www.elsevier.com/locate/mejo

Defending Cache memory against cold-boot attacks boosted by power or EM radiation analysis

Mădălin Neagu, Salvador Manich

PII: S0026-2692(16)30468-2
DOI: http://dx.doi.org/10.1016/j.mejo.2017.02.010
Reference: MEJ4153

To appear in: *Microelectronics Journal*

Received date: 3 October 2016
Revised date: 9 January 2017
Accepted date: 16 February 2017

Cite this article as: Mădălin Neagu and Salvador Manich, Defending Cach memory against cold-boot attacks boosted by power or EM radiation analysis *Microelectronics Journal,* http://dx.doi.org/10.1016/j.mejo.2017.02.010

Defending Cache Memory against Cold-Boot Attacks Boosted by Power or EM Radiation Analysis

Mădălin Neagu

Department of Computer Science,
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Madalin.Neagu@cs.utcluj.ro, neagumada@yahoo.com

Salvador Manich

Department of Electronic Engineering
Universitat Politècnica de Catalunya-BarcelonaTech
Barcelona, Spain
Salvador.Manich@upc.edu

Abstract

Some algorithms running with compromised data select cache memory as a type of secure memory where data is confined and not transferred to main memory. However, cold-boot attacks that target cache memories exploit the data remanence. Thus, a sudden power shutdown may not delete data entirely, giving the opportunity to steal data. The biggest challenge for any technique aiming to secure the cache memory is performance penalty. Techniques based on data scrambling have demonstrated that security can be improved with a limited reduction in performance. However, they still cannot resist side-channel attacks like power or electromagnetic analysis. This paper presents a review of known attacks on memories and countermeasures proposed so far and an improved scrambling technique named random masking interleaved scrambling technique (RM-ISTe). This method is designed to protect the cache memory against cold-boot attacks, even if these are boosted by side-channel techniques like power or electromagnetic analysis.

Keywords: data scrambling, cache memories, differential power analysis, side-channel attack, error correction

1. Introduction

1.1. Motivation

In a recent report [1], the author pointed out that 62% of companies worldwide were subject to payment fraud in 2014 and that credit/debit cards are the second most frequent target of payment fraud. Mobile payments are a relatively

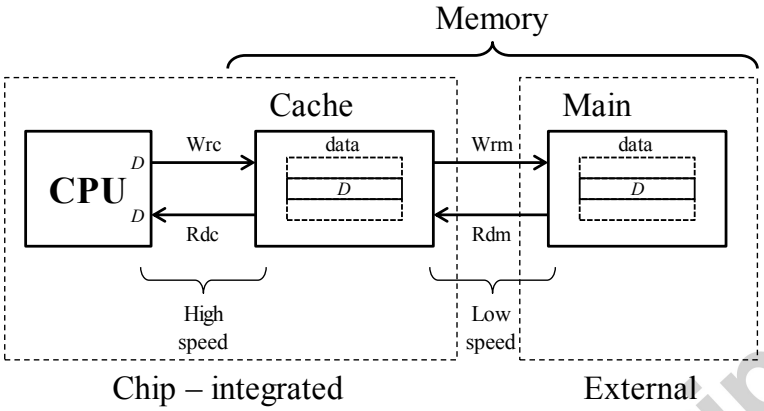


Figure 1: Simplified model of a computer memory system.

new payment method, but this trend is increasing among large companies and organizations. However, there are several uncertainties about it such as disclosure of sensitive information or secure transfer of information. Ensuring the confidentiality of sensitive information is becoming more and more crucial [2]. Very often general purpose devices like desktops, laptops or smartphones are used for private transactions with financial entities or healthcare issues, among others. In the case of devices without specialized hardware, all cryptographic operations are executed in software, resulting in an intensive use of memory [3]. This poses sensitive data at risk, including that stored in the cache memory [4].

Research on SRAMs has demonstrated that data can be maintained almost intact for a couple of minutes if the chip is kept at low temperatures or even at room temperature and without power supply [5]. This phenomenon is known as data remanence and results show that chip manufacturers do not control memory retention time as part of their manufacturing quality process. However, memory retention time varies between devices from the same manufacturer and of the same type but of different subtype or series. Also, low power versions of the same chips always seem to have longer retention times. This has opened up a whole new domain that has been widely investigated in the last decade. The scope of this work is in this realm.

When the CPU is running, it needs to work with sensitive data in plain form and, depending on the operating system, it may generate several copies in memory, exposing data to different kinds of attacks [3]. The problem with data remanence is that an adversary can use a cold-boot attack to extract critical data from memory by completely circumventing the software controlling the CPU [6]. This problem is discussed in depth in the following paragraphs.

1.2. Attacks on memory

A simplified model of a computer memory system is presented in Fig. 1. Only cache and main memories are shown. Cache memory usually has two

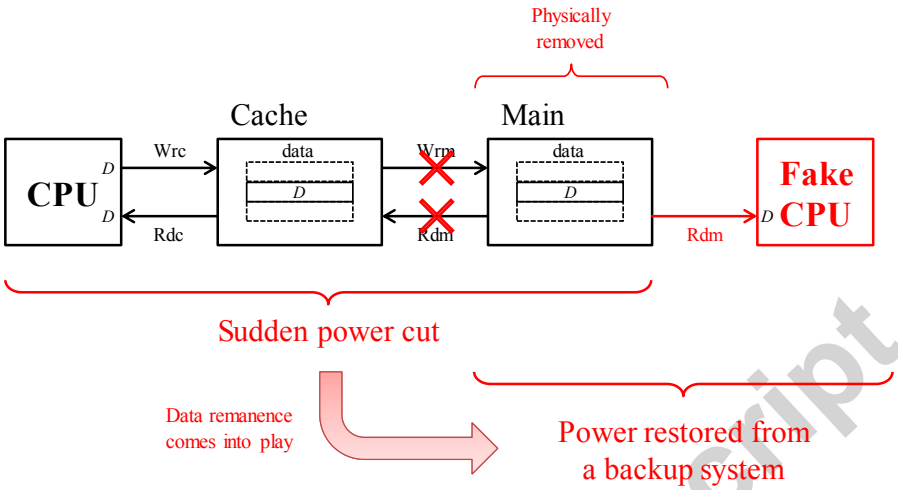


Figure 2: Cold-boot attack on main memory.

levels, L1 and L2, but this paper focuses on the L2 cache because it is the larger of the two and its size allows enough data to be stored for full computations, being considered in some cases as a secure memory [7]. Cache memory is always integrated in the CPU package, even embedded in the CPU chip or assembled with it using 3D technology in the most advanced versions. This makes it feasible to use a high speed bus for reading and writing data, keeping CPU performance at the maximum level. Main memory is commonly external; it can be expanded and communicates with cache through a lower speed bus. It reads data only in the event of cache miss and writes according to some policies like write-through or write-back.

Since the CPU reads more frequently than it writes, and often does so in similar address ranges, the main memory bus is much less used than the cache memory bus, typically 20 times less. For this reason, loss in transmission speed in the main memory bus has less impact on CPU performance than in the cache memory bus.

1.3. Cold-boot attacks

An overview of a cold-boot attack on main memory is illustrated in Fig. 2. While the software that stores sensitive data in memory is running, the power supply is suddenly disconnected and the main memory is rapidly removed, connected to a backup system and powered up again. Then, the victim's memory content is downloaded into a backup machine and from there critical data like encryption keys or any other type of sensitive data is extracted [6]. By cooling the memory modules, degradation of volatile memory is slowed down; hence, an adversary has more time to act. The results of the attacks in [6] [8] show that by cooling a memory chip at -50° C, decay within a 1MB region is 0.13% (after

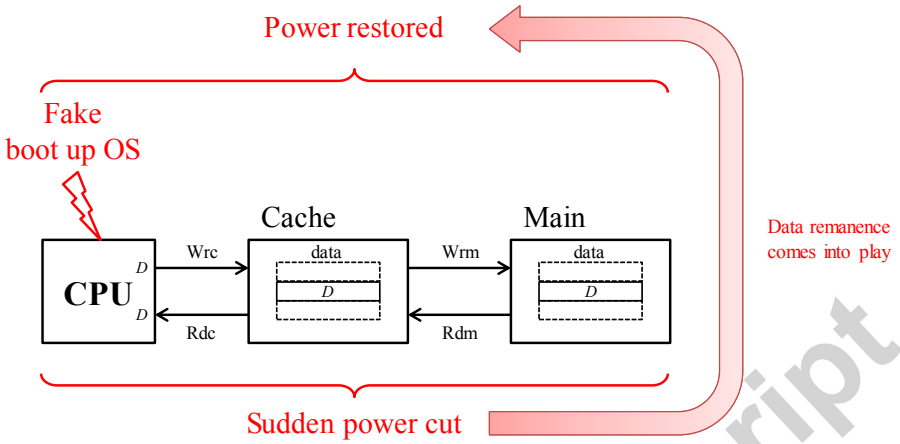


Figure 3: Cold-boot attack without removing memory chips.

60 minutes). Also, after a single minute without power supply, 99.9% of bits were recovered correctly.

This attack is not feasible on a cache memory because the latter cannot be physically removed. However, a variation in Fig. 3 illustrates how an adversary using cold-boot attack could steal data.

Once the CPU has run the software of interest and sensitive data has been stored in memory, power is suddenly cut off and immediately afterwards the system is booted up with an ad-hoc fake program which makes a backup copy of the memory. Next, data is analyzed and sensitive information extracted. As an example, this kind of cold-boot attack was conducted on several smartphones [9]. A simple reboot from an ad-hoc operating system (FROST) immediately made a backup copy of the memory content and the secret keys of encrypted flash data and other sensitive data were extracted with specialized software tools. The technique used managed to recover email messages, contact lists, credit card data and other login credentials after cooling the device to 5° C. These smartphones had a boot locking mechanism that ensured the deletion of data in the users partition and cache memory. However, not all models had this option activated by default, which might have been unknown to users.

In a recent work [7], several techniques to improve the security of smartphones and tablets were discussed. In these architectures, two types of memories internal to the CPU package can execute basic functionalities without accessing the main RAM, i.e. iRAM and L2 cache. They are considered secure mainly because after reboot, and this includes unexpected power cut offs, firmware cleans them up completely. Unfortunately, as said in [10], firmware can be attacked in many different ways, and thus it cannot be regarded as a strong security pillar.

The rest of the paper is organized as follows. In Section 2 security strategies for cache and main memory are reviewed. In Section 3, power and electro-

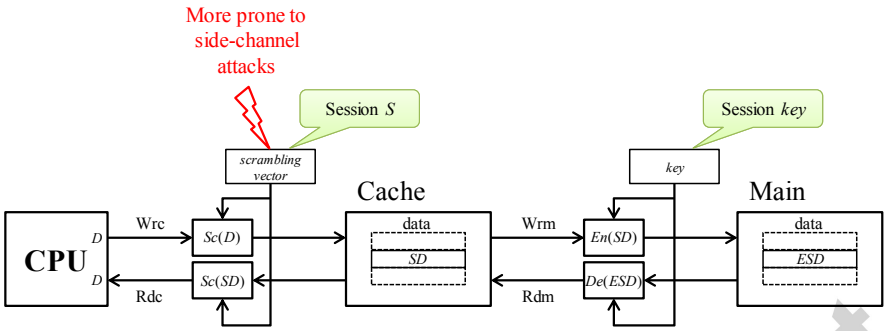


Figure 4: Published proposals securing memory at hardware level against cold-boot attacks.

magnetic radiation analysis are introduced. Next, in Section 4, the proposed solution is presented and in Section 5 the results are summarized. Finally in Section 6 the conclusions are discussed.

2. Securing memory at hardware level

In order to provide stronger security against cold-boot attacks, more close to the hardware solutions are needed, as pointed out in [7]. These can be complemented by other firmware and software techniques, leading to a whole solution reaching different abstraction levels of the system. A review of existing hardware solutions is first presented, and then other possible alternatives are proposed. Most can be placed in the general scheme of Fig. 4.

2.1. Main memory

The most secure way to protect main memory data is to encrypt it in real time. A session key is generated and a strong, secure algorithm like AES [11] or a low-latency one like PRINCE [12] is used to encrypt and decrypt according to CPU demand. The main advantage of this method is that the circuit providing the session key will change it if a reset condition occurs, which will invalidate all memory data completely, thwarting any attempt to read the memory after boot. Memory bus encryption makes use of time clearance provided by the cache memory such that the performance loss is buffered [13, 14, 15, 16, 17]. The drawback of this solution is that encryption must be executed by an independent co-processor to reach enough speed and robustness, and for power consumption to remain below reasonable limits. That is why portable devices are not expected to use this strategy mainly owing to power consumption. Some smartphones and tablets use a kind of memory encryption but this action is only applied during user locking / unlocking [7]. In [14], a refresh mechanism was added to the above scheme which changes the key periodically, thus strengthening protection against side-channel attacks. The key is generated by a specialized smart card IP. In [15], this strategy was selected to protect systems with non-volatile memory, like ferromagnetic RAM. Encryption is executed incrementally

and, in case of unexpected reboot, the whole memory is encrypted in 5 seconds. In [16, 17], memory encryption includes a time stamp to counteract replay attacks.

2.2. Cache memory

A similar protection scheme can be used for cache memory. However, no encryption process can be easily selected because of the high speed required by the bus. Scrambling techniques provide a lighter security degree. Data vectors are scrambled/descrambled with a session scrambling vector (S) by an XOR operation [18, 19]. As before, the circuit providing the S will change it after each reset, invalidating data in case of attempt to read cache after boot. The advantages are that high speed can be achieved by the scrambling and that the impact on power consumption is negligible. In fact, Intel uses this technique to transmit memory bus data, reducing high current peaks that could aggressively disturb the power supply lines [20]. The main drawback lies in that scrambling is not a securing but an obscuring technique which provides a low degree of security and are prone to side-channel attacks aiming to discover the S. Hence, frequent refresh of the S could improve the level of security, but cache data could become invalidated, and would need therefore to be updated. This would require the cache controller to shut down completely. In [18], scrambling was applied to data and addresses. The technique aims to defend cache memory against a wide range of side-channel attacks. It consists in a two-step scrambling process for both data and addresses with two different encryption keys stored in the main memory. Hence, a cold-boot attack on the main memory can disclose the memory section containing the encryption keys. In [19], a data scrambling technique was proposed to protect cache data as follows: the first half of a word is scrambled with the first bit of the first half, and the second half is then XORed with the scrambled result of the first half. The advantage is that since no scrambling vectors have to be stored in additional hardware, no data or area overhead has to be added. However, the patterns created by this method can be easily understood and peculiar data samples provide adversaries with a lot of information.

2.3. Interleaved Scrambling Technique

Interleaved Scrambling Technique (IST) is a security solution for cache protection against cold-boot attacks presented in [21]. It enhances standard scrambling [18, 19] because the scrambling vectors can be refreshed continuously without interrupting communication between the CPU and cache. It can also be integrated into a global protection scheme, as illustrated in Fig. 5.

Internally, IST works with pairs of interleaving scrambling vectors. One (the young one) is used for writing data to and reading data from the cache memory whereas the other (the old one) is used only for reading. Therefore, when the pair is active, the cache memory becomes filled with data scrambled by the young vector while being emptied of data scrambled by the old one. Once the cache is cleared of all data scrambled by the old scrambling vector, this

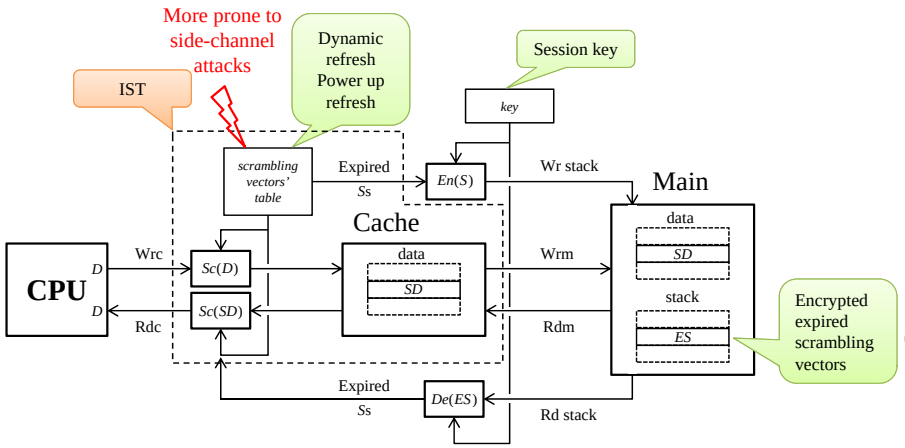


Figure 5: Interleaved Scrambling Technique. Hardware protection for cache that can be integrated into a global protection scheme.

vector expires and the pair is renewed such that the young scrambling vector becomes old and a newly generated scrambling vector becomes the young one. After reset, new scrambling vectors are generated again, invalidating cache data completely. IST can be used alone to protect the cache memory, as in [21], or be integrated into a global security solution. Fig. 5 shows one possible scheme which maintains high bus speed and a low level of power compared to alternatives that encrypt bus data. In this approach, scrambled data is also written back to main memory. The scrambling and descrambling process is all concentrated in the same unit, which is located close to the CPU. As scrambled data flow through all buses, they help to keep current levels stable, as pointed out by Intel in [20]. Data stored in memory but not copied in cache will need the corresponding scrambling vectors to be made available in case the CPU needs them. Thus, whenever a scrambling vector expires, it is made available encrypted with a session key in a buffer such that the CPU can store it in a non-cached memory page. In this way, encryption speed requirements are much lower than those of data transfer, scrambling vectors generated internally cannot be reverse engineered and cold-boot attacks on main memory data and scrambling vectors are thwarted.

2.3.1. Side-channel attacks on IST

Even though IST refreshes scrambling vectors periodically, memories protected by IST are still prone to side-channel attacks because the scrambling technique is not intrinsically robust against this type of attacks. By using power or electromagnetic radiation analysis, an adversary could discover the scrambling vector in use for writing after several attacks. Although this would be difficult in practice, he could theoretically descramble cache or main memory data downloaded after a cold-boot attack. In the coming paragraphs, this

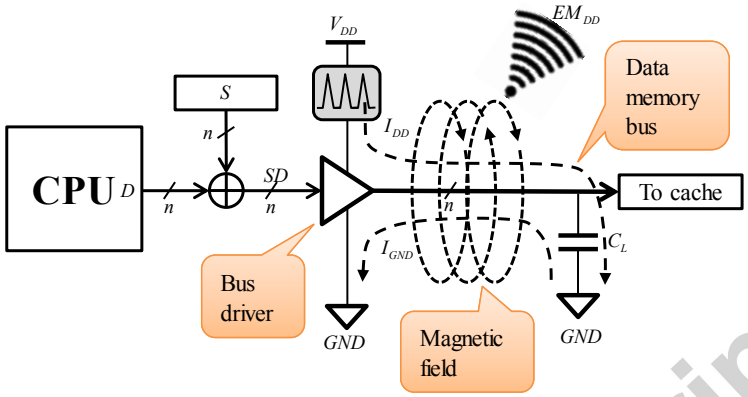


Figure 6: Switching drivers and bus line currents excite power peaks and electromagnetic pulses which leak information about data flowing into the drivers and buses

weakness is explained and the methodology to improve IST robustness against this type of side-channel attack is presented and evaluated.

3. Power (P) and Electromagnetic (EM) Radiation Analysis

For the sake of clarity and without loss of generality, our discussion focuses on a scrambling circuit using a single scrambling vector. Let us assume that a data vector D is placed in the data bus. Before it being sent to the cache, IST mixes it with a scrambling vector S through XOR gates such that the scrambled data vector $SD = D \oplus S$ is generated. Reverting this transformation (descrambling) is simple, $SD \oplus S = D \oplus S \oplus S = D$. A simplified model of the elements involved in information leakage is shown in Fig. 6.

Transmitting data flowing in a bus involves charging and discharging load capacitances, the parasitic capacitances of the lines themselves and of the next stage logic. This means that in each clock cycle some of the lines first source current from the power supply line and then drain it to ground. All switching bus lines source currents from the power supply line P_{DD} whose intensity is strongly related to the amount of scrambled data transmitted through the bus. Similarly, the magnetic fields of these currents add up and generate an electromagnetic wave EM_{DD} whose intensity depends on the scrambled data too. Hamming weight is a metric that adds up the contributions of individual bits in a word. Hence, it is often used to find correlations between data and power or radiated electromagnetic intesity [22].

Using this side-channel leakage an adversary can undergo the cache memory to a Simple Power or Electromagnetic Analysis (SPEMA) or even to a more powerful Differential Power or Electromagnetic Analysis (DPEMA). The aim of the attack is to recover the internal S with which the cold-boot attack can succeed afterwards.

In the most sophisticated version the attack would behave like this. Assume that the victim cache has sensitive data in it, scrambled with vector S_1. Immediately, the adversary attacks the victim with an SPEMA or DPEMA, estimates the scrambling vector S_1^* and carries out the cold-boot attack by abruptly disconnecting the supply and booting the system with an ad-hoc operating system. Promptly after, the adversary reads out the sensitive data from the victim cache and stores them in a backup memory. However, if the scrambling technique is like IST [21] after the cold_boot attack the scrambling vector will change to a new one, e.g. S_2 such that the stolen data recovered by the adversary will be now $SD = D \oplus S_1 \oplus S_2$, in which D is the sensitive data wanted. Therefore, the adversary will have to perform a second SPEMA or DPEMA attack to estimate S_2^*. In the final step the double scrambling is undone by the adversary on the stolen data ($SD = D \oplus S_1 \oplus S_2) \oplus S_1^* \oplus S_2^* \rightarrow D$ and the sensitive data is recovered provided that the estimations of S_1^* and S_2^* are correct.

In a conference paper [23] authors presented how an SPEMA attack could be carried out in a cache memory protected by a scrambling technique like IST and proposed a countermeasure against this consisting in to decrease the probability of correctly estimating S^*. In this paper this SPEMA countermeasure is extended to fight against DPEMA attacks too, so that the complete solution becomes resilient against both types of side channel attacks. The rest of the paper is focused on the DPEMA attacks, however the solution proposed and the results presented will include SPEMA attacks too.

3.1. Differential P or EM Radiation Analysis Attack

DPEMA is an even more powerful attack than SPEMA despite requiring a longer time to be completed. In the context of cache memory, this attack does not discover the whole scrambling vector at once but estimates it on a bit by bit basis [22].

Let us assume that the adversary attempts to estimate bit s_j of the scrambling vector. He first builds two subsets of profiling vectors (data vectors): subset $D(d_j = 0)$ contains all combinations of values at which data bit located at the same position, d_j, is constant at 0 and subset $D(d_j = 1)$, which is similar but now bit d_j is constant at 1. Then he applies the first subset $D(d_j = 0)$ in a continuous loop and estimates the average hamming weight $HW_{avg}(d_j = 0)$ of the scrambled data. He repeats the same action again with the second subset $D(d_j = 1)$ and estimates the average hamming weight $HW_{avg}(d_j = 1)$. Finally, he guesses bit s_j^* of S^* as follows:

$$s_j^* = \begin{cases} 0, if\ HW_{avg}(d_j = 0) < HW_{avg}(d_j = 1) \\ 1, if\ HW_{avg}(d_j = 0) > HW_{avg}(d_j = 1) \end{cases} \quad (1)$$

This operation is repeated for each bit and once all bits are estimated, the scrambling vector is built as the concatenation of:

$$S^* = (s_{n-1}^*, s_{n-2}^*, \ldots, s_1^*, s_0^*) \quad (2)$$

The strength of this attack lies in the fact that it is not necessary to apply all combinations of the rest of bits which are not kept constant. It is even better to change them randomly, thus reducing the amount of vectors significantly and obtaining a tighter estimate of the average hamming weight with very low noise. Furthermore, the hamming weight is not really necessary since the comparison in Eq. (1) can be made directly with the physical magnitudes $P_{DDavg}(d_j = 0)$ and $P_{DDavg}(d_j = 1)$ or $EM_{DDavg}(d_j = 0)$ and $EM_{DDavg}(d_j = 1)$ measured by the external instruments. One of the most dangerous points is that the physical magnitude can be averaged over thousands of samples, which would greatly reduce the effect of (accidental or intentional) noise and would strengthen the signal induced by the sought information.

3.2. Attack model

For the rest of the paper it is assumed that the adversary can measure current consumption using sensors attached to the power supply or measure electromagnetic power radiation by means of antenna probes. These can detect internal activity in the memory bus. He knows the model of the L2 cache and understands the IST process/operation. Moreover, he has control over some data vectors generated by the CPU, which allow him to estimate the scrambling vector in use. He cannot read cache memory sensitive data directly from the CPU, since it is assumed that the operating system blocks protected memory addresses when the critical program is in operation, therefore to do so the system must undergo a cold-boot attack such as those described above. The adversary cannot read cache content from the outside of the CPU because the former cannot be detached from the latter.

4. Proposed solution

This section presents the solution to protect the cache memory against cold-boot attacks boosted with DPEMA. The idea is to reduce the amount of information leaking from the system and to modify the correlations such that confusion prevents proper operation of the attack model, that is, to prevent the analysis with DPEMA from revealing the true scrambling vector. Since this solution is build over a previous countermeasure against SPEMA, proposed by the same authors in a conference paper [23], first this scheme is introduced and later the DPEMA countermeasure will be explained.

4.1. ISTe, SPEMA countermeasure

In [23] the model of the cache assumed includes an error detector and correction scheme [24] in which the IST is added and is the base of the SPEMA countermeasure. The general overview of the cache design is illustrated in Fig. 7.

The components of this model are the following: CPU which provides data (D^3), full adder blocks ($\sum$) that calculate carry and sum bits, scrambling vector (S^5) and XOR gates. The operation of this model is explained below. Symbol

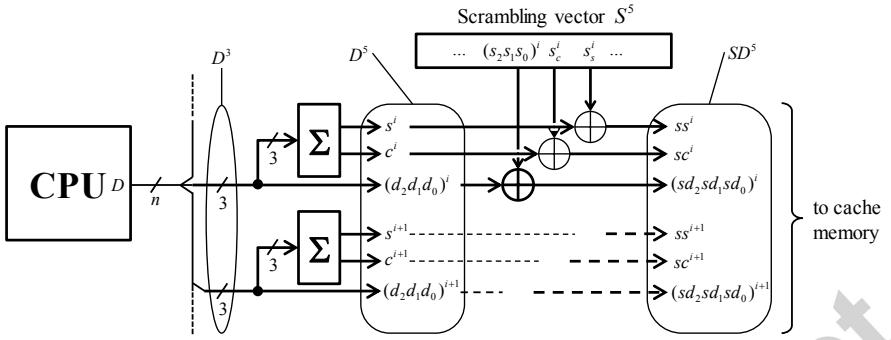


Figure 7: Model of the cache memory with error detection and correction scheme in which IST is applied (ISTe).

ISTe will distinguish this case from that in which the cache does not have an error detection and correction scheme.

In this particular scheme, data vector bits (D^3) are separated in groups of three bits to which two bits are appended corresponding to the sum and carry bits. Therefore, each data vector $D^3 = \{\ldots, (d_2, d_1, d_0)^i, \ldots\}$ is first appended with the error detection and correction bits and transformed to the new vector $D^5 = \{\ldots, (d_2, d_1, d_0)^i, c^i, s^i, \ldots\}$. Then the scrambling transformation is applied as in any regular scrambling technique, cf. Fig. 7. In correspondence, scrambling vector provides redundant bits $S^5 = \{\ldots, (s_2, s_1, s_0)^i, s_c^i, s_s^i, \ldots\}$ to protect data redundancy. These bits s_c^i, s_s^i are not randomly generated but calculated according to the following equations [23],

$$\begin{aligned} s_c &= not(carry(s_2, s_1, s_0)) = \overline{s_2 s_1 + s_2 s_0 + s_1 s_0} \\ s_s &= not(sum(s_2, s_1, s_0)) = \overline{s_2 \oplus s_1 \oplus s_0} \end{aligned} \tag{3}$$

while bits s_2, s_1, s_0 are generated randomly. Contrary to the intuition, generating s_c^i and s_s^i randomly would compromise the security. This last fact is illustrated in the results Section 5.

The SPEMA countermeasure shown in Fig. 7 is insecure against DPEMA attacks as it is illustrated in the following example.

4.2. Example of DPEMA attack on ISTe

For simplicity and without loss of generality, consider a bus of three bits, i.e. the three data bits plus two redundant bits as illustrated in Fig. 7. First the adversary prepares data vector sets for the attack, the zero vector set $D^5(d_j = 0)$ and the one vector set $D^5(d_j = 1)$ as shown in Tab. 1. It includes the sets for each one of the data bits.

The attack consists in estimating the average hamming weights for each one of these sets. The adversary activates the system and, for example, starts by applying the zero set $D^5(d_2 = 0)$ in a continuous loop such that he excites

Table 1: Zero and one data vector sets for the DPEMA attack in a three bit bus example.

$D^5(d_2 = 0)$

d_2	d_1	d_0	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

$D^5(d_1 = 0)$

d_2	d_1	d_0	c	s
0	**0**	0	0	0
0	**0**	1	0	1
1	**0**	0	0	1
1	**0**	1	1	0

$D^5(d_0 = 0)$

d_2	d_1	d_0	c	s
0	0	**0**	0	0
0	1	**0**	0	1
1	0	**0**	0	1
1	1	**0**	1	0

$D^5(d_2 = 1)$

d_2	d_1	d_0	c	s
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$D^5(d_1 = 1)$

d_2	d_1	d_0	c	s
0	**1**	0	0	1
0	**1**	1	1	0
1	**1**	0	1	0
1	**1**	1	1	1

$D^5(d_0 = 1)$

d_2	d_1	d_0	c	s
0	0	**1**	0	1
0	1	**1**	1	0
1	0	**1**	1	0
1	1	**1**	1	1

current consumption and EM radiation power at a stable rate. He calculates the average of these physical magnitudes and then estimates the average hamming weight $HW_{avg}(d_2 = 0)$. By repeating the same process with the one set, he obtains $HW_{avg}(d_2 = 1)$. Finally, with Eq. 1 he guesses bit s_2 of the scrambling vector as

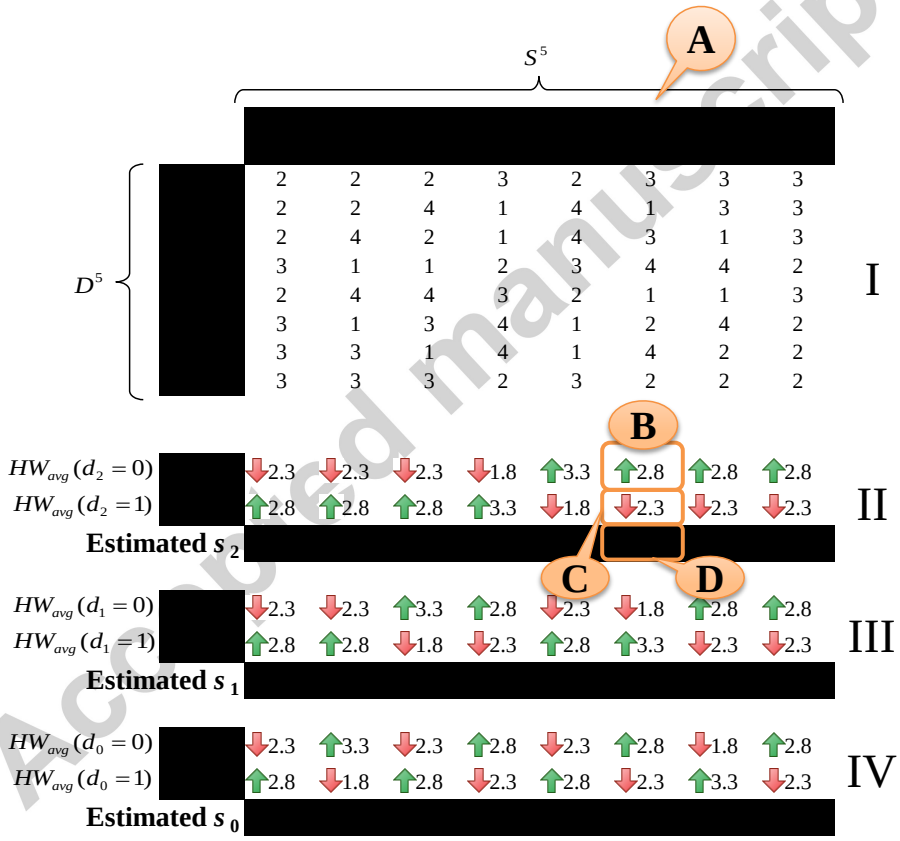
$$s_2^* = \begin{cases} 0, if HW_{avg}(d_2 = 0) < HW_{avg}(d_2 = 1) \\ ?, if HW_{avg}(d_2 = 0) = HW_{avg}(d_2 = 1) \\ 1, if HW_{avg}(d_2 = 0) > HW_{avg}(d_2 = 1) \end{cases}$$

Following the same methodology bits s_1 and s_0 are guessed too, it is unnecessary to do predictions for bits s_c and s_s because they are only part of the error correction scheme.

In Tab. 2 the complete calculations are shown. It is divided in four sub-tables: (I) has the matrix of hamming weights generated in the data bus after scrambling the data vector D^5 (shown at the left side of the matrix) with all possible scrambling vectors S^5 (at the top side of the matrix), (II) has the average hamming weights $HW_{avg}(d_2 = 0)$ and $HW_{avg}(d_2 = 1)$ generated after running the sets $D^5(d_2 = 0)$ and $D^5(d_2 = 1)$ respectively. From this averages the estimation of bit s_2 is made and shown at the botton of this sub-table. Sub-tables (III) and (IV) show the same information but for bits 1 and 0 of the data and scrambling vectors giving the estimations of s_1 and s_0 respectively.

Let's focus on one particular case. Assume that the internal scrambling vector is $S^5 = (101\ 01)$, cf. label **(A)** in the Table. When the CPU sends data vectors $D^5 = \{000\ 00, 001\ 01, ..., 111\ 11\}$ to the scrambling circuit, the hamming weights $HW = \{3, 1, 3, 4, 1, 2, 4, 2\}$ are excited in the data bus, see column under label **(A)**. Based on this fact, the adversary starts exciting the scrambling circuit with the zero data vector set $D^5(d_2 = 0)$ that will excite in the data bus only

Table 2: Example of DPEMA attack applied to a three bit data bus with ISTe scrambling technique. (I) is the matrix of hamming weights generated in the data bus after the scrambling. (II), (III) and (IV) are the average hamming weights used to estimate bit s_2, s_1 and s_0 respectively.



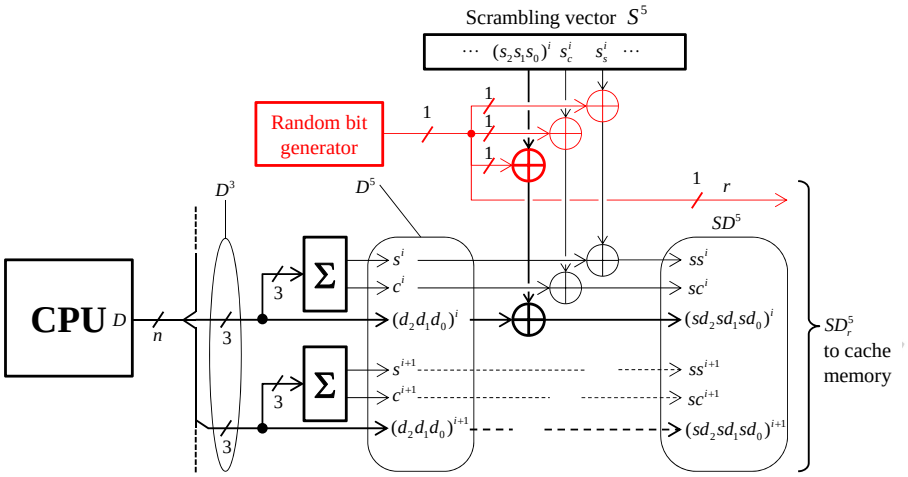


Figure 8: Overview of the RM-ISTe technique.

the hamming weights $\{3, 1, 3, 4\}$. After averaging them the adversary obtains the average hamming weight, $HW_{avg}(d_2 = 0) = 2.8$, cf. label **(B)**. Next, he excites the scrambling circuit with the one data vector set $D^5(d_2 = 1)$ that will excite in the data bus only the hamming weights $\{1, 2, 4, 2\}$ and obtains the new average hamming weight $HW_{avg}(d_2 = 1) = 2.3$ cf. label **(C)**. Finally, since $HW_{avg}(d_2 = 0) > HW_{avg}(d_2 = 1)$, bit s_2 is correctly estimated as $s_2^* = 1$, according to eq. 1, cf. label **(D)**.

4.3. DPEMA countermeasure

Random masking ISTe (RM-ISTe) is a strategy for balancing average hamming weights $HW_{avg}(d_j = 0)$ and $HW_{avg}(d_j = 1)$ to make them equal or randomly unequal in order to render Eq. (1) useless. The circuit schematic of this countermeasure is shown in Fig. 8 in which the red part shows the modifications to include this random masking.

A random bit generator providing bit r is added to the previous design. This bit changes randomly after each CPU write cycle. For $r = 0$, the scrambling vector is taken as it is whereas for $r = 1$ the scrambling vector is inverted before scrambling the data vector. Once the scrambled data is stored in the cache, bit r is appended to it. Therefore, the scrambled data vector which is sent to the cache memory is $SD_r^5 = (SD^5, r)$, where bit r equally affects all scrambled data bits.

To better understand how the countermeasure works, Fig. 9 presents the flowchart of the write cycle including an example on 9 bits. When a write cycle begins, the CPU generates the data vector D^3 that needs to be stored in the L2 cache. At the same time, a scrambling vector S^3 is retrieved from the scrambling table. The redundancy is calculated for the data vector (for every 3 bits of data, a carry and sum bit are appended) and D^5 is generated. Similarly

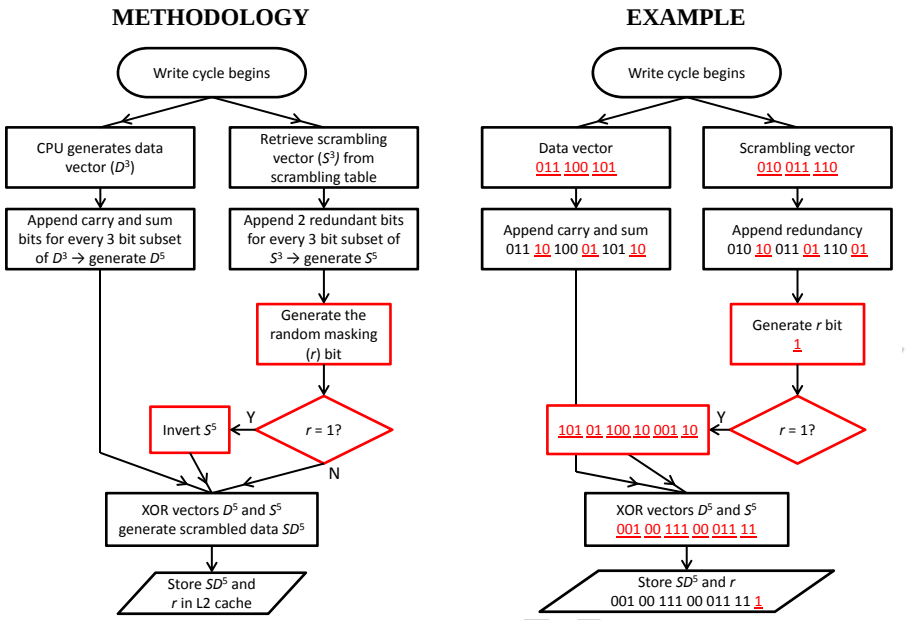


Figure 9: Architecture flowchart of the write cycle in the RM-ISTe technique. An example on 9 bits is included.

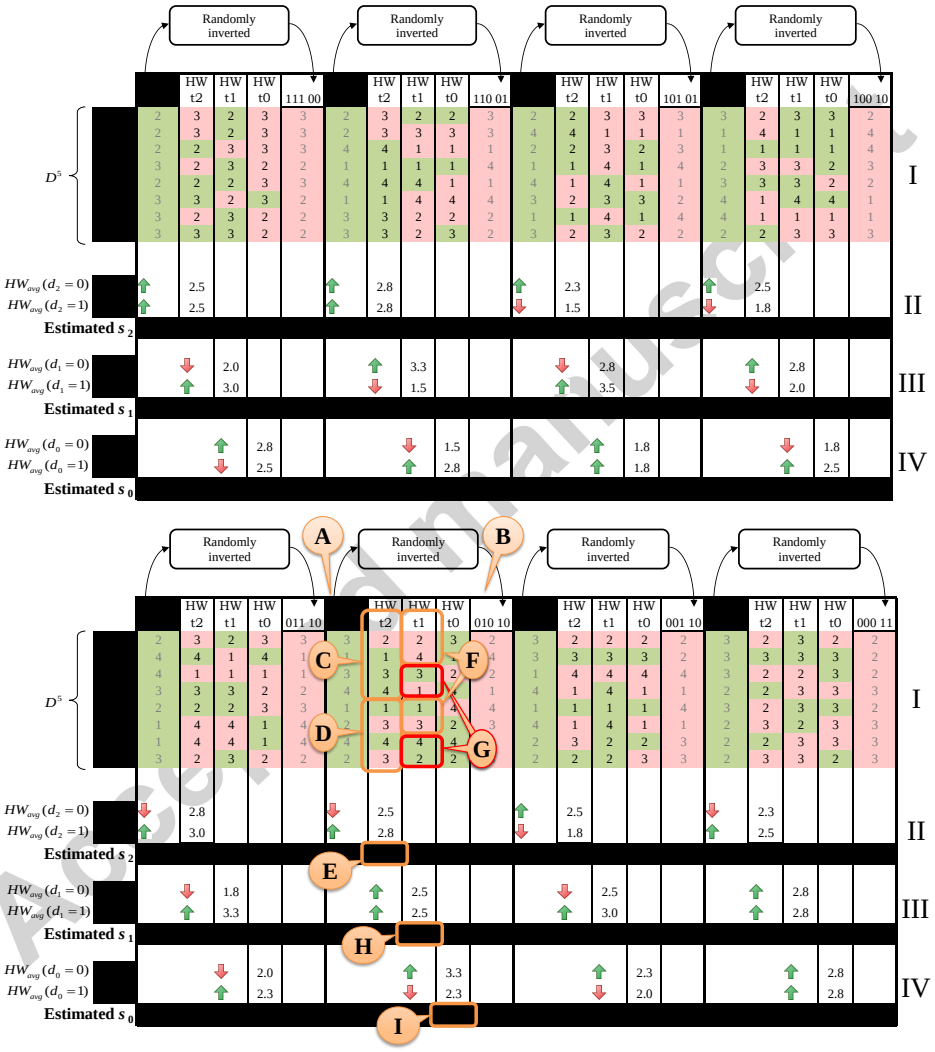
the redundancy for S^3 is computed (for every 3 bits, the 4th bit is the inverse of the carry, while the 5th bit is the inverse of the sum) and S^5 is generated. Then, the random masking bit r is obtained from the random generator and its value is checked. If it is 1, S^5 is inverted and XORed with D^5 generating the scrambled data SD^5 otherwise the inversion os S^5 is not performed. Finally, SD^5 is stored in the cache, together with the random masking bit r.

4.4. Example of DPEMA attack on RM-ISTe

Tab. 3 presents the same DPEMA attack example as in Tab. 2 but for the RM-ISTe technique. It exemplifies how the countermeasure works and as before it is made on a three bit data bus architecture. Notice that, since the scrambling technique is bit-wise, the conclusions of this example can be extended to any data bus size.

The Table is divided in the same four sub-tables (I)–(IV) as before. Sub-table (I) contains the matrix of the hamming weights generated in the data bus by the scrambling operations and sub-tables (II) to (IV) have the average hamming weights used by the adversary to estimated the scrambling vector bits. Unlike in the previous example (Tab. 2) now the top row of matrix (I) has not only all possible scrambling vectors S^5 for three bits but their inversions $\overline{S^5}$. This is so because each time the CPU writes, the data vector D^5 can be randomly scrambled with S^5 or $\overline{S^5}$. A second difference in matrix (I) is that the hamming weights are printed in three columns (HW / t2, HW / t1, HW

Table 3: DPEMA attack applied to RM-ISTe countermeasure.



/ t0) for each scrambling vector S^5. Each of these columns indicate a random selection of the scrambling vector S^5 or $\overline{S^5}$ so at three different time instants ($t2$, $t1$ and $t0$) the hamming weight obtained after the scrambling operation can be different. The background color of each number indicates what scrambling vector has been used (green S^5) or (pink $\overline{S^5}$).

As before, we will examine the case for the scrambling vector $S^5 = (101\ 01)$, cf. label **(A)** whose inverted value is $\overline{S^5} = (010\ 10)$, cf. label **(B)**. Suppose that the adversary starts, at time instant $t2$, attacking bit s_2 and for this purpose he first excites once the scrambling circuit with the zero set $D^5(d_2 = 0)$. He captures the hamming weights $\{2, 1, 3, 4\}$, cf. label **(C)**, and obtains the average hamming weight $HW_{avg}(d_2 = 0) = 2.5$, where three out of four values correspond to the use of S^5 and one to the use of $\overline{S^5}$. Then, he applies the one set $D^5(d_2 = 1)$ and captures the new values for the hamming weights $\{1, 3, 4, 3\}$, cf. label **(D)**, and estimates the average hamming weight $HW_{avg}(d_2 = 1) = 2.8$. In this second estimation, two out of four hamming weights are generated by S^5 while the other two are generated by $\overline{S^5}$. He finally applies Eq. 1 and estimates $s_2^* = 0$, cf. label **(E)**, which becomes wrong because $s_2 = 1$.

Next, the adversary repeats the procedure to obtain bit s_1, but now when he applies the zero and one sets at time instant $t1$, he will obtain different selections for the S^5 and $\overline{S^5}$ scrambling vectors, cf. labels **(F)** and **(G)**. After calculating the averages, he will obtain two equal values, $HW_{avg}(d_1 = 0) = HW_{avg}(d_1 = 1) = 2.5$, and consequently it will not be possible to reliably estimate the corresponding scrambling vector bit $s_1^* =?$, cf. label **(H)**. Finally, by repeating the process at the time step $t0$ for the last bit, the estimation obtained is $s_0^* = 1$, which in this case is correct by chance, cf. label **(I)**.

5. Evaluation and results

The proposed technique RM-ISTe is evaluated on a virtual implementation of the scrambled cache memory where several switches configure different kinds of countermeasures. The experiments are compared to the previous techniques IST and ISTe [21, 23].

The evaluation of the security is done by calculating the information that leaks through the hamming weight. The leakage function $L(s)$ which is based on information entropy [25], evaluates how much close are the estimated scrambling bits s_j^* from the real ones s_j. This function is bounded in the interval $0 \leq L(s) \leq 1$ in which 0 means an always wrong estimation while 1 corresponds to always correct estimation. The derivation of this function can be found in Appendix A.

An ad hoc simulation environment is programmed in C++ which includes the control of the data bus, the scrambling circuit and the cache memory. A wrapper emulates the behavior of the attack in the virtual environment and virtual instruments monitor the performance of the different countermeasures. The wrapper also gives us full control over the generation of scrambling vectors and observation of scrambled data. This environment runs on a Supermicro

workstation with the following specifications: 64 AMD cores of 64 bits, 256 GB of memory, 6 TB of disc space and with CentOS Linux operating system.

The virtual environment allows the modification of architecture size. Authors have full control over the data bus through which bursts of data vectors can be repeatedly sent to carry out attacks. An internal monitor measures the hamming weight generated during data transfer to the cache, including transformations made by the scrambling circuit. Monitoring the hamming weight as a direct metric for the attack is a conservative way to evaluate the strength of countermeasures because it is an upper bound of the physical measures that the adversary would achieve by the observation of power consumption or EM radiation intensity, as has been explained in previous Section 3. Furthermore, periodical refreshes of scrambling vectors that are made by IST and the derived techniques in order to limit the effectiveness of attacks is disabled. In these experiments the same scrambling vector is kept in use throughout the attack because we are more interested in evaluation of the random masking scheme strength than in the whole scheme effectiveness. Therefore results must be understood as an upper bound of the attack success.

The emulated configurations are:

- S3 - IST scrambling technique [21].
- S5R5 - ISTe″ SPEMA countermeasure, but where the two redundant bits of the scrambling vector s_c^i, s_s^i are generated randomly. This configuration is presented to illustrate the comment made in Subsection 4.1 that full random generation of the redundancy decreases the security instead of increasing it.
- S5R4C1 - ISTe′ SPEMA countermeasure, but where bit s_c^i is generated randomly while s_s^i is calculated according Eq. 3. This configuration is a middle step between S5R5 and S5R3C2.
- S5R3C2 - ISTe SPEMA countermeasure [23].
- S3M - same as S3 but with random masking RM-IST. No figure illustrates explicitly this configuration but consider Fig. 8, where bits $\{\ldots, c^i, s^i, \ldots\}$ are not added to the data vector and bits $\{\ldots, s_c^i, s_s^i, \ldots\}$ are not generated for the scrambling vector. However, a random bit generator flips the content of the scrambling vector randomly and bit r is stored in the cache together with scrambled data SD.
- S5R5M - Same as S5R5 but with random masking RM-ISTe″, Fig. 8.
- S5R4C1M - Same as S5R4C1 but with random masking RM-ISTe′, Fig. 8.
- S5R3C2M - Same as S5R3C2 but with random masking, Fig. 8. This is the full RM-ISTe proposed in this paper for defense against SPEMA and DPEMA attacks which renders the minimum leakage.

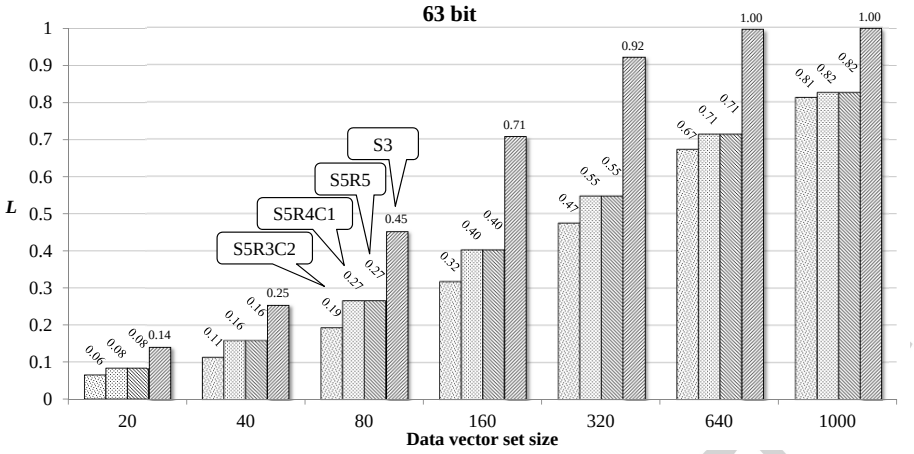


Figure 10: Information leakage achieved by DPEMA attacks applied on techniques without random masking for a 63-bit architecture.

5.1. DPEMA attacks in non-random-masking techniques

DPEMA attack results are first presented for an architecture of 63 bits. Data vector sets $D^{3/5}(d_i = 0/1)$ used during the attacks are of lengths from 20 to 1000 vectors. The attacks are repeated for 1000 different scrambling vectors and the leakage function $L(s)$ is averaged between them. Fig. 10 plots the results for the techniques without random masking, {S3, S5R5, S5R4C1, S5R3C2}.

The x-axis shows the number of vectors in the attacking data set and the y-axis gives the values of the leakage metric $L(s)$. The low effectiveness of the techniques without random masking against DPEMA is worth noting. The left column of each group shows that leakage $L(s)$ increases from 0.06 to 0.81 for the most effective technique in this group, S5R3C2, when the data vector set is changed from 20 to 1000 vectors. For the leakage level of 0.81 the number of correctly estimated scrambling vectors is 131 over 1000. The other techniques exhibit a similar trend but with higher leakage levels, giving in the worst case 1 which means that all the estimations were correct. In particular the worse behavior of S5R4C1 and S5R5 is caused by the random generation of the redundancy in the scrambling vectors. The case S3 without SPEMA countermeasure is the one that also presents the poorest results under DPEMA attacks.

Another significant trend observed is that the leakage increases when the attacking data vector set enlarge, independently of the technique used. Notice that for a set of 20 the leakages of the four techniques are $\{0.06, 0.08, 0.08, 0.14\}$ while for a set of 1000 the leakages are $\{0.81, 0.82, 0.82, 1\}$. This is an important information for the adversary because he knows that for a large enough set he can break the system completely independently of the internal countermeasure.

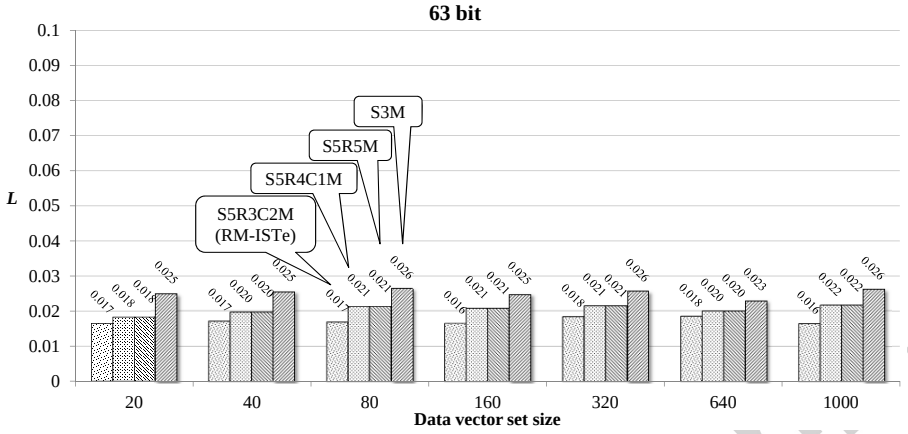


Figure 11: Information leakage achieved with DPEMA attacks applied on random masking techniques for 63-bit architecture size.

5.2. DPEMA attacks in random-masking techniques

Experiments of Fig. 10 are repeated but with the techniques using random masking {S3M, S5R5M, S5R4C1M, S5R3C2M}, results are plotted in Fig. 11.

For the attacking set of 1000 vectors, a significant decrease of the leakage from 0.81 to 0.016 is observed for S5R3C2M (the proposed RM-ISTe). It is remarkable that, among all predictions none of the scrambling vectors are correct. It is also worth noting that the leakage is approximately the same for all data vector set sizes without following any expected trend. This is particularly interesting because one of the procedures that an adversary can use to drive the attack is to predict the variation of the leakage vs. the number of data vectors applied. In the techniques using random masking, the above prediction does not provide any useful feedback. Finally, the combination of techniques designed against SPEMA attacks in [23] provide an additional degree of protection when used with random masking. Compare for the case of 1000 data vectors the case wihout SPEMA countermeasure (S3M) to the case with SPEMA countermeasure (S5R3C2M), which give leakages of 0.026 and 0.016 respectively.

5.3. DPEMA attacks for different architecture sizes

Tab. 4 summarizes the results for several architecture sizes going from 9 to 63 bits, for attacks carried out with data vector sets of 1000 vectors. All the numbers are averaged for 1000 attacks using different scrambling vectors.

In the table, first column (Arch. size) indicates the size of the bus. Second column (S3 / IST) lists the leakage for all architectures using the S3 technique, which gives a leakage of 1 in all cases. The next three columns (S5R5 / ISTe$''$, S5R4C1 / ISTe$'$, S5R3C2 / ISTe) correspond to the techniques without random masking but including a SPEMA countermeasure. In this group, a maximum leakage of 1 is found for all techniques in the smallest architectures while minimum leakages are 0.815, 0.815 and 0.813 respectively for the largest architecture,

Table 4: Information leakage measured for DPEMA attacks in different architecture sizes and techniques. Attacking data vector set is 1000. Number of scrambling vectors tested is 1000. In grey background cells at least one scrambling vector has been correctly estimated.

Arch. size	S3 IST	S5R5 ISTe''		S5R4C1 ISTe'		S5R3C2 ISTe		S3M RM-IST		S5R5M RM-ISTe''		S5R4C1M RM-ISTe'		S5R3C2M RM-ISTe	
9	1	1.000	0.0%	1.000	0.0%	1.000	0.0%	0.198	80.2%	0.142	85.8%	0.142	85.8%	0.110	**89.0%**
12	1	0.999	0.1%	0.999	0.1%	1.000	0.0%	0.147	85.3%	0.108	89.2%	0.108	89.2%	0.088	**91.2%**
15	1	0.997	0.3%	0.997	0.3%	0.999	0.1%	0.121	87.9%	0.087	91.3%	0.087	91.3%	0.065	**93.5%**
18	1	0.993	0.7%	0.993	0.7%	0.997	0.3%	0.090	91.0%	0.070	93.0%	0.070	93.0%	0.054	**94.6%**
21	1	0.990	1.0%	0.990	1.0%	0.993	0.7%	0.080	92.0%	0.057	94.3%	0.057	94.3%	0.050	**95.0%**
24	1	0.984	1.6%	0.984	1.6%	0.986	1.4%	0.064	93.6%	0.052	94.8%	0.052	94.8%	0.046	**95.4%**
27	1	0.973	2.7%	0.973	2.7%	0.981	1.9%	0.055	94.5%	0.047	95.3%	0.047	95.3%	0.036	**96.4%**
30	1	0.963	3.7%	0.963	3.7%	0.974	2.6%	0.059	94.1%	0.043	95.7%	0.043	95.7%	0.037	**96.3%**
33	1	0.952	4.8%	0.952	4.8%	0.957	4.3%	0.050	95.0%	0.039	96.1%	0.039	96.1%	0.033	**96.7%**
36	1	0.930	7.0%	0.930	7.0%	0.945	5.5%	0.048	95.2%	0.034	96.6%	0.034	96.6%	0.033	**96.7%**
39	1	0.928	7.2%	0.928	7.2%	0.928	7.2%	0.045	95.5%	0.033	96.7%	0.033	96.7%	0.029	**97.1%**
42	1	0.914	8.6%	0.914	8.6%	0.921	7.9%	0.039	96.1%	0.031	96.9%	0.031	96.9%	0.025	**97.5%**
45	1	0.895	10.5%	0.895	10.5%	0.903	9.7%	0.037	96.3%	0.027	97.3%	0.027	97.3%	0.025	**97.5%**
48	1	0.883	11.7%	0.883	11.7%	0.889	11.1%	0.035	96.5%	0.027	97.3%	0.027	97.3%	0.023	**97.7%**
51	1	0.872	12.8%	0.872	12.8%	0.883	11.7%	0.033	96.7%	0.024	97.6%	0.024	97.6%	0.021	**97.9%**
54	1	0.850	15.0%	0.850	15.0%	0.859	14.1%	0.032	96.8%	0.024	97.6%	0.024	97.6%	0.021	**97.9%**
57	1	0.838	16.2%	0.838	16.2%	0.845	15.5%	0.029	97.1%	0.023	97.7%	0.023	97.7%	0.021	**97.9%**
60	1	0.828	17.2%	0.828	17.2%	0.821	17.9%	0.028	97.2%	0.023	97.7%	0.023	97.7%	0.018	**98.2%**
63	1	0.815	18.5%	0.815	18.5%	0.813	18.7%	0.026	97.4%	0.021	97.9%	0.021	97.9%	0.018	**98.2%**

representing a reduction of 18.7%. In all the cases at least one scrambling vector has been estimated correctly, this is shown painting the background in grey shadow.

The following four columns (S3M / RM-IST, S5R5M / RM-ISTe″, S5R4C1M / RM-ISTe′, S5R3C2M / RM-ISTe) correspond to the techniques with random masking and therefore all of them are DPEMA countermeasures. Except the first, the rest also include a SPEMA countermeasure. The maximum leakage is found for the smallest architecture size with 0.198 for (S3M / RM-IST), representing a reduction of 80.2% in leakage while the minimum leakage is found in (S5R3C2M / RM-ISTe) for the biggest architecture size with 0.018 that represents a reduction of 98.2%. In all four cases none of the scrambling vectors are correctly estimated for architecture sizes larger than 18 bits, this is shown with the white background of the cells.

5.4. Implementation costs

To evaluate the implementation costs we have followed the same strategy as in [26] and [27] which consists in predicting them with the CACTI tool [28]. This tool generates cost predictions for cache memory architectures that can be tuned for several parameters including line size, associativity, number of banks, technology nodes, etc. It allows to do space exploration of different alternatives during the design phase.

We consider the following methodology. The particular logic implementation whose costs needs to be evaluated is split in sub-blocks whose architecture needs to resemble as close as possible to a cache memory. Then, different cache memories are dimensioned according to the sub-block parameters and technology and their cost predictions obtained with the CACTI. Finally an artifact is created to combine the predictions of the sub-blocks following the rules of the global design from which the final cost predictions are generated.

Table 5: Implementation costs of the previous and current proposed techniques.

	Size (KB)		Area occupied (mm^2)		Power consumption (mW)		Access time (ns)	
L2 cache	16.00		0.1479		94.50		0.3651	
└—IST	4.00	20%	0.0684	32%	74.50	44%	0.3227	47%
└—ISTe	17.20	52%	0.1573	52%	96.80	51%	0.3701	50%
└—RM-IST	4.50	22%	0.0706	32%	75.32	44%	0.3274	47%
└—RM-ISTe	**17.70**	**53%**	**0.1602**	**52%**	**97.82**	**51%**	**0.3727**	**51%**
L2 cache	32.00		0.2394		97.30		0.4019	
└—IST	5.65	15%	0.0486	17%	50.50	34%	0.3113	44%
└—ISTe	30.50	49%	0.2352	50%	96.67	50%	0.3973	50%
└—RM-IST	6.65	17%	0.0885	27%	83.96	46%	0.3347	45%
└—RM-ISTe	**31.50**	**50%**	**0.2401**	**50%**	**97.90**	**50%**	**0.3995**	**50%**
L2 cache	64.00		0.5493		159.40		0.5015	
└—IST	8.00	11%	0.0691	11%	57.90	27%	0.3128	38%
└—ISTe	55.52	46%	0.5023	48%	149.70	48%	0.4853	49%
└—RM-IST	10.00	14%	0.1148	17%	81.93	34%	0.3520	41%
└—RM-ISTe	**57.52**	**47%**	**0.5136**	**48%**	**151.97**	**49%**	**0.4889**	**49%**
L2 cache	128.00		1.1452		280.20		0.6238	
└—IST	11.31	8%	0.1222	10%	84.40	23%	0.3589	37%
└—ISTe	103.25	45%	0.7692	40%	202.20	42%	0.5759	48%
└—RM-IST	15.31	11%	0.1455	11%	92.49	25%	0.3605	37%
└—RM-ISTe	**107.25**	**46%**	**0.7917**	**41%**	**206.47**	**42%**	**0.5839**	**48%**

In our particular case all the scrambling techniques that we present are based on the IST (Interleave Scrambling Technique) architecture [21]. This technology consists of two main blocks, the L2 cache memory itself and the *scrambling vector table* which contains sets of scrambling vectors that are selected according to certain replacement rules and additional auxiliary registers and flags. The other three main sub-blocks are the redundancy generator and checking code modules, the scrambling circuit and the random generator. The extraction of parameters and artifacts for the cost estimation are as follows:

- L2 cache memory – In all IST versions each cache line needs one extra flags. In RM-IST an additional flag is included per word. In ISTe words are extended with data redundancy.

- *Scrambling vector table* – In IST the size of this table grows as the square root of the L2 cache memory size according to the rules of [21]. In the ISTe cases the scrambling vectors (two per line) of the table are extended with the redundancy. With respect to the effect of the delay (access time) of the cache emulating this table, it is added to the L2 cache memory time as a worst case scenario. With respect to the area and power they are added too.

- Redundancy generator and checking code modules, and scrambling circuit – They grow linearly with the size of the data bus so it is assumed that

the memory cache ports (L2 cache and *scrambling vector table*) properly emulate the cost overhead of these three elements too.

- Random generator – we do not consider the cost of the random generator because it is constant and independent of the architecture size. If implemented as a pseudo-random generator its impact is negligible with respect to the other sub-blocks.

Results are presented in Tab. 5. Column (Size (KB)) contains the capacity of the base L2 cache memory and the equivalent size extension of the cache necessary to implement each one of the scrambling techniques. At the right side of each number the increment in percentage is shown with respect to the base L2 cache memory. Four base L2 cache memory sizes have been considered: 16, 32, 64 and 128 KB and in each one of them the four scrambling techniques are implemented {IST, ISTe, RM-IST, RM-ISTe}.

In the rest of columns three costs are shown: (Area occupied), (Power consumption) and (Access time). All of them are obtained for a technology node of 45 nm. It is remarkable to see that the overheads decrease for larger cache memory sizes, which is caused by the slower increase of the *scrambling vector table* as indicated above. In particular, for a 128 KB L2 cache size and the proposed technique (RM-ISTe) the costs are: 41% area overhead, 42% power overhead and 48% access time overhead.

6. Conclusions

In this paper cold-boot attacks on cache memories boosted by differential power and electromagnetic analysis (DPEMA) are considered. While it is known that scrambling techniques, like the Interleaved Scrambling Technique (IST) can be effective against cold-boot attacks it is demonstrated that a DPEMA can be used to discover the internal scrambling vector and consequently to make the cold-boot attack effective, thus breaking the security of IST.

In this paper a new strategy is presented that can be added to the IST making this robust against DPEMA. It is named random masking (RM) and a complete solution is presented RM-ISTe which becomes effective against cold-boot attacks boosted by SPEMA (static) and DPEMA (dynamic) analysis aiming to discover the internal scrambling vector. Several examples illustrate the operation of the methodology proposed and experiments are presented to evaluate its effectiveness for different architecture sizes. It is seen that the leakage emanating from the power or electromagnetic radiation is reduced to a 98.2% with respect to the plain IST technique. The cost of the implementation considering a technology node of 45 nm, for a cache memory size of 128 KB is: 41% for the area overhead, 42% for the power consumption overhead and 48% for the access time overhead, respectively.

References

[1] M. J.P., Payments fraud and control survey (2015).

[2] F. Paget, Financial fraud and internet banking: threats and countermeasures (2009).

[3] D. R. Piegdon, L. Pimenidis, Hacking in physically addressable memory, in: Seminar of Advanced Exploitation Techniques, WS 2006/2007, Vol. 12, 2007.

[4] K. Harrison, S. Xu, Protecting cryptographic keys from memory disclosure attacks, in: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), IEEE, 2007, pp. 137–143.

[5] S. Skorobogatov, Low temperature data remanence in static ram (2002).

[6] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten, Lest we remember, Communications of the ACM 52 (5) (2009) 91. doi:10.1145/1506409.1506429.

[7] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. de Lara, H. Raj, S. Saroiu, A. Wolman, Protecting data on smartphones and tablets from memory attacks, ACM SIGPLAN Notices 50 (4) (2015) 177–189.

[8] M. Gruhn, T. Müller, On the practicability of cold boot attacks, in: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, IEEE, 2013, pp. 390–397.

[9] T. Müller, M. Spreitzenbarth, Frost: Forensic recovery of scrambled telephones, in: Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 373–388. doi:10.1007/978-3-642-38980-1_23.

[10] A. Cui, M. Costello, S. J. Stolfo, When firmware modifications attack: A case study of embedded exploitation., in: NDSS, 2013.

[11] V. Rijmen, J. Daemen, Advanced encryption standard, Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology (2001) 19–22.

[12] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, et al., Prince–a low-latency block cipher for pervasive computing applications, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2012, pp. 208–225.

[13] L. Su, A. Martinez, P. Guillemin, S. Cerdan, R. Pacalet, Hardware mechanism and performance evaluation of hierarchical page-based memory bus protection, in: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2009.

[14] W. Enck, K. Butler, T. Richardson, P. McDaniel, A. Smith, Defending against attacks on main memory persistence, in: 2008 Annual Computer Security Applications Conference (ACSAC), Institute of Electrical & Electronics Engineers (IEEE), 2008. doi:10.1109/acsac.2008.45.

[15] S. Chhabra, Y. Solihin, i-NVMM, Vol. 39, Association for Computing Machinery (ACM), 2011. doi:10.1145/2024723.2000086.

[16] I. Anati, J. Doweck, G. Gerzon, S. Gueron, M. Maor, A tweakable encrypion mode for memory encryption with protection against replay attacks, wO Patent App. PCT/US2011/053,170 (2012).
URL http://www.google.com/patents/WO2012040679A3?cl=en

[17] S. Gueron, U. Savagaonkar, F. Mckeen, C. Rozas, D. Durham, J. Doweck, O. MULLA, I. Anati, Z. Greenfield, M. Maor, Method and apparatus for memory encryption with integrity check and protection against replay attacks, wO Patent App. PCT/US2011/042,413 (2013).
URL http://www.google.com/patents/WO2013002789A1?cl=pt-PT

[18] B. Dolgunov, A. Aharonov, Memory randomization for protection against side channel attacks, uS Patent 8,726,040 (2014).

[19] R. V. Sai, S. Saravanan, V. Anandkumar, Implementation of a novel data scrambling based security measure in memories for vlsi circuits, Vol. 8, 2015.

[20] Intel, 5th generation intel core processor family, intel core m processor family, mobile intel pentium processor family, and mobile intel celeron processor family (2015).

[21] M.-I. Neagu, L. Miclea, S. Manich, Improving security in cache memory by power efficient scrambling technique, IET Computers & Digital Techniques 9 (6) (2015) 283–292. doi:10.1049/iet-cdt.2014.0030.

[22] P. Kocher, J. Jaffe, B. Jun, P. Rohatgi, Introduction to differential power analysis, Journal of Cryptographic Engineering 1 (1) (2011) 5–27. doi:10.1007/s13389-011-0006-y.

[23] M. Neagu, L. Miclea, S. Manich, Defeating simple power analysis attacks in cache memories, in: 2015 Conference on Design of Circuits and Integrated Systems (DCIS), Institute of Electrical & Electronics Engineers (IEEE), 2015. doi:10.1109/dcis.2015.7388557.

[24] N. Madalin, L. Miclea, J. Figueras, Unidirectional error detection, localization and correction for DRAMs: Application to on-line DRAM repair strategies, in: 2011 IEEE 17th International On-Line Testing Symposium, Institute of Electrical & Electronics Engineers (IEEE), 2011. doi:10.1109/iolts.2011.5994540.

[25] R. M. Fano, The transmission of information, Massachusetts Institute of Technology, Research Laboratory of Electronics, 1949.

[26] J. Liu, B. Jaiyen, R. Veras, O. Mutlu, Raidr: Retention-aware intelligent dram refresh, in: ACM SIGARCH Computer Architecture News, Vol. 40, IEEE Computer Society, 2012, pp. 1–12.

[27] P. Papavramidou, M. Nicolaidis, Reducing power dissipation in memory repair for high defect densities, in: 2013 18th IEEE European Test Symposium (ETS), IEEE, 2013, pp. 1–7.

[28] Cacti tool v5.3.
URL http://quid.hpl.hp.com:9081/cacti/

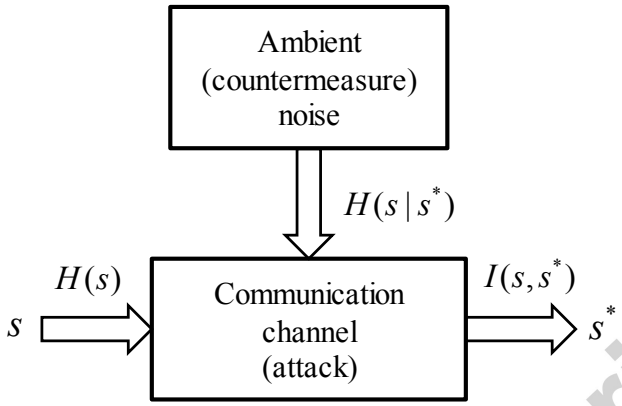


Figure A.12: Communication channel model used to evaluate the leakage.

Appendix A. Leakage function

In this paper by leakage it is meant the amount of information extracted from the scrambling vector to produce an accurate estimate of it. Information entropy is a metric for the measurement of the amount of information carried by a set of symbols and is commonly used as an indication of leakage in secure systems. The evaluation of the leakage exploited by an adversary during an attack can be modeled as the communication channel shown in Fig. A.12.

Assume that s is a scrambling vector bit and s^* is its estimate obtained by the attack. The attack can be viewed as a communication channel through which two types of information are sent: signal values (scrambling vector bits) and noise produced by the countermeasures in use. The ability of the attack to separate the signal from the noise will determine the degree of leakage achieved. According to the definitions of communication channels in [25], entropy $H(s)$ is the amount of information contained in scrambling vector bits. Conditional entropy $H(s|s^*)$ is the information responsible for errors in the communication channel. In our problem, it represents the noise injected by the countermeasures, which make estimates of s^* more or less correlated with s. Finally, $I(s, s^*)$ in the output channel is the entropy of the bit ensemble s, s^* and represents the amount of information from the input channel s that reaches the output channel s^*. This entropy represents the amount of leakage achieved by the attack and will be renamed as leakage function $L(s)$ or simply leakage. According to [25], this leakage function is evaluated in the channel as

$$L(s) = I(s, s^*) = H(s) - H(s|s^*) \tag{A.1}$$

which represents the balance of information in the channel. The evaluation of entropies $H()$ proceeds as follows.

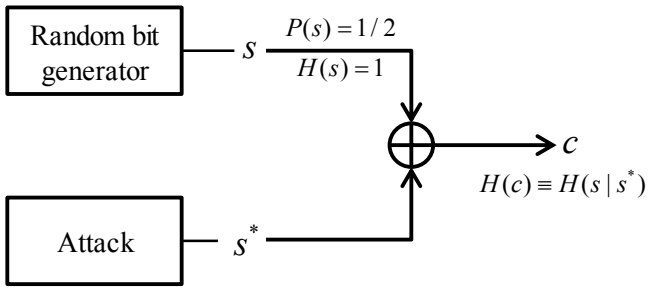


Figure A.13: Evaluation of entropies in an attack scenario.

Scrambling vector bits s are obtained from a random source. Hence, it can be assumed that their probability is $P(s) = P(s = 1) = 1/2$. Since entropy is the average of all information (log function) contained in binary symbols, we have that $H(s) = -[P(s) \cdot log_2 P(s) + P(\overline{s}) \cdot log_2 P(\overline{s})]$, where base 2 log is used, and thus $H(s) = 1$, Fig. A.13. This indicates that scrambling vectors are generated with the maximum amount of information possible, and accordingly have the highest uncertainty.

Conditional entropy $H(s \mid s^*)$ is calculated from the bit probability using the following equation:

$$H(s \mid s^*) = - \sum_{\substack{i=s,\overline{s} \\ j=s^*,\overline{s^*}}} P(i,j) \cdot log_2 P(i \mid j) \tag{A.2}$$

where $P(i,j)$ and $P(i \mid j)$ are the joint and conditional probabilities, respectively. These probabilities can be assessed in real experiments using the XOR logic function, cf. Fig. A.13. If the probability in one of the inputs of the XOR is $P(s) = 1/2$, then the following symmetries are observed in these probabilities:

$$\begin{aligned} P(\overline{s}, \overline{s}^*) &= P(s, s^*) = P(\overline{c})/2 \\ P(\overline{s}, s^*) &= P(s, \overline{s}^*) = P(c)/2 \\ P(\overline{s} \mid \overline{s}^*) &= P(s \mid s^*) = P(\overline{c}) \\ P(\overline{s} \mid s^*) &= P(s \mid \overline{s}^*) = P(c) \end{aligned} \tag{A.3}$$

whose substitution in Eq. A.2 results in output c entropy of the XOR:

$$H(s \mid s^*) = H(c) = -[P(c) \cdot log_2 P(c) + P(\overline{c}) \cdot log_2 P(\overline{c})] \tag{A.4}$$

By considering Eqs. A.1 and A.4 and taking the boundary conditions in Fig. A.13, the final expression for the leakage function is

$$L(s) = 1 - H(c) \tag{A.5}$$

For an open system, the evaluation of the leakage function starts by collecting the individual bits of the scrambling vector S and estimated scrambling

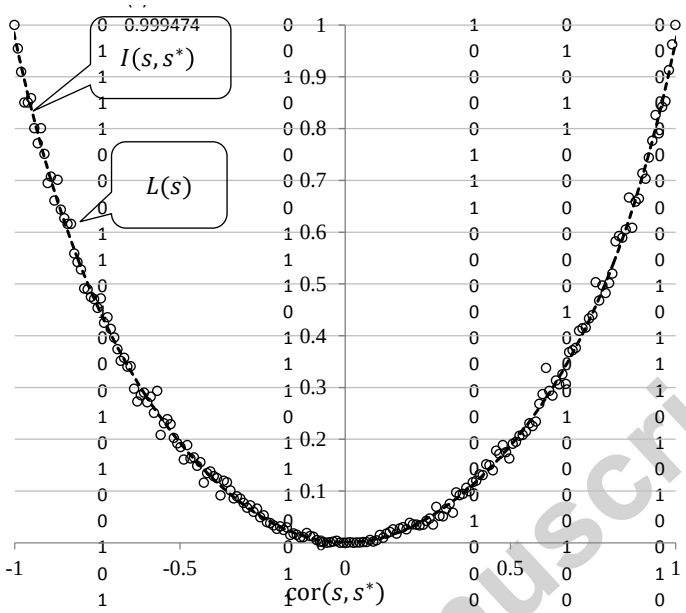


Figure A.14: Simulation of the circuit in Fig. A.13 and Eq. A.5, and theoretical curve derived from Eq. A.1.

vector S^*. Since the attack will be repeated many times using different (usually random) data vectors, after each one the result of the comparison vector C must be generated, probability $P(c)$ is estimated from its bits and then the leakage function is evaluated. To improve the accuracy of the estimate, several attacks can be performed keeping the same scrambling vector S and then all the bits of the set of vectors C are collected together to estimate the leakage function. Finally, if the strength of the countermeasure needs to be tightly assessed, then the same procedure must be applied but now including some experiments where the scrambling vector is changed too.

In order to illustrate Eq. A.5, a short Monte-Carlo simulation was made for the circuit in Fig. A.13. A total of 2000 random bits were generated for each s and s^* pair and the correlation between them in the [-1,1] interval was modified. Fig. A.14 summarizes the results of Eq. A.5, including 200 simulations and the theoretical curve derived from Eq. A.1. It is interesting to see that the maximum amount of information is leaked by predicting not only the same bit $s = s^*(cor(s, s^*) = 1)$ but also the inverted one $s = s^*(cor(s, s^*) = -1)$.