Smart Card Reverse-Engineering Binary Code Execution Using Side-Channel Analysis

François-Xavier Aranda
Thales 3S,
18 Avenue Édouard Belin, 31000 Toulouse, France
Email: francois-xavier.aranda@thalesgroup.com

Jean-Louis Lanet
SSD Team – XLIM/Université de Limoges
83 Rue d'Isle, 87000 Limoges, France
Email: jean-Louis.lanet@xlim.fr

***Abstract*—Using side-channel observations to reverse-engineer code execution is a research area that has been refined over the past decade. Mostly done on protected encryption algorithms at the beginning, these particular analysis aim more and more global code recovery. Based on the statistical behavior of a targeted platform, they study and look for a way to characterize the execution of instructions into side-channel signals. Disassembling a program, based only on power statistical analysis and some public material, is a challenge that is considered in very few papers. The most recent one applies "Template Attacks" and recovers most of instructions executed. Another choose pattern recognition to build a dictionary in order to recover code sequences, while a prior one uses "Differential Power Analysis" to understand and retrieve the implementation of known, or unknown, block cipher algorithms such as DES[1]. We present these three approaches in this paper as a state of the art of reverse-engineering methods aiming code recovery while using side-channel analysis.**

I. Introduction

A smart card usually contains a microprocessor and various types of memories: RAM (for runtime data and OS stacks), ROM (in which the operating system and the romized applications are stored), and EEPROM (to store the persistent data). Due to significant size constraints of the chip, the amount of memory is small. Today, most smart cards on the market have at most 5 KB of RAM, 256 KB of ROM, and 256 KB of EEPROM. A smart card can be viewed as a secure data container, since it securely stores data and it is securely used during short transactions. Its safety relies first on the underlying hardware. To resist from probing an internal bus, all components (memory, CPU, cryptoprocessor, etc.) are on the same chip which is embedded with sensors covered by a resin. Such sensors (light sensors, heat sensors, voltage sensors, etc.) are used to disable the card when it is physically attacked. The software is the second security barrier. The embedded programs are usually designed neither for returning nor modifying sensitive information without guaranty that the operation is authorized.

Smart cards are devices prone to attacks in order to gain access to services or assets stored by the card. Several means have been used to retrieve these valuable information and side channel analysis or fault injection appears to be the most efficient. One of the assets is the program used by the smart card. Having a precise knowledge of the software and in particular the control flow used could be helpful for new attacks or understanding the algorithms.

Reverse engineering is a process of reading the software's binary code to find what the software can make the computer do. The term reverse engineering has its origin in the analysis of hardware where the practice of deciphering designs from products is commonplace. While the hardware objective traditionally is to duplicate the system, the software objective is most often to gain a sufficient design-level understanding. There is a lot of commercially available reverse engineering tools that can provide a set of limited views of the source code. Unfortunately none of them can be applied to the smart card world, because the binary code is romized and no access to the code is provided to the attacker. The only way is to analyze information that leak from the system. One is particularly interesting, the power consumption which reflects the activity of the processor.

[1]Data Encryption Standard.

II. Instructions Recovery Problem and Side-Channel Analysis

Before starting to describe the various methods developed for instructions recognition on informations' systems, we need to clearly expose both reasons and issues addressed to this particular research area. It should be a good way to understand the ins leading of such methods. As side-channel analysis is not only used for reverse-engineering processes but also in plenty other domains, we consider interesting to present furthermore these procedures offering ways to extract protected informations and exploit them using statistical methods.

A. Why Recovering Instructions Sequences?

Obviously, a code sequence should be considered as a sensitive information and should be protected from disclosure. Intellectual property and safety are important in software industry as developing new algorithms and implementations is expensive and time consuming. In the cryptography domain, knowing an algorithm architecture can reveal defaults and damage both its integrity and efficiency.

Reverse-engineer an algorithm implementation can also be used by security providers to check if common flaws are avoided to guaranty a product marketing. So, being able to prevent software misuse and protect source code from replication is critical, not only in security domain but generally in computer programming.

However, recover which instructions are executed on information systems is not trivial and requires some mathematical, physical and electronics background to be performed. First, an attacker should be able to gain knowledge on the targeted platform. Then, he should be able to exploit this information either by observation or statistic analysis. Public materials as physical leakage are, of course, very useful during a

characterizing step. This modeling process requires several skills and imagination to find the better way to fit a system behavior. For all these reasons, code sequences reverse-engineering is capital and requires special attention.

B. Side-Channel Analysis and Emission Sources

Using electric devices, such as embedded systems, to store or compute information may involve some risks of disclosure. When an electrical system is on, its power consumption, for example, may vary depending on which operations it processes or data it carries. A malicious user of this platform could exploit this information to gain protected knowledge, and use it to corrupt or reproduce a targeted system.

Side-channel leakage can be of many kind. All has something to do with electrical devices power consumption. This is why we start our presentation of side-channel analysis with power consumption monitoring. Figure 1 shows a way to capture power consumption signals from a microprocessor.

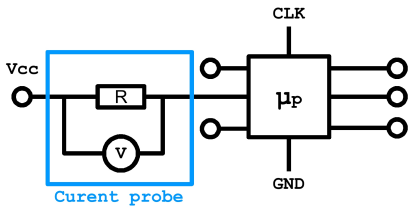

Figure 1. Current probe installation scheme on a microprocessor.

If power traces differ because operations or entries from one signal to another, one can easily conclude that informations are leaking through out the device. Quantifying this information and using it in order to understand inhere processes is part of the side-channel analysis. It can lead an attacker to use statistical analysis as well as signal theory to extract informations before exploiting it.

The very first side-channel source was the time difference between two same subroutine executions using two different entry data. This kind of attacks, so called "Timing Attack" [10], was particularly useful to get knowledge on RSA[2] keys during the "Square and Multiply" step of modular exponentiation. In fact, the algorithm duration strongly depends on the different operations it performs. By observing precisely enough the signals retrieved, attackers were able to simply read key bits during exponentiation.

Later, consumption model evolved to increase the precision on inhere data manipulation. SPA[3] methods were able to prove that, not only power consumption is influenced by computation, but it is also related to data processing.

In fact, as mathematical operations are performed by switching logical gates states, these changes need an electrical impulsion theoretically detectable on power traces. This can be an issue when the leaking implementation is meant to provide a secure encryption such as DES or RSA. In RSA case, once again, the "Square and Multiply" algorithm flaw depends on key manipulation. When a zero is treated, only the square step was performed, otherwise it performed both square and multiply operations.

[2]Rivest Shamir Adleman.
[3]Simple Power Analysis.

Simultaneously to SPA, another well known attack was developed to exploit the differences between two signals to extract informations. DPA [11], uses the bias caused by a crypto-system's power consumption while it processes and uses cipher keys. One of the great improvement compared to SPA was that this method settle for noisy signals as it performed a cleaning step by characterizing it.

Using statistical analysis of a system power consumption, while executing thousand different operations, provides to a malicious user a consumption model. It allows him to reproduce the targeted system electrical behavior with a certain measurable error. Next, by analyzing the differences between the simulated consumption and the actual one while guessing the value of the key bits, being right or wrong. This method allows him sorting this proposition and recover the encryption key.

With quite the same philosophy, CPA [1] is based on statistical models. It differs from DPA by its discrimination tool. During CPA, the correlation coefficient value is used to evaluate a guessed proposition. This method uses a consumption model based on the Hamming distance or weight, itself validated by experiments.

Always with the same goal of better approaching a system behavior, the Hamming model uses theoretical aspects of logical gates while switching from a state to another. If a logical gate does not switch, its own electrical consumption shall remain the same. By considering that the overall system consumption is theoretically proportional to the sum of the logical cells' one, this model focuses on their state inversion or preservation to estimate a system power consumption. Then, one can compare the estimation results to an actual signal with statistical tools like correlation coefficient.

Other signals carry information one can use to perform statistical analysis. For example, due to magnetic induction, any current variation provides a magnetic field fluctuation which, being related to power consumption, leaks exploitable information. Besides this physic fact, variations into electro-magnetic field give very useful spacial informations for an attacker. These emanations, alone or combined to others side-channel signals, can be also used to reduce monitoring noises coming from non targeted functionalities.

For example, if one knows where a specific key is stored, by positioning an electro-magnetic probe at this location, he can better target signal acquisition. Therefore, each time this data is manipulated, the magnetic traces obtained should be more precise and give better results than the power analysis methods presented before. Henceforth, these attacks are named SEMA, DEMA and CEMA, where EM stand for "Electro-Magnetism" [14], [13], [6].

Last, we quickly present two other ways to gain knowledge with side-channel sources. Because of Joule effect, any resistive electrical system results in heat generation. The thermal difference gives a spacial information on where specific operations are executed [12], [16], [9]. Like in electro-magnetism based attacks, and according to the heat intensity, one can distinguish if memories are read or written. It can be used, for example, to retrieve where are stored static tables used by encryption algorithms.

Light emission can also be used to locate parts of algorithm. This method was recently used to perform DPA on a

FPGA[4] implementing a DES algorithm [5]. This particular method requires the attacker to proceed to the opening of the ceramic package that generally protects any system in order to observe the light emission. Unlike previous side-channel sources, this one uses an invasive method to get the information and thus is not used very often. This concludes our overview of side-channel origins and analysis that are commonly used for reverse-engineering method.

III. Reverse-Engineering of Instruction Sequences

Here, we discuss various prior works on code recovery using side-channel analysis. As seen during Section II-A, many low-programming problems could be avoided by recovering the actual code sequence executed on a system. We have seen that the major difficulty was to clearly identify operations or data manipulated, which side-channel analysis can answer in many situations.

Our state of the art presents chronologically three previous papers describing three original ways to treat this reverse-engineering problem. For each, we present context and knowledge needed to precisely understand the whole process. Then we describe each step for each method before listing pros and cons, and give our opinion.

A. SCARE of the DES

Published in 2005 during "Applied Cryptography and Network Security" conference, this article [4] written by Remy Daudigny, Herv Ledig, Frederic Muller and Frederic Valette, presents SCARE[5] on block cipher algorithm DES. By observing power consumption against ins and outs of the encryption standard, they manage to recover specific implementation of each of its component.

This method was applied on DES but could have been tested on any symmetric algorithm using block encryption. The interest to test this approach on DES was that its architecture was public and allows quick verification on the reverse-engineering results. Based on Christophe Clavier's work on A3/A8 algorithm [3], they consider having no information on the implementation of each step of the encryption process before applying their method. In next section, we quickly remind DES scheme before precisely describing the attack principles and present its application.

1) Data Encryption Standard Scheme and History: This block cipher algorithm using private keys was selected by the United States of America, National Bureau of Standard in November 1976. At this time, IBM was proposing its algorithm called "Lucifer" and designed by Horst Feistel in 1971. After several modifications, asked by the National Security Agency, the Lucifer child, DES, obtained its standardization in 1977.

On Figure 2, we detail Feistel network, still used nowadays for various block cipher algorithms and hash functions, such as Threefish, Triple DES or SHADE. As one can see, it is composed of several nested rounds using XOR and non linear function F combinations. This last uses a different key for each loop, itself derived from a master key. The Figure 3 depicts the key scheduling process for the DES algorithm.

[4]Field-Programmable Gate Array

[5]Side-Channel Analysis for Reverse-Engineering

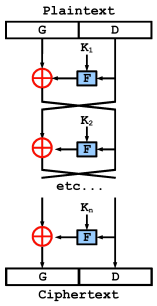

Figure 2. Feistel Standard Scheme.

DES needs 56-bit keys and processes 64-bit data blocks. It requires 16 rounds, each one spiting the previous 64-bit message into two 32-bit data blocks. The right part of the previous message is computed with the F function while the left part is added to the resulting block.

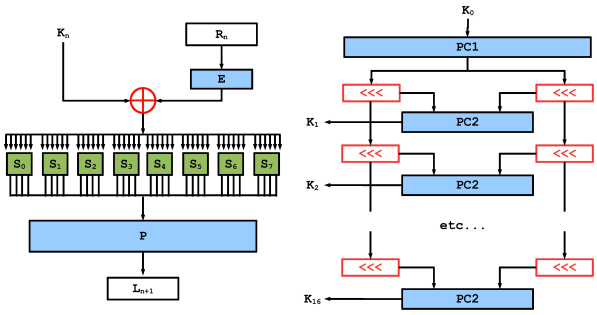

Figure 3. On the left : DES non linear function F. On the right : Key scheduling scheme of DES.

From one Feistel scheme to another, the F function and key scheduling method differ but not necessarily the global structure. In DES case, key schedule uses shift operation and two Permutation Choice tables ($PC1$ and $PC2$) to provide subkeys to each round. Its non linear function is described on Figure 3.

For its F function, DES computes the right part of the previous round result with an expansion table E before adding the output to the round subkey. Next, thanks to eight S-Boxes providing a non linear transformation, it reduces the 56-bit intermediate message to a 32-bit one. Last, a Permutation table P mixes the message bits to provide, to the next round, its left part of the message.

Before executing the 16 loops to the 64-bit message block, DES applying an Initial Permutation (IP) on it. At the end of iterations, the inverse table (IP^{-1}) is applied to the last intermediate 64-bit message before giving the encrypted one.

2) SCARE Principles: The article goal is to demonstrate that using side-channel information leakage on a system implementing a DES algorithm can lead to disclose tables values of each Feistel component. These secrets, constant part of the algorithm, are supposed to be unknown and DES example is taken to prove the method.

The authors propose to use statistic power analysis to see which message bit has the strongest probability to be manipulated at given clock cycles. To perform DPA, the attacker

needs M power traces corresponding to M messages being encrypted by DES. Lets $\tau_i(t)$ be the power consumption observed for the message i at time t. They consider time division as clock cycles, meaning a power trace value at time t represents its value for the clock cycle index t. Next, they choose one bit a from plain-text to sort the power traces into two groups G_0 and G_1, depending on its value. Then, for each clock cycle t they compute the V_t value :

$$V_t = \left| \frac{1}{|G_0|} \sum_{\tau_j \in G_0} \tau_j(t) - \frac{1}{|G_1|} \sum_{\tau_j \in G_1} \tau_j(t) \right| \quad (1)$$

If $V_t > \lambda$, λ being an appropriate threshold, the bit a is manipulated during clock cycle t otherwise it is not.

After being able to detect group of bits manipulation during each table application, they manage to rebuild its values. For example, while expansion table is applied, the first group of bits manipulated together during 5 clock cycles are bits 0, 1, 2, 3, 4 and 31. Now, it turns out the first line of expansion table manipulates these 6 bits. The 5 next clock cycles observations show a manipulation of bits 3, 4, 5, 6, 7 and 8, corresponding to the second line of E table.

They continue their attack and finally succeed in recovering each table values presented before, except for the S-Boxes. They voluntary skip this part, arguing that the previous Clavier's work, on A3/A8 algorithm, do a similar attack to recover tables from a private cipher algorithm. However, they find a specific implementation by observing each S-Box independently. They saw that, when the second S-Box was applied, the output bits from the first one was also manipulated. They explain these results by the fact that S-Boxes output may be temporarily stored into the same register.

They make the same observation while monitoring the key schedule process. It seems that the round sub-key bits are using the same register than the intermediate cipher bits. They conclude that this information may be useful if an attacker wants to retrieve a key with a known plain-text/cipher attack.

The authors concluded their article by justifying that the SCARE method is not only for reverse-engineering private block cipher algorithm but can also be considered as a prior step to more conventional attacks such as SPA and DPA. Obviously, facing the amount of information depending on implementation they manage to recover, an attacker may accelerate its classic statistic analysis by using SCARE in order to better target key recovery.

3) Observations on SCARE: Even if this method is not clearly aiming code recovery, it manages to understand and highlight some implementation particularities depending on the system monitored.

Also it mixes multiple methods, from the statistical one to simple observations, and insists on the fact that observing the power consumption by clock cycle can provide priceless informations on the targeted system. Finally, this attack clearly proves that sensitive data stored in memories are not safe against reverse-engineering method.

B. Reverse Engineering of Embedded Software Using Syntactic Pattern Recognition

This publication, compiled into "Lecture Note of Computer Science" in 2010 [7], was written by Mike Fornigault, Pierre-Yvan Liardet, Yannick Teglia, Alain Tremeau and Frederique Robert-Inacio. This work claims that observing precisely enough a system consumption during a code sequence execution, an attacker using SPA should be able to recognize consumption pattern of a given instruction. Therefore, the authors propose to build a statistic pattern for each instruction and provide a dictionary to a recognition module to reverse-engineer executed code sequences.

Their method consists in associating an instruction to a power signature, itself composed by multiple prototypes coming from SPA observations. A similar technique has been presented by Dennis Vermoen, Marc F. Witteman and Georgi Gaydadjiev in 2007, only this time aiming reverse engineering of JavaCardapplets [15]. With an efficient classification tool, and thanks to convex form comparison, they are able to evaluate the probability for an instruction, executed on a CISC[6] architecture, to correspond to a given power trace.

As CISC systems are specific, we will quickly remind the architecture before presenting the different steps of this pattern recognition based reverse-engineering method.

1) CISC Architecture: Generally opposed to RISC[7] architecture, CISC micro-processors allows multiple complex addressing modes, and thus, provides big instruction set. The best example for this kind of structure is the x86 Intel family. They authorize micro-programing (i.e. instruction redefinitions) and was meant to fill the gap between low level and high level code design.

CISC is able to great modularity because each instruction is a complex combination of simple statements allowing easy ALU[8] register manipulation.

Later, thanks to compilation evolutions, CISC loosed their interest and tend to be replaced by cheaper and faster RISC architectures. In fact, accelerating this type of processor was an issue whereas RISC was already allowing instruction prefetch process. So, because of their lack of flexibility, their speed limits and the constraints they are no longer used for electronic computing devices.

2) Methodology Steps: This attack consists in two main steps. The first one characterizes the power consumption signals for each instruction. They define the power consumption $P(I)$, corresponding to an instruction I, as a linear combination of instruction executions and data manipulation, for one part, and consumption linked to previous instruction execution for the other part.

Using experimental results, they observe that signal components linked to previous instruction are negligible. Thus, they describe $P(I)$ as follow :

$$P(I) = P_{operation} \times \epsilon_{operation} + P_{data} \times \epsilon_{data} \quad (2)$$

where $P_{operation}$ represents the power consumption of instruction's I execution, P_{data} the data one and ϵ the associated noise to each component.

[6]Complex Instruction Set Computing.
[7]Reduced Instruction Set Computing.
[8]Algorithm Logical Unit.

The learning stage, following characterization, abstracts each instruction as a combination of general atomic statements. These operations can load a register value, add it, proceed to binary operations such as "and" and "or", and finally, multiply it to other values. Knowing the general statements used for each instruction, an attacker should be able to build signature prototypes characterizing a given power trace.

This first step looks like a dictionary production. To do so, each clock cycle in power trace corresponding to an instruction execution is split into significants consumption peaks like depicted on Figure 4, each instruction being executed on R clock cycles.

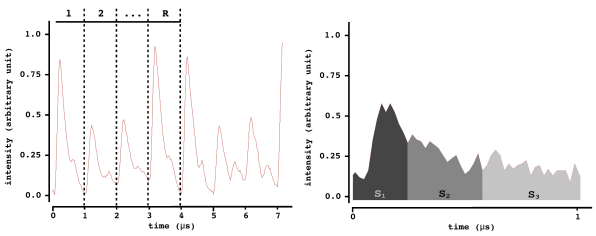

Figure 4. On the left : power consumption during an instruction execution. On the right : a clock cycle detail.

For each relevant peak composing a clock cycle, a pattern S_i is associated. The combination of all patterns defines an instruction signature. To quantify the similarity between two convex forms, a reference A, and a given one X, the authors use the value $p_{A_X}(X)$ defined as follow :

$$p_{A_X}(X) = \frac{1}{\lambda_X(A_X)} \frac{\mu(X)}{\mu(A_X)} \tag{3}$$

with A_X, the smallest homothety of A contained into X, $\lambda_X(A_X)$ the homothetic coefficient of the smallest X homothety contained into A_X, $\mu(X)$ and $\mu(A_X)$ being the respective surfaces of X and A_X.

Similarity between two instructions, I_1 and I_2, signatures during clock cycle k is evaluated by maximizing the previous parameter p for each S_i and finally sum these maximums like defined on Equation (4).

$$M(C_{k,1}, C_{k,2}) = \sum_i max_l(p_{S_{1,j}}(S_{2,l})) \tag{4}$$

The learning steps find the best signature prototype for each clock cycle. By monitoring several executions of an instruction carrying various data, it obtains an average consumption pattern for the targeted instruction thanks to the M value. The prototype having the most similar signatures to the power traces observed is selected as a clock cycle signature. By repeating this step for each clock cycle composing an instruction execution, the learning step provides for each instructions a pattern stored in a dictionary.

The recovery stage simply consists in comparing a power trace to the pattern dictionary and selecting the best similarity score for each group of R clock cycles to retrieve the most likely instruction executed.

3) Method analysis and remarks: Even if this reverse-engineering method is very specific to a particular architecture, it proposes an interesting scheme. It proves that CISC systems are good candidates for pattern recognition because of instruction granularity.

As each instruction uses the same statement set and being able to characterize and precisely recognize these statements, this article proposes an efficient method to recover code sequences based on a signature discrimination. The pattern construction step is important and finally provides all tools needed for recognition stage. This prior work is the first one offering a complete code reverse-engineering solution even if it seem complicated to adapt it on other architectures.

C. Building a Side-Channel Dissassembler

This article published into "Transactions on Computational Science" journal in 2010, was written by Thomas Eisenbarth, Christof Paar et Björn Weghenkel. It echoes a previous work done by Martin Goldack [8] presented as a Master Thesis in 2008. Using Chari's "Template Attacks" [2], it suggests a statistical approach to the code recovering problem. Therefore, its title is unequivocal and supposes that this method can be applied on any platform types, which is a huge improvement compared to previous works.

Before presenting the different aspects of this method, we introduce "Template Attacks" principles and "Hidden Markov Models" used for the system abstraction. Again, the attack consists in two separated steps, one for characterization, the other for instruction recognition. Side-channel statistic analysis and platform modeling take much more importance here than in any other prior work we treated, hence we consider this paper the most advanced one.

1) Template Attacks: This kind of method supposes that the attacker has an open and programmable platform identical than the one targeted. He should be able to extract a model from it, allowing him to proceed to code recognition on any similar closed system.

Unlike most of statistic analysis methods trying to reduce noise impact in order to extract information from experimental observations, "Template Attacks" aim a multivariate noise modeling for this purpose. In fact, monitoring on an open sample lots of experimental behaviors allows a white noise characterization whereas, for example, DPA reduces its impact by difference.

Theoretical principals suggest a Bayesian classification of operation set provided by a system. This classification permits hypothesis reduction on what is really executed by the platform. Usually, an attacker tries to maximize resemblance between an observed sample and a theoretical model. In "Template" approach, the better the noise model fits, the less it needs an estimation.

This method first focuses on monitoring, for a given operation O_i, lots of traces with variation of data carried. Then, for each operation, it averages the curves obtained, M_i, and looks for relevant points where average dispersion between operations is maximal. This dimension reduction is necessary in order to proceed complex computations only on selected points $(P_1, \cdots, P_N)$ in order to make easier operation discrimination. Last, for each O_i, it builds noise vector $B_i(T) = (|T(P_i) - M_i|)_{i \in [1 \cdots n]}$, for each sample T. With covariance matrix, $\Sigma_{B_i}[u, v] = cov(B_i(P_u), B_i(P_v))$ and mean vector M_i, the method suggests a template for O_i operation thanks to the tuple (M_i, Σ_{B_i}).

Then, the attack considers that the signal emitted by the system during O_i execution is M_i, and that the probability to have a noise vector n is given by a multinormal distribution detailed in Equation (5).

$$Pr_{N_i}(n) = \frac{exp(-\frac{1}{2}^t n \Sigma_{B_i}^{-1} n)}{\sqrt{(2\pi)^N |\Sigma_{B_i}|}} \quad (5)$$

To select the best operation due to a certain observations, the method reduces hypotheses by maximizing this probability of noise apparition while minimizing the cumulated error of rejected operations. In fact, as noise characterization follows statistic laws, and its possibility to quantify error while comparing operations, reducing hypothesis allows an attacker to associate an operation to a certain error probability. There, to clearly identify an operation against a given power trace, he should select the one with the least error probability value.

2) Hidden Markov Models: Mostly used in automatic treatment for language and pattern recognition, these tools are meant to statistically model hidden-state machines. The following game is usually taken as an example to explain Markov models.

Lets have two bags, A and B, each containing a certain quantity of tokens a and b. Considering two other bags, A' and B' each containing a certain quantity of tokens j and k. The game start with bags A and A'. We take one token to each one and write down the value of the one coming from A'. Before putting back each token in its origin bag, we look at the one coming from bag A. If its value is a, the next draws will still be done from bags A and A', otherwise, for a b value, from bags B and B'.

Two sequences can be extracted from this game. The first one will be an information emission (writing down the token value), the second one will be considered as a transition. This procedure is illustrated by Figure 5.

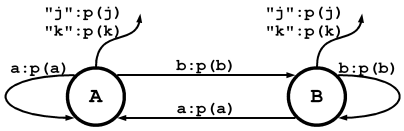

Figure 5. A two states Hidden Markov Model example.

This abstraction of an information system seems relevant as any platform executing an instruction, i.e. switching from one state to another, can produce side-channel signals.

3) Presentation of the Disassembling Method: Like previous work, this one suggests two main steps for its reverse-engineering process. First a learning stage, where each instruction is associated to a template. Then a recovery step using a Markov scheme.

The profiling step builds a template for each instruction as defined earlier, independently from the data manipulated by the instruction. Thanks to the estimation of the instructions related power consumption distribution, the method builds a Bayesian classification using a theoretical multinormal distribution. Therefore, the probability $Pr(x|\mu_k, S_k)$, for a signal sample x, to belong to a given class C_k is defined by Equation (5). To build a template, the method defines the noise vector $n = x - \mu_k$, the average vector μ_k of C_k, and its covariance matrix S_k.

Now, lets have an observation corresponding to the execution of an instruction, which is independent from the samples used before to statistically build the Bayesian classes. Its belonging to one of these last corresponds to a maximization of the probability $Pr(\mu_k, S_k|x)$, $\forall(\mu_k, S_k)$. Because we need to be able to invert the S_k matrix during classes distribution estimation, the authors proceed to reduce the signals size while maximizing the variance. To do so, they use two similar methods : Principal Component Analysis and Fisher's Linear Discriminant in order to select relevant points in signals like the strict application of "Template Attacks" say so.

As $Pr(\mu_k, S_k|x) = Pr(x|\mu_k, S_k) \times Pr(\mu_k, S_k)$, with $Pr(\mu_k, S_k)$ being the occurrence probability in a code sample of a given instruction which template is (μ_k, S_k), and because Hidden Markov Models needs transition probability between each states, we need to study occurrences of both single and pair of instructions. This step is performed by analyzing various public code samples plus some specific implementations of encryption algorithms the authors provide. Once these language grammar information extracted, the Markov Model can be initialized and the belonging probability is computed.

The code sequence recovery consists in parsing the Markov state machine while testing, for each transition, which synthetic signal emitted better fits the actual one. Thanks to the parsing algorithms of Viterbi and "Foward-Backward", an attacker can use its model to retrieve the instructions executed.

The first algorithm tries to maximize the probability of a path in Markov Model by focusing only, for each recursive depth, on the emission compatibility before transitions probabilities. However, during the recursive backtrack, it takes into account the transition to correct errors that could have made by template comparison.

The second algorithm, does the opposite by first looking for the most likely path into the state machine only considering transitions probabilities. Once done, it uses template comparison to clearly choose the best way to parse the model until a solution pop out.

To the authors, the best solution seems to be the Viterbi one because of its results and low complexity compared to the "Foward-Backward" one.

4) Article Overview: First we wanted to highlight the fact that this article was the last one published in this domain. Its use of "Template Attack" is quite a smart solution to the instructions' identification problem. Also, its system abstraction using Markov models fits well the actual behavior of an executing platform. However, despite of the results they obtained, these mechanisms are both dependent to the statistical grammar information extracted from code sample analysis.

The probability for a given template to correspond to one instruction needs all instructions occurrences to be computed. Then, the Hidden Markov Model requires transition probability between instructions' pairs to be initialized. These two information could be biased and should be used carefully. Even if we consider that using them could bring some application issues, the overall method is well described and is, for now, the best analytic method existing to answer code sequences reverse-engineering using side-channel analysis

problems.

IV. Conclusion

We have presented here the state of art in reverse engineering code for devices in which the code is not accessible. In fact all these works represent the first step of reverse engineering which consists mainly in retrieving high level information from a binary code. We have shown that this step was possible even in the absence of binary code using side channel information. The most difficult operation is the instruction recognition using only the consumption traces. We are currently improving these methods using a novel approach based on genetic algorithms for classification. This approach is a multi criteria optimization problem and such algorithms seem in a first attempt to provide good recognition score lowering false positives. Therefore, this state of art was mandatory to overlook the different approaches already available for this purpose to yet develop our own. As these methods are each specific, being able to synthesize them also helps us on our current work.

References

[1] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[2] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[3] C. Clavier. An Improved SCARE Cryptanalysis Against a Secret A3/A8 GSM Algorithm. In P. D. McDaniel and S. K. Gupta, editors, *ICISS*, volume 4812 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2007.

[4] R. Daudigny, H. Ledig, F. Muller, and F. Valette. SCARE of the DES. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 393–406, 2005.

[5] J. Di-Battista, J.-C. Courrège, B. Rouzeyre, L. Torres, and P. Perdu. When Failure Analysis Meets Side-Channel Attacks. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2010.

[6] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[7] M. L. Gavrilova, C. J. K. Tan, and E. D. Moreno, editors. *Transactions on Computational Science X - Special Issue on Security in Computing, Part I*, volume 6340 of *Lecture Notes in Computer Science*. Springer, 2010.

[8] M. Goldack, C. Paar, and T. Eisenbarth. Side-Channel Based Reverse Engineering For Microcontrollers. Master's thesis, Ruhr University Bochum, 2008.

[9] H. Huang, G. Quan, and J. Fan. Leakage temperature dependency modeling in system level analysis. In *ISQED*, pages 447–452. IEEE, 2010.

[10] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[11] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[12] M. Meterelliyoz, J. P. Kulkarni, and K. Roy. Analysis of sram and edram cache memories under spatial temperature variations. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(1):2–13, 2010.

[13] O. Meynard, D. Réal, S. Guilley, F. Flament, J.-L. Danger, and F. Valette. Characterization of the Electromagnetic Side Channel in Frequency Domain. In X. Lai, M. Yung, and D. Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 471–486. Springer, 2010.

[14] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In I. Attali and T. P. Jensen, editors, *E-smart*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[15] D. Vermoen, M. F. Witteman, and G. Gaydadjiev. Reverse Engineering Java Card Applets Using Power Analysis. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2007.

[16] J. Viraraghavan, B. Amrutur, and V. Visvanathan. Voltage and temperature aware statistical leakage analysis framework using artificial neural networks. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(7):1056–1069, 2010.