique, based on a machine learning method, called boosting [9], requires information neither on the dump structure, nor on the application context, for the classification between these two sets of data. The technique is then improved by comparing dumps of different devices belonging to the same application. Our technique reaches quite a high success rate: we applied it to EMV-based dumps and Calypso-based dumps, obtaining a 99 % success rate.

2 Dump Examples

To illustrate the problem considered in this paper, Sect. 2 provides details on two dumps extracted from EMV and Calypso cards. The cards contain elementary files that have been retrieved using the cards' APIs. The files are made of

records. Files can be *linear fixed* (linear data structure of fixed length), *linear variable* (linear data structure of variable length), or *cyclic* (oldest data are erased to store newest data). The information is contained in (non-necessarily contiguous) **fields**, e.g., holder's name, holder's zip code, a cryptographic key, etc. A pedestrian approach has been used to analyse the dumps, given that there does not exist automatic tools that can achieve this task.

2.1 EMV Dump

Figure 1 is a (partial) anonymised dump of a credit card compliant with the EMV specifications [8]. Each numbered line represents a record. The underlined sequences are fields that contain the holder name, the issuer's public key modulus, the amount, and the date of the last transactions.

```
 1.  9F3602004D
 2.  9F13020046
 3.  9F170103
 4.  9F4F109F02069F27019F1A025F2A029A039C01
 5.  70615F201A4A4F484E2F534D4954482E4D5220202020202020202020202020205F300202018C1B9F02069F03069
     F1A0295055F2A029A039C019F37049F45029F4C088D1A8A029F02069F03069F1A0295055F2A029A039C019F37
     049F4C08
 6.  9F49039F3704701A5F25030911015F24031003315A0849750000075922345F340100
 7.  70369F0702FF008E0E00000000000000000020301031F009F0D059800B420009F0E050050480009F0F05B820B
     4F8005F280202509F4A0182
 8.  70329F080200028F01069F320103922434592451B87DA8C05BA7F1DE5DC802BF59D394D6CC034A046F46995E0
     245E437AED7B899
 9.  000000001350400250097810032600
10.  000000001770400250097810032600
11.  000000002090400250097810032500
12.  000000007707400250097810032400
```

Fig. 1. Extract of an anonymised credit card dump.

Holder Name. The underlined sequence of the 5th record represents the name of the holder of the credit card (*MR John Smith*) encoded using ASCII and padded with the repeated pattern 0x20.

Issuer's Public Key Modulus. The underlined sequence of the 8th record represents the issuer public key modulus used by the authentication protocol.

Transactions. Records 9 to 12 represent the last four transactions (cash withdrawal) made by the card. The first underlined sequence represents the transaction amount (13.50 euros for the 9th record) and the second one is the date of the transaction (2010/03/26 for the 9th record).

The EMV card contains a cyclic file that stores information on the transactions. For any new transaction, the information in the cyclic file is rotated such that the record about the oldest transaction is discarded to save room for the newest transaction.

2.2 Calypso Dump

Figure 2 is a (partial) anonymised dump of a transportation card compliant with the Calypso specifications [4]. The record names (ICC, Holder1, Holder2, etc.) are available in the specifications, but the content of the records is not defined by Calypso. The content is indeed let to the discretion of the public transportation operator. The provided example illustrates that a single card may contain several encoding rules, and the information in the card is not necessarily byte-aligned.

```
ICC     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 13 43 00

Holder1 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
Holder2 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

EnvHol1 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
EnvHol2 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

EvLog1  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
EvLog2  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
EvLog3  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

ConList 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
```

Fig. 2. Anonymised transportation car dump.

On the Holder1 line, the first underlined sequence represents the BCD-encoded birth date of the holder: 1975/08/10. The second underlined sequence that continues on the Holder2 line represents the name of the holder, namely "James Smith". To decode this information the binary representation of the sequence must be split into 5-bit pieces (omitting the first bit of the sequence), which are then decoded with the rule (decimal representation): A=1, B=2, C=3, etc. This information is not byte-aligned.

EvLog1, EvLog2, and EvLog3 are the last three trips performed by the card, stored in a cyclic file. For example, the first underlined sequences in the EvLog lines correspond to the validation time, which is "11:53 am" for EvlLog1. This information is retrieved using the binary representation of 0x592 (omitting the last bit), and converting it to an integer that represents the number of minutes since the beginning of the day. The second underlined sequence in each log represents the validation date of the card during the trip: 2008/12/09, for EvlLog1. This information is retrieved by using the binary representation of 0x5108 (omitting the two first bits) and by converting it to an integer that represents the number of days since 1997/01/01. Other information on this line are the transportation means (metro, bus, tramway), the bus line number, the number of travellers who shared the card for that trip, the station, etc. Additional information can be found in the dump, e.g., the serial number of the card, the manufacturer, the date of manufacture, etc.

3 Statistical Analysis

Retrieving the meaningful information from a dump using a statistical analysis is a difficult problem. In particular, the meaningful information is drowned in a mass of information that include pseudo-random values generated by cryptographic means. This paper consequently focuses on a preliminary step in the forensics process that distinguishes meaningful information from cryptographic materials. To start with, we explain below the difficulty to use statistical tests to perform this task in our framework.

3.1 Statistical Tests for (Pseudo-) Random Generators

There exist many statistical tests for random and pseudo-random number generators. The NIST statistical test suite [19] includes the most important ones, while keeping small the redundancy between them. We consequently decided to consider this suite for our experiments.

A statistical test aims to verify a given *null hypothesis*, which is *data are random* in our experiments. A p-value represents the strength of the evidence against the null hypothesis. This p-value is computed from the reference distribution of the tested statistical property. NIST uses an asymptotic distribution.

The hypothesis is rejected if the p-value is lower than the level of significance of the test (for example 0.01 or 0.001). Thus, a threshold of 0.01 means that one sequence among 100 sequences is expected to be rejected. A p-value greater than this threshold (respectively lower) indicates that the sequence is considered to be random (respectively non-random) with a 99 % confidence.

The NIST proposes two methods to decide whether or not a generator is suitable for a cryptographic use. A set of sequences is produced by the generator, and its quality is evaluated by means of statistical tests. The result is determined from the rate of sequences that successfully pass each test (p-value greater than the level of significance), or from the uniformity of the p-values.

Even if tests like the *monobit test*, the *longest runs test*, or the *approximate entropy test* could be theoretically applied to short sequences (100 bits), the recommended length is 20, 000-bit long according to the NIST, because asymptotic approximations are used to determine the limiting distribution. Additional information on these statistical tests can be found in [19].

Moreover, some tests like the *linear complexity test* or the *random excursions test* require at least 10^6 bits to be applied. For short sequences, the NIST suggests that asymptotic distribution would be inappropriate and would need to be replaced by exact distributions that, according to them, would commonly be difficult to compute. Thus, [1,7,23] introduce new tests with their exact distribution, and [24] suggests a new method to take the decision of randomness for short sequences. Unfortunately, although these approaches can deal with short sequence, they require a significantly large set of such sequences.

3.2 Statistical Tests in Our Context

Dumps obtained from embedded devices typically contain information fields whose lengths are between 1 bit and $1,024$ bits (the size of an entire dump is typically 100-bit to $40,000$-bit long).

Each sequence tested can be seen as an output of a different generator (name, date, ciphered or hashed data, etc.), then for a dump, only one sequence per generator can be tested. Section 3.1 and the above-mentioned arguments justify that most of statistical tests are not suited to short sequences, and the technique used by the NIST to decide whether or not a sequence is random is therefore not applicable. Moreover, there is no technique that use a combination of statistical tests to take the decision of randomness. In our context, the decision of the classification of each bit into meaningful information or cryptographic material is only done by directly comparing a p-value to a threshold, but this threshold need to be determined.

4 Distinguish Cryptographic Materials from Meaningful Information

A first step to distinguish meaningful information from cryptographic materials in a memory dump of an embedded device consists in establishing a methodology to apply the statistical tests. Applying the tests to the entire dump is inefficient. Instead, tests should be applied to each field of the considered dumps. Unfortunately, neither the location nor the size of the fields of the dump are known. The methodology consequently consists in classifying the data (*meaningful* or *cryptographic*) bit by bit, instead of field by field.

A second step consists in performing a learning phase where the methodology is applied to dumps for which the classification of bits is known. This ground truth allows us to determine the decision threshold: the statistical tests provide a score to each bit of the considered dump, and the score is compared to the threshold to decide whether a bit is classified *meaningful* or *cryptographic*.

Then, a boosting algorithm [12] is used, namely a machine learning approach that identifies the most appropriate statistical tests to be applied and how to combine their results.

The identified tests can then be applied to dumps whose ground truth is unknown.

Finally, comparing the classification obtained by this combination of statistical tests on different dumps of the same application, we propose a technique that improves the classification for each dump of the application.

4.1 Applying Statistical Tests to Dumps

Given dumps cannot be split into fields, the classification of the data has to be done bit by bit. However statistical tests are not applicable to single bits. As a consequence, bits need to be grouped into sequences. A methodology that separates a dump into several overlapping sequences is thus proposed.

Let D be a n-bit dump (bits indexed from 1 to n), a *sequence length* ℓ, and a *shift* s, with $0 < s \leq \ell \leq n$. The *shift* represents the distance between two start bits of two consecutive tested sequences. The $i+1$-th tested sequence of D is from the bit index $(i \times s + 1)$ to $i \times s + \ell$ with $0 \leq i \leq \lfloor \frac{n-\ell}{s} \rfloor$. In the case where s does not divide $n - \ell$, the bits from $\lceil \frac{n-\ell}{s} \rceil \times s + l - s + 1$ to n are never tested, and so a last sequence from index $n - \ell + 1$ to n is tested. For each tested sequence, the *statistical test* returns a p-value. Due to the use of a *shift* s potentially shorter than the *sequence length* ℓ, each bit of D is tested in at most $\lceil \frac{\ell}{s} \rceil$ different sequences. All bits of D are therefore related to a variable number of p-values. However, the classification method used in Sect. 4.2 works with the same number of scores per bit. A *score function* that takes p-values as input and outputs a single score is therefore applied to the p-values of each bit. The function can be for example the mean of the p-values.

As a consequence, the previous parameters, namely *sequence length*, *shift*, and *score function* play an important role in the quality on the classifier. Furthermore, some statistical tests require an *internal parameter*. For example, the *serial test* looking at the proportion of each possible block of m bits in the tested sequence, takes m as additional input. It leads to a set of N features, $\mathbf{F} = \{F_j,\ 1 \leq j \leq N\}$ where each feature is defined by a 5-tuple (*statistical test*, *sequence length*, *shift*, *score function*, *internal parameter*). Applying a feature F_j to D outputs a score set $S_j = \{s_i^j,\ 1 \leq i \leq n\}$ – as presented in Fig. 3 – where s_i^j is the score assigned by F_j to the i-th bit of D. Applying all the features F_j of $\mathbf{F}$ so generates a set $\mathbf{S} = \{s_i^j,\ 1 \leq j \leq N,\ 1 \leq i \leq n\}$ as presented in Fig. 4.

4.2 Bits Classification Using Statistical Tests

In order to decide whether a bit in a dump should be classified as *cryptographic* or *meaningful*, scores returned by each feature need to be compared to a threshold. This process that determines the class of each bit using the scores and a pre-definite threshold is a classifier. Given a n-bit dump D and its score set S_j obtained by applying a feature F_j, and the predetermined threshold t, the classifier C_j computes the prediction $P_j = \{p_i^j \in \{\textit{cryptographic}, \textit{meaningful}\},\ 1 \leq i \leq n\}$ for D where the prediction of the class of the i-th bit of D is done as following:

$$p_i^j = \begin{cases} \textit{cryptographic}, & \text{if } s_i^j > t \\ \textit{meaningful}, & \text{otherwise} \end{cases}$$

Using scores returned by each F_j together with the ground truth of D, represented by $G = \{g_i \in \{\textit{cryptographic}, \textit{meaningful}\},\ 1 \leq i \leq n\}$, a learning process determines the best threshold to use. We use the learning process described in [22] whose complexity is $\mathcal{O}(n)$ in our case. The best threshold is the one that leads to the classification which is the most similar to the ground truth. Namely, the classification that maximises the recognition rate R_j, where R_j is computed as following:

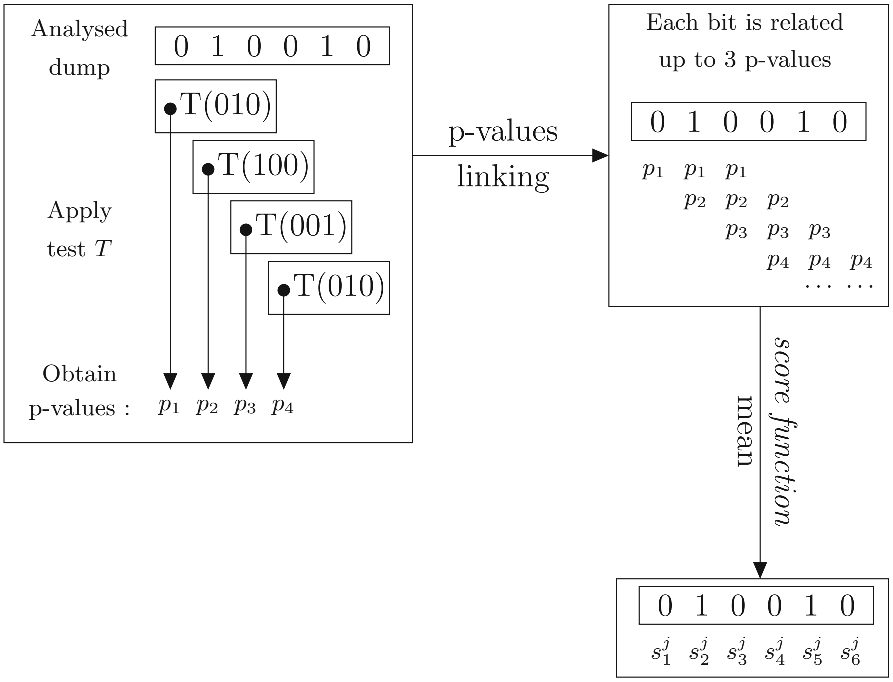
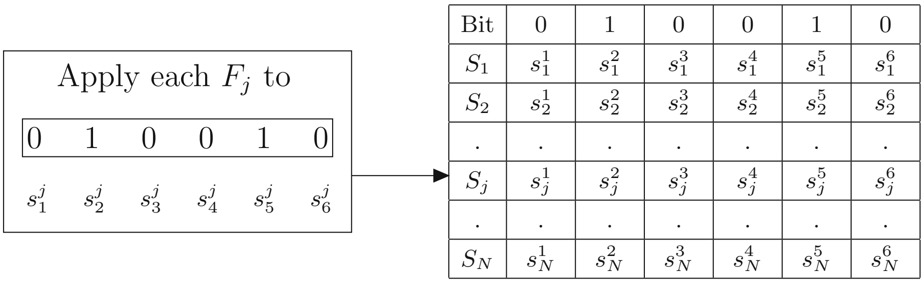


Fig. 3. Applying a feature F_j = (T, 3, 1, mean, .) on a 6-bit dump D. The process assigns the score set $S_j = \{s_i^j,\ 1 \leq i \leq 6\}$ to D.



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_N	s_N^1	s_N^2	s_N^3	s_N^4	s_N^5	s_N^6

Fig. 4. Set $\mathbf{S} = \{s_i^j,\ 1 \leq j \leq N\}$ of scores obtained after applying all features F_j of the set $\mathbf{F}$ to a short dump D of 6 bits length.

$$R_j = \frac{\sum_i^n r_i^j}{n} \text{ with } r_i^j = \begin{cases} 1, & \text{if } G_i = p_i^j \\ 0, & \text{otherwise} \end{cases}$$

Applying the learning process to all F_j leads to a set of classifier $\mathbf{C} = \{C_j,\ 1 \leq j \leq N\}$, where each C_j is related to the feature F_j.

4.3 Boosting of Statistical Tests

In order to determine the best features F_j to use, we propose to use the boosting technique that is a machine learning technique to combine several classifiers of the set $\mathbf{C}$ into a final classifier $\widehat{C}$.

More precisely, we propose here to use the AdaBoost algorithm [9] which is the most popular boosting method. Given a dump, AdaBoost first selects the best classifier of $\mathbf{C}$ and adds it to the final classifier $\widehat{C}$. The best classifier of the set $\mathbf{C}$ is the one that leads to the best recognition rate respecting the ground truth of the dump. Then looking at the obtained classification, misclassified bits by $\widehat{C}$ receive a more important weight. The boosting then selects the new best classifier taking into account the bit weights (thus this second classifier focuses on misclassified bits) and adds it to $\widehat{C}$. The weights of the bits misclassified by $\widehat{C}$ are updated. Repeating those actions until all the bits are correctly classified, or a certain preset number of classifiers in $\widehat{C}$ is reached. AdaBoost does not return the optimal classifier $\widehat{C}$ because it behaves as a greedy algorithm but it is efficient whereas the naive optimal algorithm is computationally infeasible. AdaBoost's complexity is $\mathcal{O}(|\mathbf{C}|L)$, where L is the complexity of the learning algorithm.

The final classifier provided by the boosting is then used to distinguish *cryptographic material* from *meaningful information* in dumps where the ground truth is unknown.

4.4 Merging Classifications in a Set of Dumps

Dumps belonging to the same application share similarities on their data. However, the prediction $\widehat{P}$ obtained by the final classifier is done independently for each dump. Therefore a merging technique is proposed to combine their classification in order to improve the recognition rate.

A set of dumps of the same application can be obtained by dumping memory of different cards belonging to several holders. It can also be acquired by dumping the same card at different time of the card lifetime, e.g. before and after a cash withdrawal for bank cards.

One may expect that dumps belonging to a given application to be identically structured, i.e. containing the same fields, in the same locations. For example, in some Calypso dumps, the name, the birth date, and the postal code of the holder or details of his last trips are always located at the same place in the dump, with the same encoding. The data of these fields vary for each dump but the class (meaningful information or cryptographic material) is the same. The merging process is also applied to cyclic records, because they contain the same fields in the same locations. Therefore a classification of the bits of the application is computed rather than a classification for each dump.

Given a set $\mathbf{D} = \{D_k,\ 1 \leq k \leq d\}$ of d dumps of length n bits belonging to the same application the merging process creates a prediction $P_{merging}$ that replaces all the prediction of the final classifier. The prediction $P_{merging}$ is computed as follows:

$$P_{merging} = \{Majority(\widehat{P}_i^1, \widehat{P}_i^2, \dots \widehat{P}_i^d),\ 1 \leq i \leq n\}$$

where $\widehat{P}_i^k$ represents the prediction of the final classifier for the i-th bit of the dump D_k, and *Majority* represents the application of a majority vote to the classes. The $P_{merging}$ obtained has a better recognition rate than the prediction obtained independently for each dump of $\mathbf{D}$.

5 Experiments

We present the data and the values of the features used by our boosting experiment. The final classifier obtained by AdaBoost is applied to real EMV dumps. The merging process finally allows us to reach a recognition rate greater than 99 % on these EMV dumps.

5.1 Generating Data for the Learning Phase

When fitting the classifiers, AdaBoost requires a sufficient amount of data belonging to each class (i.e. *meaningful information* and *cryptographic material*) from different embedded objects, to be representative of all the existing embedded devices.

We have extracted the data of about 300 devices (using CardPeek [16] or RFIDIOT [15]) from various applications: access control, transportation, banking, health insurance and loyalty cards, train tickets, e-passports, ski passes, etc. Unfortunately, only a small part of these data can be used for the learning phase, because the ground truth of a large part of these data is unknown. In order to solve this problem, we set up a large synthetic dump containing data similar to real dumps, inspired by our 300 dumps. This synthetic dump is modeled by several sequences of variable lengths, containing cryptographic materials (hashed or ciphered data, cryptographic keys, etc.) or meaningful information (dates, names, etc.).

A generated synthetic dump is 100, 000-bit long. It contains approximately 65 % of meaningful information, where sequences are between 80 and 300-bit long. Cryptographic materials are generated from various cryptographic algorithms as RSA, AES, SHA-1, etc. They are truncated to obtain the expected length. Meaningful information includes dates with different encoding, e.g., ASCII, BCD, and various formats like YYYY-MM-DD, YY/MM/DD, etc. There are also names, textual information, and zip codes with various encoding techniques.

5.2 Considered Features

Given each feature is described by 5 parameters (*Statistical test*, *Sequence length*, *Shift*, *Score function*, *Internal parameter*) and each parameter can be assigned with various values, the set $\mathbf{F}$ contains approximately 10, 000 features.

Statistical Test: It is the statistical test that is applied to the dump sequences. All these tests are the NIST tests except those that require 10^6 bits or a specific pattern to test. These NIST tests represent 8 tests: *monobit*, *runs*, *block frequency*, *serial*, *discrete Fourier transform*, *approximate entropy*, *cumulative sum*, and *longest runs*. These 8 tests provided in the NIST suite are completed by the *autocorrelation* [13] and tests suited to short sequences: *TBT* [1] and *saturation point* [23]. There is so 11 tests in total.

Sequence Length: We have decided to lower bound the sequences to 32 bits because tests on too short sequences are not relevant. Sequence lengths used in our experiments are thus chosen in the set $\{32, 48, 76, 100, 128, 192, 256\}$.

Shift: 10 different shifts are used, represented by a percentage of the sequence length: $10\,\%, 20\,\% \ldots 100\,\%$.

Score Function: It represents the method that computes the score from the set of p-values of each bit. The functions mean, min, max, and the geometric mean are used here.

Internal Parameter: When the test requires an internal parameter, several values are considered for this parameter. For example, for the serial test the parameter is the block length. When it is applied to a sequence of 256 bits, the possible size of the blocks are from 2 to 6 bits.

5.3 Learning with AdaBoost

The boosting algorithm must be set with a parameter that is the number of classifiers of the final classifier. This parameter must be well suited to the context to avoid overfitting the final classifier. This phenomenon occurs when the final classifier is too adapted to the data used for fitting, which leads to a poor recognition rate on other data. We have experimented with a number of classifiers from 1 to 50.

Two synthetic dumps are generated, one represents the learning set and the second the validation one. The boosting creates a final classifier $\widehat{C}$ from the learning dump and then this classifier is applied to the validation one. Then, varying the number of classifiers in $\widehat{C}$ on the learning dump, each obtained $\widehat{C}$ is applied to the validation one. The $\widehat{C}$ which leads to the best recognition rate on the validation dump is saved.

The final classifier obtained after learning with AdaBoost is composed of five tests with their parameters, described as following:

- Approximate Entropy on sequences of 192 bits by blocks of 2 bits and a *shift* of 19 bits, the *score function* is *max*.
- Longest runs on sequences of 256 bits by blocks of 8 bits and a *shift* of 25 bits, the *score function* is *min*.

- TBT on sequences of 256 bits by blocks of 4 bits and a *shift* of 76 bits, the *score function* is *mean.*
- Serial test on sequences of 256 bits by blocks of 3 bits and a *shift* of 76 bits, the *score function* is *mean.*
- Cumulative sum on sequences of 256 bits with a *shift* of 230 bits, the *score function* is *max.*

The recognition rate of these tests is 91.7 % on the learning dump and 90.7 % on the validation one.

The boosting algorithm selects the most pertinent statistical tests in relation to our context of short sequences belonging to memory dumps. Note that, slightly varying the learning data, the boosting algorithm returns other strong classifiers (with different statistical tests and parameters) providing similar recognition rate. Namely, generating three dumps D_1, D_2, and D_3, then applying the boosting to the scores obtained by the features on D_1 and D_2 creates two final classifier C_{F_1} and C_{F_2}. These two classifiers can consist in different statistical tests but when they are applied to the dump D_3 they provide similar recognition rate. Using more statistical tests in the final classifier improves the recognition rate on the learning dump, we obtain 98.1 % with 50 statistical tests, but the recognition rate on the validation one is 90.0 %, it is a case of overfitting. Using only one statistical test in the final classifier leads to a recognition rate of 89.9 % on the learning dump and a recognition rate of 87.7 % on the validation one. One can notice that these statistical tests use more than 200 bits to take their decision, but they are able to detect the class of sequences that are shorter than 200 bits, because all bits are tested several times due to the shift between tested sequences.

Experiments have been done with our own python program using the AdaBoost-SAMME.R algorithm [11] from Scikit-learn [17]. Calculating the p-values array on a large dump of $100,000$ bits for our set of $10,000$ features took several hours. Calculations have been made on a 64-core processor (4 AMD Opteron 6282SE 2.6 GHz) with 512 GB of RAM available. Running the AdaBoost algorithm, on a single core, takes between a few minutes (when $|\widehat{C}| = 1$) and two hours (when $|\widehat{C}| = 50$).

5.4 Recognition on Real Dumps

In this subsection, the classifier trained on synthetic data is used to classify meaningful information and cryptographic materials on real dumps of memory. This set is applied to more than 30 EMV dumps [8], 2 VITALE dumps (the French health insurance card) and 7 Calypso dumps. In these cards, the meaning of an important part of the data is publicly known (EMV, VITALE) or a previous work of the authors allows to determine it, so, the ground truth (i.e. theoretical classification of the data) is easily accomplished. It represents more than $600,000$ bits of data with $140,000$ cryptographic bits and $500,000$ bits of meaningful information. As result, we obtained a recognition rate of 92.1% for cryptographic bits and 98.6% for bits of meaningful information. When the final classifier is applied to Calypso cards, we get 100 % of recognition. Note that there are only bits of *meaningful information* in our Calypso dumps. Applying the final classifier to these dumps, is almost instantaneous, for large dumps (40, 000 bits), it takes less than 2 s.

Some further analysis can improve the results, for example if a single cryptographic bit (resp. meaningful bit) is surrounded by a significant amount of meaningful bits (resp. cryptographic bits), then this bit is likely misclassified. Errors are often localised on the frontier between two fields, one containing cryptographic material and another one containing meaningful information.

Table 1. Detection of cryptographic bits and meaningful bits in EMV, VITALE, and Calypso dumps

Dump type	Cryptographic bits	Recognition rate	Meaningful bits	Recognition rate
EMV	131, 384	92.3 %	379, 352	98.0 %
VITALE	9, 168	90.0 %	126, 160	99.9 %
Calypso	0	–	9, 681	100.0 %

5.5 Merging Process on EMV Cards

Table 1 shows that our method applied to a single dump already provides good results. We now still improve the results by analysing in parallel several dumps obtained from the same application. We call this improvement the *merging process.*

In the following experiments, 10 fields representing in total 3, 560 bits are selected, split as 3, 312 cryptographic bits and 248 bits of meaningful information. These fields are information about the holder, the card or cryptographic materials. Since they are repeated numerous time in each dump (cyclic records) and our database is composed of 34 EMV dumps, the merging process takes the decision of the classification of the fields using 21, 120 bits of meaningful information and 124, 512 cryptographic bits. Applying the merging process to all these fields, we obtain a 100 % recognition rate. Merging 3 to 5 dumps is usually enough to reach a 100 % recognition rate.

Table 2 provides the results of the merging process for each selected field from EMV cards, where the class is M for meaningful information and C for cryptographic material. The classic rate column is the mean of recognition rates of the analysis applied to each dump separately. The merging rate column represents the recognition rate when applying the merging process.

Note that the memory structure of the dumps of a given application is not always the same in practice: some records are possibly missing, or are not of the same length, the number of repetitions of cyclic records can vary or the data stored in a field is not always of the same length and the value of the

Table 2. Recognition rate of the merging process on several fields of EMV cards.

Field name	Class	Length (bits)	Classic rate	Merging rate
Issuer PK Certificate	C	1,024	93.1 %	100.0 %
Signed Static App. Data	C	960	93.8 %	100.0 %
ICC PK Certificate	C	1,024	93.1 %	100.0 %
ICC PK Remainder	C	144	83.2 %	100.0 %
Issuer PK Remainder	C	160	86.2 %	100.0 %
App. Label	M	16	99.0 %	100.0 %
App. Preferred Name	M	16	97.3 %	100.0 %
App. Effective Date	M	24	97.6 %	100.0 %
App. Expiration Date	M	24	99.5 %	100.0 %

non-used bits of the allocated space for the field is uncertain. For example, Mifare cards own always the same structure in their memory which is separated into several sectors of fixed length. Whereas EMV cards are made of files that contain records, and depending on the bank, these files and records can differ. They do not store necessarily the same number of transactions, do not contain all possible records of the EMV specification, the field of the name is padded with 0x20 when the name is shorter than the allocated space, etc. Consequently, a pre-processing phase is needed to identify the records in each dump of the application. In our dumps, all records are separated due to the data recovery technique. This pre-process aims to match the records between the dumps. This operation does not require the knowledge of the card specification, because it is performed by analysing the structure of the data of the dump and the data into the record. It includes the size and the location of the records in the dump, combined with the presence of runs of 0 or 1 separating fields in the record.

6 Conclusion and Perspectives

This paper investigates memory carving techniques for embedded devices. Given that cryptographic material in memory dumps makes carving techniques inefficient, we introduce a methodology to distinguish meaningful information from cryptographic material in memory dumps. We propose a technique to apply statistical tests to memory dump from embedded devices. Our approach uses an adaptive boosting algorithm based on results of statistical tests for randomness. We obtained a recognition rate of about 95 % on real dumps from EMV, Vitale and Calypso cards. We also suggested to analyse several dumps in parallel, which increases the recognition rate up to 100.0 % on EMV cards. Merging the classification of several dumps of the same application reaching a rate of 100.0 % with only 3 merged dumps for considered fields.

References

1. Alcover, P.M., Guillamón, A., del Ruiz, M.C.: A new randomness test for bit sequences. Informatica **24**(3), 339–356 (2013)
2. Avoine, G., Kalach, K., Quisquater, J.-J.: ePassport: securing international contacts with contactless chips. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 141–155. Springer, Heidelberg (2008)
3. Burdach, M.: Physical memory forensics (2006). https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Burdach.pdf
4. Calypso CNA: Calypso. http://www.calypsostandard.net/
5. Cohen, M.I.: Advanced carving techniques. Digital Invest. **4**(3), 119–128 (2007)
6. Coisel, I., Sanchez, I., Shaw, D.: Physical attacks against the lack of perfect forward secrecy in dect encrypted communications and possible countermeasures. In: International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 594–599 (2015)
7. Doğanaksoy, A., Çalık, C., Sulak, F., Turan, M.S.: New randomness tests using random walk. In: National Cryptology Symposium II (2006)
8. EMVCo: EMV integrated circuit card specifications for payment systems, June 2008
9. Freund, Y., Schapire, R., Abe, N.: A short introduction to boosting. J. Jpn. Soc. Artif. Intell. **14**(5), 771–780 (1999)
10. Friedman, W.F.: The Index of Coincidence and its Applications in Cryptanalysis. Aegean Park Press, California (1987)
11. Hastie, T., Rosset, S., Zhu, J., Zou, H.: Multi-class adaboost. Stat. Interface **2**(3), 349–360 (2009)
12. Kajdanowicz, T., Kazienko, P.: Boosting-based sequential output prediction. New Gener. Comput. **29**(3), 293–307 (2011)
13. Knuth, D.E.: The Art of Computer Programming: Seminumerical Algorithms, vol. 2. Addison-Wesley, Reading (1997)
14. Lanet, J.L., Bouffard, G., Lamrani, R., Chakra, R., Mestiri, A., Monsif, M., Fandi, A.: Memory forensics of a java card dump. Smart Card Research and Advanced Applications. LNCS, vol. 8968, pp. 3–17. Springer, Heidelberg (2014)
15. Laurie, A.: Rfidiot. http://rfidiot.org/
16. Pannetrat, A.: Cardpeek. http://pannetrat.com/Cardpeek/
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
18. Poisel, R., Tjoa, S.: A comprehensive literature review of file carving. In: 2013 Eighth International Conference on Availability, Reliability and Security (ARES), pp. 475–484. IEEE (2013)
19. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document April 2010
20. Shamir, A., van Someren, N.: Playing hide and seek with stored keys. In: Franklin, M.K. (ed.) FC 1999. LNCS, vol. 1648, pp. 118–124. Springer, Heidelberg (1999)
21. SKIDATA AG: Skidata. http://www.skidata.com/en.html
22. Su, J., Zhang, H.: A fast decision tree learning algorithm. AAAI **6**, 500–505 (2006)
23. Sulak, F.: A new statistical randomness test: saturation point test. Int. J. Inf. Secur. Sci. **2**(3), 81–85 (2013)

24. Sulak, F., Doğanaksoy, A., Ege, B., Koçak, O.: Evaluation of randomness test results for short sequences. In: Carlet, C., Pott, A. (eds.) SETA 2010. LNCS, vol. 6338, pp. 309–319. Springer, Heidelberg (2010)
25. Van Deursen, T., Mauw, S., Radomirovic, S.: mCarve: carving attributed dump sets. In: USENIX Security Symposium. pp. 107–121 (2011)
26. Yoo, B., Park, J., Lim, S., Bang, J., Lee, S.: A study on multimedia file carving method. Multimedia Tools Appl. **61**(1), 243–261 (2012)