On the Use of Error Detecting and Correcting Codes to Boost Security in Caches against Side Channel Attacks

Mădălin Neagu, Liviu Miclea
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Madalin.Neagu@cs.utcluj.ro
Liviu.Miclea@aut.utcluj.ro

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

***Abstract*— Microprocessor memory is sensitive to cold boot attacks. In this kind of attacks, memory remanence is exploited to download its content after the microprocessor has been struck by a hard boot. If just in this moment, a crypto-algorithm was in execution, the memory data can be downloaded into a backup memory and specialized tools can be used to extract the secret keys.**

In the main memory data can be protected using efficient encryption techniques but in caches this is not possible unless the performance becomes seriously degraded. Recently, an interleaved scrambling technique (IST) was presented to improve the security of caches against cold boot attacks. While IST is effective for this particular kind of attacks, a weakness exists against side channel attacks, in particular using power analysis.

Reliability of data in caches is warranted by means of error detecting and correcting codes. In this work it is shown how these kinds of codes can be used not only to improve reliability but also the security of data. In particular, a self-healing technique is selected to make the IST technique robust against side channel attacks using power analysis.

Keywords*— *data scrambling, cache memories, cold boot attack, self-healing memories, side channel attack.

I. INTRODUCTION

Modern day ICs contain significant information that must be highly secured, so that it stays hidden. In the last six years, there have been reported a large variety of attacks against ICs and memories, mostly targeting the vulnerabilities of cache and main memories, smartcards, etc. This problem broadens when the attacks target credit cards and other kind of legal supports including biometric information [1]. Desktop computers are increasingly being used for secure purposes. Operating systems like Windows and MacOS include encryption tools for securing data in disk. In these two cases it has been seen that encryption/decryption keys are stored in memory as plain text and therefore that they can be downloaded successfully [2].

In systems aimed for security it is then necessary to protect the memory, especially the cache, because most of the time designers assume that it is intrinsically secure and therefore they forget to protect this, even with the most basic mechanisms. In this paper, we demonstrate that the cache memory can be secured against two kinds of attacks: power analysis side channel attacks and cold boot attacks, using a combination of error correction and data scrambling technique. In the next paragraphs the following points are reviewed concerning cache memories: different kind of side channel attacks, cold boot attacks and error detection/correction techniques.

Side-channels attacks on cache memories which are embedded in the CPU chips are becoming a serious threat when critical data is temporarily stored. These types of attacks rely on monitoring the behavior of the target in various ways. Time-based attacks observe the execution time of the victim when accessing cached and non-cached values and are somehow feared because they can be executed remotely. In access-based attacks, the opponent tests the cache state when corrupt data enter the memory. However, it is required that both, the attacker and victim, share the hardware platform. The last type of side-channel attacks are based on monitoring the sequence of cache hits and misses [2]. This is possible by measuring the power consumption. By exploiting the information leaked through side-channel attacks, sensitive data can be extracted from the cache memory. In this scope, the authors in [4] propose a tool for the static analysis of cache side-channel attacks. They focus on the effect of data leakage from software-based solutions and cache configurations.

In order to mitigate cache side-channel attacks, most of the existing proposals rely on controlling the information exchange between the hardware and software layers [5][6]. Cryptographic systems that run specific algorithms like the AES are widely targeted by side-channel attacks. In [7], the authors present an attack for the AES-128 algorithm that can recover the secret key and propose general countermeasures for these attacks.

Some modern CPUs have a shared L3 cache because the chip contains 2 or more cores. Because it is difficult to perform an attack on lower levels of the cache, the authors from [8] describe and execute an attack on the shared L3 cache. This attack is possible if the memory pages are shared between the spy program and the victim. The results indicate that it is possible to recover 98% of the bits of the private key, from a RSA encryption algorithm.

Cold boot attacks target cache and main memories and require physical access to the device. By freezing the memory chip after a sudden power-off, the data inside can still be accessed for a small period of time. Authors in [2] conduct a similar experiment and manage to recover private keys from encryption algorithms and applications. Note that because of

the remanence, data is accessible even after 5 minutes if the temperature of the device is low enough. The same cold boot attack was tested on Android smartphones in [9] where they propose a tool set for extracting disk encryption keys from RAM, contact lists and other sensible data. A recent methodology for securing cache memories against cold boot attacks was presented in [10] and is explained in detail in the next section because it is the base for the protection against side-channel attacks proposed in this paper.

As the title suggests, we wish to use the benefits of error detection and correction methodologies, in order to improve cache security. Usually, error detection and correction codes (EDC / ECC) are used in memories for mitigating soft errors. Thus, stored data that is corrupted by one or more errors can be reconstructed. This is possible due to the redundant information that is added to the original data. If errors occur in the data bits, the redundant bits help to detect errors and to recover the original data word. Transient faults are likely to cause symmetric soft errors or multiple unidirectional errors. In this scope, the authors in [11] present a comparative study on unidirectional error correction codes and propose a technique that chooses the suitable method, depending on the memory organization. A novel methodology that deals with unidirectional bit-flips in DRAMs was proposed in [12]. This method is explained in detail in the following section because is selected in this paper to leverage the interleaved scrambling technique [10].

In this work, we propose to combine the methodologies from [10] and [12]. In fact, it is seen that [10] is effective against cold boot attacks but that it could be circumvented using a side-channel attacks that would analyze the power consumption or the electromagnetic radiation emanating from the cache. To this end, the error detecting and correcting codes [12] are used to boost the security against this kind of attack.

The paper is organized as follows. Section II summarizes the previous techniques which are used in this work. Section III discusses the underlying problem of securing the cache memory from the side-channel attack perspective. Section IV describes the proposed solution and in Section V the technique is tested and evaluated from several points of view. Finally, in Section VI conclusions are drawn.

II. Review of Previous Techniques

In this paper the protection against power analysis side channel attack is achieved by combining an *error detection, localization and correction technique* (eDLC) [12] with the securing *interleaved scrambling technique* (IST) [10]. In this section these techniques are briefly explained to help clarifying the rest of the paper.

A. *Unidirectional error detection, localization and correction (eDLC)*

Permanent faults in memories are difficult to mitigate, but are barely encountered. On the other hand, transient faults that generate soft errors due to radiation are more likely to happen and the main causes are usually high-energy neutrons and alpha-particles. Dynamic RAMs are susceptible to radiation-type errors that manifest as unidirectional bits flips (due to the capacitor inside the DRAM cell). In [12], we proposed an error detection, correction and localization methodology for soft unidirectional errors that occur in DRAMs. The method is based on creating a full adder tree that generates redundant bits (carry and sum) at each subset of three data bits, during the write phase of the cache, see Fig. 1. The redundant bits count the number of 1's in the data subset. During the reading phase, the redundant bits are checked in the eDLC unit and in presence of an error they are used to localize and correct the data.

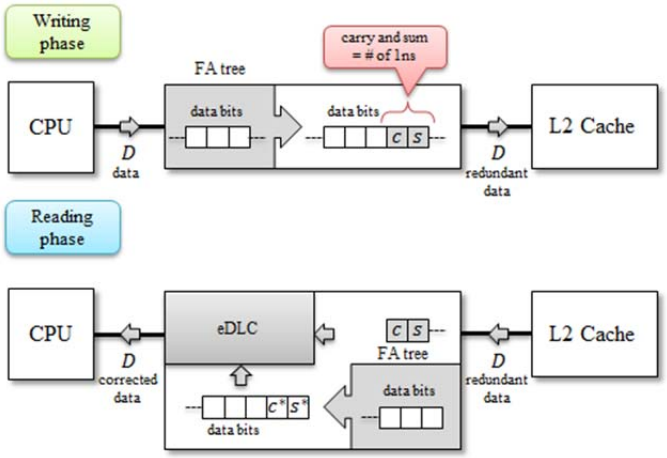

Fig. 1. The error detection, correction and localization technique (eDLC)

B. *Interleaved scrambling technique*

The interleaved scrambling technique from [10] increases the security of a cache memory by storing scrambled data in the cache. It protects the memory against cold boot attacks using the following principle. During operation the data, that is stored inside the cache, is first XORed with the keys of an internal table (scrambling table), see Fig. 2. When data are read back, they are first XORed with the same keys to undo the scrambling. During a cold boot attack, the scrambling table automatically resets with new keys so that when the read operation is attempted it will produce corrupted data.

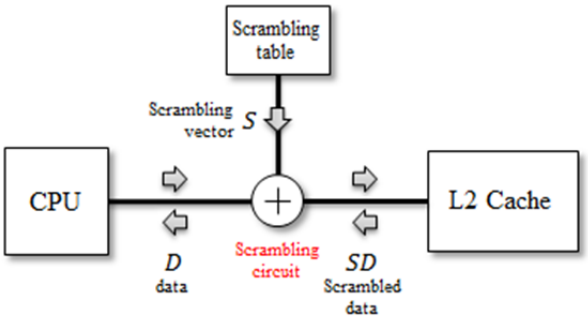

Fig. 2. Interleaved scrambling technique. It protects againts cold boot attacks

Using an internal mechanism of the scrambling table, keys are maintained in such a way that in normal operation they are periodically refreshed without the need of a full cache data update (interleaving mechanism). The whole set of keys point to subsets of addresses of the cache such that if a hacker discovers one key, during the limited time span before the key refresh, he would be able to execute a cold boot of the system and unscramble the data of the addresses in the same subset.

Key discovery can be theoretically achieved using power analysis, that is a kind of side-channel attack.

In the next section, this type of attack is explained in detail and the solution is addressed.

III. Statement of the Problem

It is assume that the system under attack is the L2 cache that includes a unidirectional eDLC technique and is protected against cold boot attacks using an IST. In this section it is shown that under these circumstances a hacker can use a side channel attack technique based on simple power analysis (SPA) to recover the internal keys of the IST scrambling table and therefore to discover the content of the cache data.

A. Attack model

Let's consider that the hacker knows the IST technique of the cache and how it is implemented. He ignores the internal keys and thus his objective is to discover them – from now on these keys will be named *scrambling vectors* (S) –. He can measure the current consumption or the radiated energy that emanates from the boundaries of the cache and can add a program in the operating system that sends repeatedly controlled data to predetermined addresses of the cache.

As it is overviewed in Fig. 3, the IST used in [10] selects a *scrambling vector* S from a table whose content is generated randomly. During the writing cycle it is mixed with data D that comes from the CPU using an XOR array and the scrambled data SD generated is sent to the cache. During this process the voltage transitions that take place in the XOR array and subsequent bus lines consume energy and therefore draw some amount of current from the power supply. By doing an appropriate profile of it, the amount of transitions occurring in the scrambling circuit can be estimated.

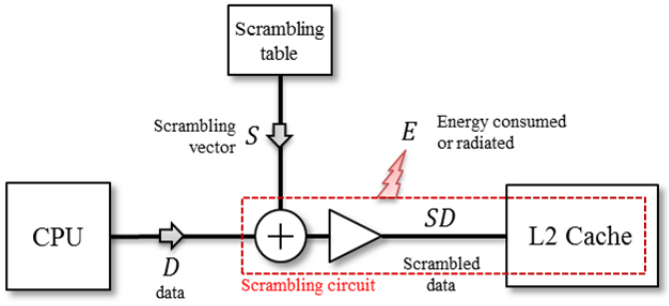

Fig. 3. Energy consumed or radiated during the write cycle of a cache.

Similarly to the current consumption, the transitions of the voltages in the lines, especially in the long lines, produce the emission of electromagnetic radiation that can be measured using a micro-antenna. With the proper control over the system and a scanning device, the scrambling circuit can be located and the electromagnetic impulses measured in such a way that the amount of transitions existing in the scrambling circuit can be estimated.

B. Side channel leakage model

Considering a standard bus implementation, the transmission of data is preceded and followed by rest states. Thus the number of transitions occurring is strongly correlated to the *hamming weight* of the data generated in the scrambling circuit. If E is the radiated or consumed energy then it can be modeled by a certain function $E = f(hw(SD))$ which in turn it will depend on the *hamming weight* of the scrambled data. For the attack it is enough two know that $f()$ is a monotonic function in order to generate an estimation of $hw(SD)$ from the measured E. The attacker knows that there is a one-to-one relationship between the maximum and minimum values such that if SD^* is the scrambled data maximizing the *energy* then it will also maximize the *hamming distance* and similarly will happen for the value SD_* giving the minimum. The bijective relationship (1) applies,

$$\begin{aligned} E_{max} &\overset{f()}{\leftrightarrow} H_{max} = hw(SD^*) \\ E_{min} &\overset{f()}{\leftrightarrow} H_{min} = hw(SD_*) \end{aligned} \tag{1}$$

C. Attack procedure

Assume the scrambling function $SD = D \oplus S$. S is the *scrambling vector* and is the target for the attack. Data D is the search space and if it doesn't include any ECC code then the maximum value achievable is $SD^* = (11\ldots1)$. Below we will see how to focus the attack with ECC.

A brute force attack consists of generating all possible D's and writing them to the same cache address. During this process the hacker records E for each datum. Then the datum giving E_{max} is selected, D^*. Since the attacker knows that $SD^* = (11\ldots1)$ then it must be $S = \overline{D}^*$.

For large data bit vectors brute force explodes making this strategy unfeasible. But because of the linearity of the scrambling operation, the attack can be executed stepwise. First, a subset of bits of data is selected and the brute force is applied. Once the local maximum is found then the search moves to the next subset of bits and the same procedure is applied. This follows until all bits are scanned out and, in the end, the target S is found.

When data includes ECC bits then the IST is organized as follows. Assume that after receiving the CPU data, the checker adds the redundant bits, such that we have an extended data vector $(D, \psi(D))$ where $\psi(D)$ are the redundant bits. In the scrambling table, the scrambling vectors are also extended with additional bits generated randomly to scramble the redundant bits of the data. Thus we have (S, S_ψ) being S the bits scrambling the data and S_ψ the bits scrambling the redundancy. The scrambled data (SD, SR) are finally generated as follows,

$$\begin{aligned} SD &= D \oplus S_D \\ SR &= \psi(D) \oplus S_\psi \end{aligned} \tag{2}$$

When the attack is executed it searches for the maximum *hamming weight* in the scrambled data, $(SD, SR)^*$. However, now the search in the data space is limited because of the redundant bits $\psi(D)$. During the search, e.g. think of the brute force strategy, two possible end points can be reached: (1.) only one maximum is found $(SD, SR)^*_U$ or (2.) more than one maximum exist, $(SD, SR)^*_M$.

1. In the case of $(SD, SR)^*_U$ the attack is straightforward. If the number of bits in SR is less than the number of bits in SD the unique maximum must be $((11\ldots1), SR)$ being SR

of no relevance for the attack. Then the *scrambling vector* is found with $S = \overline{D}^*$.

2. In the case of $(SD, SR)^*_M$ the maximum is not unique and this means that different values of D_i^* may give the same hamming weight $H^* = hw(D_i^* \oplus S) + hw(\psi(D_i^*) \oplus S_\psi))$ for $i = 1,2, \ldots$. This introduces an uncertainty to the hacker because now the breaking function $S_D = \overline{D}^*$ may not be valid. In general, there will be several values to consider and in particular none of them may include the case where $SD = (11 \ldots 1)$.

The fact that case 1 or 2 happens it depends on the *scrambling vector*, and in particular of the S_ψ segment of it. In the next section we present the proposed solution that is based on the following principle. Segment $S\psi$ is not completely generated at random but a filter is applied to the random generator such that case (2.) is always guaranteed.

Next illustrative examples for the cases (1.) and (2.) are shown.

D. Example

1) Case 1. Unique maximum

Table I presents a numeric example.

TABLE I. EXAMPLE OF UNIQUE MAXIMUM

Scrambling vector

S			S_ψ	
1	0	1	1	0

Data with ECC

D			$\psi(D)$	
0	0	0	0	0
0	0	1	0	1
0	**1**	**0**	**0**	**1**
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Scrambled data

SD			SR	
1	0	1	1	0
1	0	0	1	1
1	**1**	**1**	**1**	**1**
1	1	0	0	0
0	0	1	1	1
0	0	0	0	0
0	1	1	0	0
0	1	0	0	1

Hamming weight

H	
3	
3	
5	*
2	
3	
0	
2	
2	

At the left we have 3 bit data with 2 check bits, they count the number of ones in data. All possible values are listed. At the top we have the *scrambling vector* (101,10) and below that, the scrambled data (SD, SR). At the right, the *hamming weight* is evaluated. We can see that a unique maximum exists with a value of 5 and it corresponds to $SD = (111)$. Therefore, the *scrambling vector* can be discovered by complementing the data $D^* = (010)$ giving $S = (101)$.

2) Case 2. Multiple maximums

Table II presents the numeric example.

The content of the Table is the same except that now $S_\psi = (01)$ instead of (10). When the scrambled data is calculated, two maximums are found with *hamming weights* equal to 4, being the corresponding data $D^* = \{(011), (110)\}$. None of them contain the scrambled data $SD = (111)$ and as a consequence the *scrambling vector* cannot be obtained inverting any of both data.

TABLE II. EXAMPLE OF MULTIPLE MAXIMUMS

Scrambling vector

S			S_ψ	
1	0	1	0	1

Data with ECC

D			$\psi(D)$	
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	**1**	**1**	**1**	**0**
1	0	0	0	1
1	0	1	1	0
1	**1**	**0**	**1**	**0**
1	1	1	1	1

Scrambled data

SD			SR	
1	0	1	0	1
1	0	0	0	0
1	1	1	0	0
1	**1**	**0**	**1**	**1**
0	0	1	0	0
0	0	0	1	1
0	**1**	**1**	**1**	**1**
0	1	0	1	0

Hamming weight

H	
3	
1	
3	
4	*
1	
2	
4	*
2	

IV. PROPOSED SOLUTION

For our solution we start assuming that the ECC is based on the eDLC technique presented in Section II.A. As previously explained, it is a partial self-healing code that provides correction of bits on-the-fly and therefore significantly reduces the number of accesses to the external memory. It makes the eDLC coding oriented to high performance and reliable systems. When security is added through the IST methodology, the large overhead of the eDLC coding leverages the security against side channel attacks as it is explained hereafter.

Keep in mind that the eDLC coding subdivides data in subsets of three bits and adds two additional bits corresponding to the Berger code to them, like the configuration shown in the examples of tables I and II. Let's assume that $D^3 = (d_2, d_1, d_0)$ is one of these subdivisions. eDLC coding adds the carry and sum bits of the addition of these bits such that we have the codeword space $D^5 = (d_2, d_1, d_0, c, s)$, where $(c, s) = \sum_{all} d_i$.

To guarantee that after scrambling we will reach case (2.), *scrambling vectors* are generated in the non-codeword space such that never all 1ns are found in the scrambled data. Let's assume that our subset of bits scrambling D^5 is $S^5 = (s_4, s_3, s_2, s_1, s_0)$. The following generating function assures this condition, that set $S^5 \notin D^5$ and that multiple maximum *hamming weights* will always exist in the scrambled data.

$$\begin{gathered}(s_4, s_3, s_2, s_1) = rand() \\ s_0 = \overline{s_4 \oplus s_3 \oplus s_2}\end{gathered} \tag{3}$$

where $rand()$ is the random generator function and bit s_0 is the NXOR of the three most significant bits. Table III shows the codeword space of D^5 and the non-codeword space of S^5.

A. Stepwise attack

Consider the case where the hacker attacks the system by scanning all possible values of the data in a subset base. First, he takes the three data bits D_0^3 and generates all possible combinations, detects the maximums of the energy and records the values giving these maximums D_0^{3*}. Then moves to the next subset D_1^3, repeats the procedure and identifies the maximums D_1^{3*}. He continues until all subsets are scanned and the maximum subsets are created. Finally, he combines all the maximum subsets to create the set of data D^* giving the

maximum *hamming weighs* which are the candidates to extract the scrambling vector S using the equation $S = \overline{D}_i^*$.

TABLE III. CODEWORD AND NON-CODEWORD SPACES FOR D^5 AND S^5 RESPECTIVELY

	D^5				
	b_2	b_1	b_0	c	s
0	0	0	0	0	0
5	0	0	1	0	1
9	0	1	0	0	1
14	0	1	1	1	0
17	1	0	0	0	1
22	1	0	1	1	0
26	1	1	0	1	0
31	1	1	1	1	1

	S^5				
	s_4	s_3	s_2	s_1	s_0
1	0	0	0	0	1
3	0	0	0	1	1
4	0	0	1	0	0
6	0	0	1	1	0
8	0	1	0	0	0
10	0	1	0	1	0
13	0	1	1	0	1
15	0	1	1	1	1
16	1	0	0	0	0
18	1	0	0	1	0
21	1	0	1	0	1
23	1	0	1	1	1
25	1	1	0	0	1
27	1	1	0	1	1
28	1	1	1	0	0
30	1	1	1	1	0

Now, we verify the multiplicity number for the maximum *hamming weights*. In Table IV all possible *hamming weights* for the scrambled data $SD^5 = D^5 \oplus S^5$ are shown. At the top we have S^5 and at the left D^5. Assume that the hacker scans all possible values for the data vector, this is equivalent to trace the matrix column-wise. For each the column we count the number of maximums found. This is summarized below the Table.

TABLE IV. HAMMING WEIGHTS FOR THE DATA VECTOR AND SCRAMBLING VECTOR SPACES

$hw(SD^5)$ \ S^5	1	3	4	6	8	10	13	15	16	18	21	23	25	27	28	30
0	1	2	1	2	1	2	3	**4**	1	2	3	**4**	3	**4**	**3**	**4**
5	1	2	1	2	3	**4**	1	2	3	**4**	1	2	3	**4**	**3**	**4**
9	1	2	3	**4**	1	2	1	2	3	**4**	3	**4**	1	2	**3**	**4**
D^5 14	**4**	**3**	2	1	2	1	2	1	**4**	3	**4**	3	**4**	3	2	1
17	1	2	3	**4**	3	**4**	3	**4**	1	2	1	2	1	2	**3**	**4**
22	**4**	**3**	2	1	**4**	3	**4**	3	2	1	2	1	**4**	3	2	1
26	**4**	**3**	**4**	3	2	1	**4**	3	2	1	**4**	3	2	1	2	1
31	**4**	**3**	**4**	3	**4**	3	2	1	**4**	3	2	1	2	1	2	1
H^*	4	3	4	4	4	4	4	4	4	4	4	4	4	4	3	4
# of	**4**	**4**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**4**	**4**

Top number of maximums = 4 (for 4 cases)
Bottom number of maximums = 2 (for 12 cases)
Average number of maximums = 2.5

In a non-protected cache the number of elements in the set D^* is one and then S can be extracted immediately. For our case, in the most favorable situation all D_i^3 may have 2 maximums at most and therefore the total number of elements in the set D^* will be the combination of all the subset values, that is 2^l where l is the number of subsets. In the worst case the number of maximums in all the subsets will be 4 and thus the total number of elements in D^* will be 4^l. In the Table V these two limits are calculated for different sizes of data words. The average number of maximums is therefore 2.5^l. It can be seen that after 32 bits the number of elements that the hacker has to investigate is significantly large and for higher number of bits it becomes impractical.

TABLE V. NUMBER OF ELEMENTS IN THE SET OF MAXIMUMS D^*

Data word length	# subsets l	# Elements in D^* min	avg	max
8	3	8	15	64
10	4	16	39	256
12	4	16	39	256
14	5	32	97	1024
16	6	64	244	4096
20	7	128	610	16384
24	8	256	1525	65536
28	10	1024	9536	1048576
32	**11**	**2048**	**23841**	**4194304**
40	14	16384	372529	268435456
48	16	65536	2328306	4,295E+09
56	19	524288	3,638E+07	2,749E+11
64	22	4194304	5,684E+08	1,759E+13

In Fig. 4 an overview of the complete cache protection is shown.

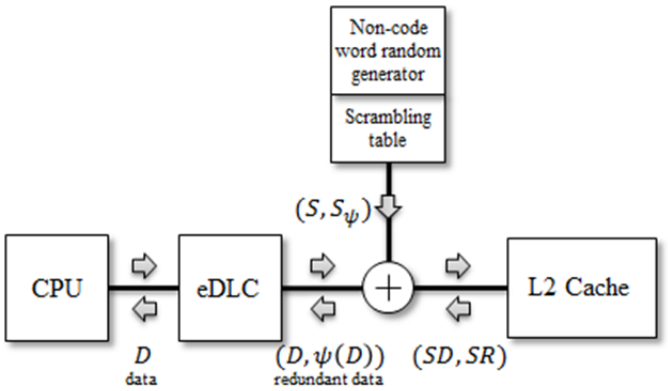

Fig. 4. Overview of the leveraged error correction and interleaved scrambling technique.

In the next section results for the propose technique are presented.

V. EXPERIMENTAL RESULTS

As a first experiment we have prepared a workbench to simulate attacks in the proposed protected cache. A program has been made in C++ that emulates the eDLC and ITS units using the non-codeword generation for the *scrambling vectors*. For 12 data bit lengths, brute force attacks have been programmed scanning all the data space. One attack means that a *scrambling vector* is generated and data is scanned exhaustively while the maximum *hamming weights* are saved and counted. These results have been registered for 3,000 attacks and are shown in the histogram of Fig. 5.

As it can be observed, in 1,008 attacks, the number of maximum *hamming weights* have been 16 while in 15 attacks the number of maximums have increased to 256, as predicted in Table V. The average number of maximums is 37.92, very close to the predicted average. In any attack a unique maximum is not found which confirms the validity of the technique against the SPA attack.

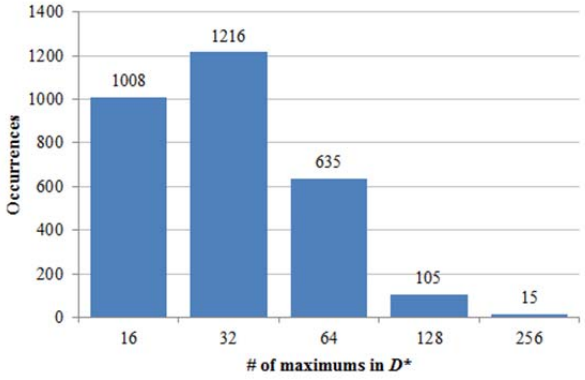

Fig. 5. Histogram of number of maximums found in the D^* set for 12 data bits after 3.000 attacks. Average is 37.92.

In order to evaluate the impact of the proposed solution in a cache, a 4-way comparison between standard L2 cache, IST from [10], eDLC from [12] and the proposed solution from Section IV is made. The CACTI tool from [12] is used for estimating the area and power consumption for different size architectures.

Table VI contains the CACTI tool results. The area and power consumption are simulated for different cache sizes and compared to the others. Note that the same tool has been used to simulate the methods from [12] and [10], as well as the proposed solution because each of them is a memory in itself. The IST is simulated using the equation from [10]. For the eDLC technique, only the first level of FAs was taken into consideration and simulated similarly as with the IST. However, due to data redundancy, the overall size is a little bit higher. Because the proposed solution is a combination of the IST and eDLC, it scales as the addition of the previous two proposals.

TABLE VI. CACTI RESULTS FOR DIFFERENT CACHE SIZES

	Size (KB)	Area occupied (mm^2)	Power consumption (W)
L2 cache	16	0.1479	0.0945
IST	4	0.0365	0.0394
eDLC	6.28	0.0842	0.0706
Proposed solution	10.28	0.1165	0.0824
L2 cache	32	0.2394	0.0973
IST	5.65	0.0486	0.0505
eDLC	8.87	0.1079	0.0794
Proposed solution	14.52	0.1409	0.0910
L2 cache	64	0.5493	0.1594
IST	8	0.0691	0.0579
eDLC	12.56	0.1292	0.0869
Proposed solution	20.56	0.1376	0.0703
L2 cache	128	1.1452	0.2802
IST	11.31	0.1222	0.0844
eDLC	17.75	0.1607	0.0978
Proposed solution	29.06	0.2280	0.0948

VI. CONCLUSIONS

In this paper, a novel technique for securing cache memories against side channel attacks is presented. Data scrambling techniques are a suitable against cold-boot attacks, as presented in [10], while the eDLC method from [12] is adequate for soft error detection and correction. By blending the two techniques and employing them in cache memories, side channel attacks become less effective. The evaluation of the proposal has been made using programmed models for the cache and the technological tool CACTI to evaluate the area and power impact of the solution proposed.

VII. ACKNOWLEDGEMENTS

This work was partially funded by the Spanish research program TEC2013-41209 and by the COST action IC1204.

REFERENCES

[1] F. Paget, "Financial fraud and internet banking: threats and countermeasures", Report, McAffee Avert Labs, 2009.

[2] J. A. Halderman, et. al., "Lest we remember: cold-boot attacks on encryption keys", Communications of the ACM – Security in the Browser, Volume 52, Issue 5, pp. 91 – 98, 2009.

[3] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, G. Palermo, "AES power attack based on induced cache miss and countermeasure", in Proceedings of the International Conference on Information Technology: Coding and Computing, Vol. 1, pp. 586 – 591, 2005.

[4] G. Doychev, D. Feld, B. Köpf, L. Mauborgne, J. Reineke, "CacheAudit: a tool for the static analysis of cache side channels", in Proceedings of the 22nd USENIX Conference on Security, pp. 431- 446, 2013.

[5] Z. Wang, R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks", in Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), pp. 494 – 505, 2007.

[6] T. Kim, M. Peinado, G. Mainar-Ruiz, "StealthMem: System-level protection against cache-based side channel attacks in the cloud", in Proceedings of 21st USENIX Security Symposium, USENIX Association, pp. 11 – 11, 2012.

[7] D. Gullasch, E. Bangerter, and S. Krenn. "Cache Games - Bringing Access-Based Cache Attacks on AES to Practice", In SSP, pages 490–505. IEEE, 2011.

[8] Y. Yuval, K. Falkner, "Flush+Reload: a high resolution, low noise, L3 cache side-channel attack", in Proceedings of the 23rd USENIX conference on Security Symposium, pp. 719 – 732, 2014.

[9] T. Muller, M. Spreitzenbarth, "FROST: Forensic Recovery Of Scrambled Telephones", in Proceedings of International Conference on Applied Cryptography and Network Security, pp. 373 – 388, 2013.

[10] M. Neagu, L. Miclea, S. Manich , "Interleaved scrambling technique: A novel low-power security layer for cache memories", Test Symposium (ETS), 2014 19th IEEE European, 2014.

[11] J. Singh, "Unidirectional Error Correcting Codes for Memory Systems: A Comparative Study", in Proceedings of Second International Conference on Advanced Computing and Communication Technologies, pp. 187 – 189, 2012.

[12] M. Neagu, L. Miclea, J. Figueras, "Unidirectional error detection, localization and correction for DRAMs: Application to on-line DRAM repair strategies", Proceedings of the 2011 IEEE 17th International On-Line Testing Symposium (IOLTS), pp. 264-269, 2011.

[13] CACTI tool v5.3, http://quid.hpl.hp.com:9081/cacti/