Side-Channel Security Analysis of Ultra-Low-Power FRAM-based MCUs

Amir Moradi and Gesine Hinterwälder

Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany
{amir.moradi, gesine.hinterwaelder}@rub.de

Abstract. By shrinking the technology and reducing the energy requirements of integrated circuits, producing ultra-low-power devices has practically become possible. Texas Instruments as a pioneer in developing FRAM-based products announced a couple of different microcontroller (MCU) families based on the low-power and fast Ferroelectric RAM technology. Such MCUs come with embedded cryptographic module(s) as well as the assertion that – due to the underlying ultra-low-power technology – mounting successful side-channel analysis (SCA) attacks has become very difficult. In this work we practically evaluate this claimed hardness by means of state-of-the-art power analysis attacks. The leakage sources and corresponding attacks are presented in order to give an overview on the potential risks of making use of such platforms in security-related applications. In short, we partially confirm the given assertion. Some modules, e.g., the embedded cryptographic accelerator, can still be attacked but with slightly immoderate effort. On the contrary, the other leakage sources are easily exploitable leading to straightforward attacks being able to recover the secrets.

1 Introduction

Side-Channel Analysis (SCA) attacks have become a serious threat to cryptographic implementations. Regardless of the theoretical robustness of a cryptographic primitive, secret materials used by its implementation can easily be recovered in case of absence of SCA-dedicated countermeasures. Case studies like [1, 4, 11–13] confirmed the effectiveness of such attacks to overcome the security of commercial products. Hence, the producers of security-related applications have moved towards integrating SCA countermeasures. For example the FPGA architecture *UltraScale* [18] – recently announced by Xilinx – offers many security features including DPA-protected bitstream encryption. Along the same lines, Microsemi has integrated many solutions to improve physical security of Actel's FPGA family *SmartFusion2* [10].

Texas Instruments (TI) has introduced ultra-low power FRAM-based microcontrollers (MCUs) with a couple of security features [15,16]. Ferroelectric RAM (FRAM) technology enables large-scale non-volatile memories that offer faster write operations, much larger tolerated number of write cycles, and a much lower

power consumption compared to equivalent flash memories. The low power consumption as well as the embedded cryptographic modules (e.g., an AES core) are the key factors of the offered security. It is claimed that due to the ultra-low-power feature of such MCUs and their low operating voltage (1.5 V) SCA attacks become extremely difficult to mount.

This article deals with the aforementioned features and presents practical investigation results with respect to the claimed hardnesses. The results of our analyses on an MSP430FR5969 MCU are summarized as:

- A couple of different power analysis attacks are feasible on the embedded AES module. We should highlight that such attacks are not as straightforward as those mounted on crypto engines of other MCUs e.g., Atmel's XMEGA [8]. That difference is mainly due to the low-power feature of the integrated AES module.
- Regardless of the underlying low-power architecture, software implementations of cryptographic algorithms executed on the underlying MCU are victims of power analysis attacks. Unsurprisingly, the secrets of such implementations can be easily revealed by means of straightforward state-of-the-art attacks.
- Due to the restricted speed of the FRAM technology, TI integrated a dedicated cache to be used when the MCU operates at a higher frequency than the access frequency of FRAM. As a known issue, the cache (miss/hit) can be a source of SCA leakage. We report case studies, which make use of this feature to launch effective SCA attacks.
- The internal architecture of MCUs is usually not known to end users. Such architectures can turn an implementation of a sound masking scheme to a vulnerable design. In order to examine such issues we consider an implementation of the *masking with randomized look-up table* countermeasure [14] which has particularly been developed for FRAM-based MCUs [7]. Our analysis shows that the unknown internal architecture of the underlying MCU causes the provably-secure masked design to have a first-order leakage, while it is supposed to provide security at all orders.

2 Features

Here we shortly recall a couple of features of TI's FRAM-based MCUs. We focus on those specifications, which are related to our security analyses.

2.1 AES Accelerator

In many of TI's FRAM-based MCUs – including the MSP430FR59xx family – an AES accelerator module is embedded. It supports both encryption and decryption for all key lengths (128, 192, and 256). Further, on-the-fly as well as offline round key generation scenarios are supported, and it is facilitated to be used in ECB, CBC, OFB, and CFB modes of operation. It should be highlighted

Table 1. AES accelerator performance figures

Key length	Encryption (clock cycles)	Decryption (clock cycles)
128 bits	168	168
192 bits	204	206
256 bits	234	234

that the AES module has not been designed for speed-critical applications although it can perform a complete encryption and decryption much faster than corresponding software on the same MCU. Table 1 shows the number of clock cycles the AES module requires to complete the respective operations.

As it is a stand-alone module, the MCU can perform other operations while the AES module is busy. It is noteworthy that the numbers given in this table are with respect to the on-the-fly computation of round keys while the decryption module requires to receive the last round key. The interested reader is referred to [17] for more detailed information including the performance of the other modes.

2.2 FRAM Architecture

As a promising alternative to non-volatile storage such as flash, FRAM technology offers many advantages. It avoids the long delays as well as the high current supply required for programming (writing). The advantageous features of the FRAM technology focus mainly on write operations. High speed (125 ns delay), low power (82 μA/MHz), and super high (10^{15}) write cycles have been reported for the 130 nm MSP430FR family of TI's MCUs.

As a disadvantage we should refer to the fact that FRAM reads are destructive. That is, every read must be followed by a write operation (with the same data). However, this is automatically handled by the FRAM controller, and the end user does not need to pay any attention to this. Therefore, the frequency of FRAM read operations are limited to the write speed. Due to this limit, TI has integrated a read cache in front of the FRAM to accelerate the operations in case the MCU operates at a higher frequency than the FRAM. In the MSP430FR family, the FRAM can be operated at up to 8 MHz without use of this cache. When the MCU operates at a frequency of 16 MHz (the maximum operation frequency of the MSP430FR family), the cache is utilized.

The integrated cache is a two-way associative cache containing two cache sets [17]. Each of these sets consists of two lines of four words (64 bits). The cache controller selects one of the cache lines to preload FRAM data and preserves recently-accessed data in the other cache line. If one of the four words stored in one of the cache lines is requested (a cache hit), no FRAM access occurs, and the requested data is read from the cache with full system speed. However, if none of the words that are available in the cache is requested (a cache miss), a wait state (one clock cycle at 16 MHz) controls the CPU to ensure proper

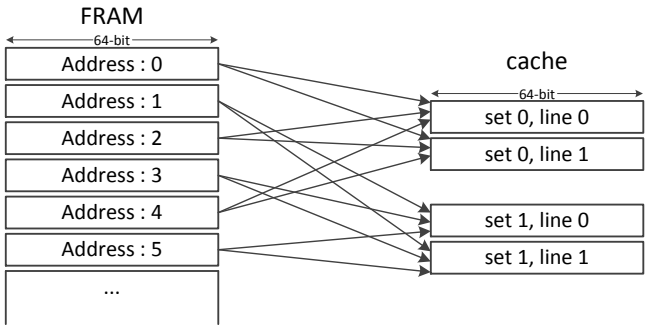


Fig. 1. Structure of the two-way associative cache

FRAM access. Therefore, memory read accesses on consecutive addresses can be executed without wait states when they are within the same cache line.

Each 64-bit location in FRAM can be cached in only one of the two sets in the cache. As shown in Figure 1, the most common scheme is to use the least significant bit of the FRAM location's address as the indicator to the corresponding cache set. We should emphasize that FRAM contains both program code and data, e.g., look-up tables, which are to be stored in the non-volatile memory. Hence, frequent jumps and frequent accesses to the pre-stored tables in FRAM can negatively affect the cache performance.

3 Analyses

In this section we present various analyses that we performed on an FRAM-based MCU. We first present the framework that we used, and then describe each analysis in detail.

3.1 Setup

The practical analyses have been conducted on an MSP-EXP430FR5969 Launchpad Evaluation Kit, and we used IAR Embedded Workbench IDE as well as Code Composer Studio to develop and compile the codes. This evaluation platform has been developed to facilitate power measurements. As shown in Figure 2, we could easily place a 1.8 Ω resistor at the VCC path of the MSP430FR5969 MCU while no stabilizing capacitor was placed between the measurement point and the MCU. We monitored the current passing through the MCU by means of a LeCroy WaveRunner HRO 66Zi digital oscilloscope at a sampling rate of 1 GS/s. We also used an I/O pin of the MSP-EXP430FR5969 Launchpad Evaluation Kit as trigger signal to align the collected traces. We provided a 16 MHz crystal oscillator as external clock source, and by clock source configurations drove the MCU at our desired frequency (explained below for each target).

Due to the very low power consumption of the MCU, we employed a DC blocker (BLK-89-S+ from Mini-Circuits) and an AC amplifier (ZFL-1000LN+

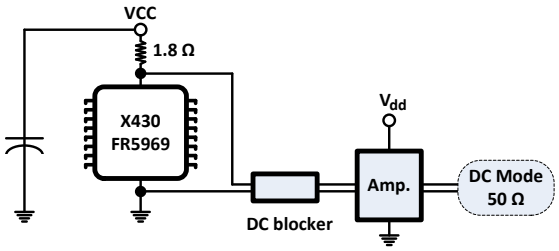


Fig. 2. Measurement setup

from Mini-Circuits) to collect the power traces with a considerably high quality. Further, we limited the oscilloscope bandwidth to 20 MHz to reduce the electrical noise.

Metrics: For the side-channel analysis we mainly used correlation power analysis (CPA) [3] to mount key-recovery attacks. However, in some cases we applied a statistical t-test [5]. The goal of such a scheme is not to examine a key-recovery attack, but rather it provides an overview of the existence of a leakage which might be exploited by an attack. The concept of t-test is based on the classical DPA attack of [9].

Following the concept of DPA, the traces $t \in T$ are categorized into two groups g_1 and g_2. Recall that Welch's (two-tailed) t-test is computed as

$$t = \frac{\mu(t \in g_1) - \mu(t \in g_2)}{\sqrt{\frac{\delta^2(t \in g_1)}{|g_1|} + \frac{\delta^2(t \in g_2)}{|g_2|}}},$$

where μ and δ^2 denote the sample mean and the sample variance respectively, and $|.|$ stands for the cardinality. The t-test indeed examines the validity of the *null hypothesis* as the samples in both groups (g_1 and g_2) were drawn from the same population. If the null hypothesis is correct, it can be concluded with a high level of confidence that a corresponding DPA attack cannot exploit the leakage.

For such a conclusion the Student's t-distribution density function in addition to the degree of freedom is applied to determine the probability of rejecting the aforementioned hypothesis (for more information see [5]). For typical evaluations, a threshold for $|t|$ as > 4.5 is defined to reject the null hypothesis and conclude that the device exhibits a first-order leakage. This process is repeated at each sample point independently.

The remaining point to mention is the way that the categorization of traces into the groups g_1 and g_2 is performed. For a *specific* t-test this classification is done based on a chosen intermediate value. During the measurements the input (plaintext) is taken randomly while the key is kept constant for all the collected traces. With respect to the corresponding DPA attack – for example – an Sbox

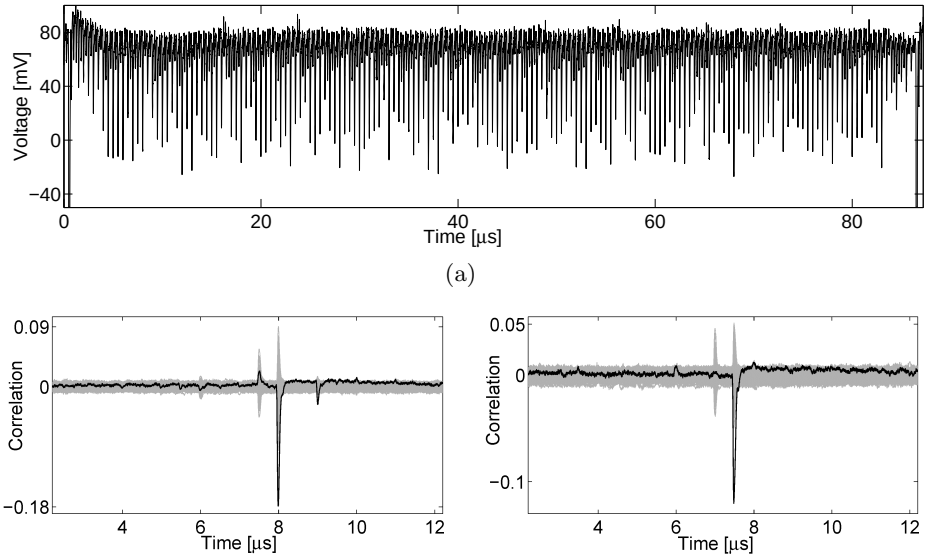
output bit directs the classification. Many intermediate values should be considered in the evaluations to examine the feasibility of each corresponding DPA attack. Instead, a *non-specific* t-test can be performed, which can examine the existence of leakage without any required intermediate value. In such a test, a fixed input (plaintext) is selected, and the measurements are randomly interleaved between the fixed and random inputs. So the non-specific t-test is also called *fixed vs. random* t-test. Hence, based on the given input (fixed or random) the traces are categorized into g_1 and g_2. It is noteworthy that such a leakage assessment scheme has been also used in [2].

3.2 AES Hardware Accelerator

As the first target we focus on the AES accelerator. As stated in Section 2.1, the AES module can perform the encryption and decryption functions in a couple of different settings. We evaluate only the AES-128 encryption function with on-the-fly round key computation, which takes 168 clock cycles. Following the configurations given in [17] we developed assembly code (in IAR Embedded Workbench IDE) to activate the aforementioned function. We intentionally wrote the code in such a way that it waits in a loop till the operation of the requested encryption is finished. It allows us to observe only the leakage of the AES accelerator module. Further, we configured the MCU to operate at a frequency of 2 MHz.

Figure 3(a) shows an exemplary power trace confirming its ultra-low power consumption. In order to examine the vulnerability of such a module to power analysis attacks, we first performed a couple of specific t-tests with intermediate values including i) the cipher round output bits (128 cases), ii) XOR between the cipher input and output bits (128 cases), iii) the SubBytes output bits (128 cases), iv) XOR between the SubBytes input and output bits (128 cases), v) the SubBytes output bytes (16×256 cases), and vi) XOR between different Sbox output bits ($\binom{16}{2} \times 8$ cases). The best results have been achieved considering the SubBytes output bits as well as the XOR between the SubBytes input and output bits (cases iii and iv). As a proof of concept we performed CPA attacks with the corresponding power models. For instance, Figure 3 presents the results of two CPA attacks using 100 000 traces. The power models have been chosen as i) an Sbox output bit and ii) a bit of the XOR result of an Sbox input and output. It is noteworthy that the attacks with common power models like Hamming weight (HW) models are also feasible, but not as efficient as those mentioned above.

We should stress here that although the AES accelerator is vulnerable to these state-of-the-art attacks, the effort an attacker needs to put in to recover the key is higher compared to e.g., the cases of Atmel's XMEGA [8] and KeeLoq [4]. This hardness results mainly from the low-power feature of the underlying technology. Further, as stated, we kept the MCU in an idle state to be able to observe the leakage of the AES module. The power consumption peaks related to the normal operation of the MCU are actually much higher than that of the AES module (see Figure 4(a)). Such high power peaks are due to the FRAM reads (as stated followed automatically by a write) as the program code (instructions) has



(a)

(b) model: an Sbox output bit

(c) model: a bit of Sbox input XOR output

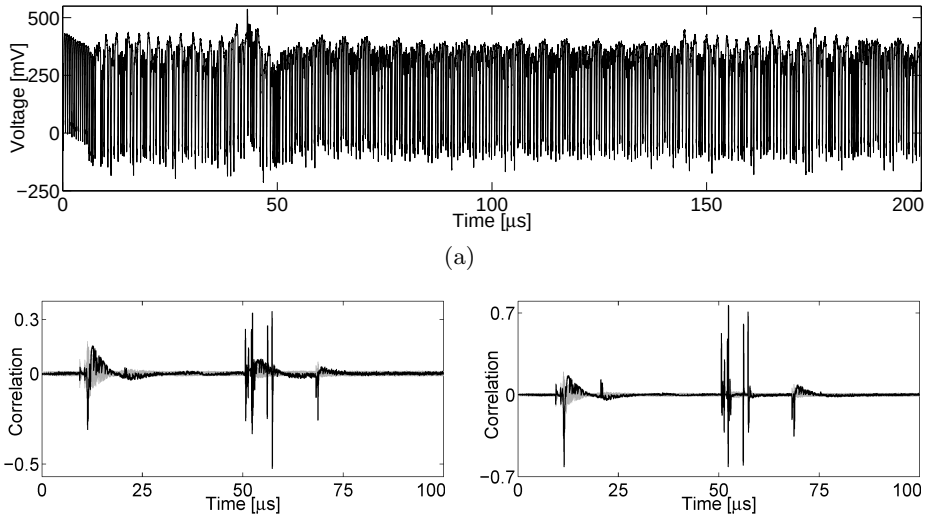
Fig. 3. AES in hardware: (a) a sample trace (full AES), (b) and (c) CPA attack results using 100 000 traces

been stored in FRAM. In short, when the MCU operates simultaneously with the AES accelerator, the above-presented attacks become harder.

3.3 AES in Software

As a case study to examine the leakage of the normal operation of the MCU, we took the AES-128 implementation recommended by TI and publicly available at `http://www.ti.com/tool/AES-128`. Both encryption and decryption functions are written in C, and we used IAR Embedded Workbench IDE to compile the *encryption* code and ran it on our evaluation kit at a frequency of 8 MHz. We first realized that the length of the traces is not constant, and the implementation needs different number of clock cycles depending on the plaintext. The source of this issue was found in the way that *multiply by 2* (used for MixColumns) has been implemented:

```
unsigned char galois_mul2(unsigned char value)
{
  if (value>>7)
  {
    return ((value << 1)^0x1b);
  } else
    return (value << 1);}
```



(a)

(b) model: an Sbox output bit

(c) model: HW of an Sbox output

Fig. 4. AES in software @ 8 MHz: (a) a sample trace (first round), (b) and (c) CPA attack results using 100 000 traces

As a result the implementation is vulnerable to a classical timing attack by predicting whether the extra reduction (XOR by the polynomial) is required or not. We have implemented such an attack and could easily recover the key using less than 5000 timing measurements (of the full encryption). Further, such an implementation is trivially vulnerable to a simple power analysis attack, where by observing each power trace the adversary can conclude whether the extra reduction was performed or not directly leading to a shrink in the key space.

Regardless of this issue we examined the efficiency of the start-of-the-art power analysis attacks. A sample trace covering the first round of the encryption is shown in Figure 4(a). In short, several attacks by different hypothetical power models are feasible, as expected. The results of two CPA attacks predicting a bit of an Sbox output as well as the HW of the Sbox output are shown in Figure 4. It is noteworthy that compared to the AES accelerator, the leakages associated with the execution of the MCU instructions are an order of magnitude easier to exploit. In fact, the ultra-low-power feature of the underlying MCU does not play an important role to harden the attacks.

3.4 Cache

As explained in Section 2, the FRAM is equipped with a cache to accelerate the access to consecutive memory locations when the MCU runs at a higher speed than the FRAM. In order to examine the effect of cache miss/hits we considered the AES encryption function in software (the case study of Section 3.3). To enable the cache we adjusted the clock source settings to operate the MCU at

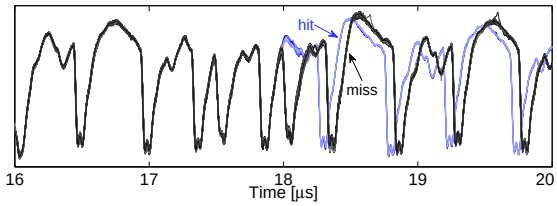


Fig. 5. AES in software @ 16 MHz: detection of cache miss/hit through power traces

a frequency of 16 MHz. Figure 5 shows a couple of traces during the SubBytes operation (one Sbox call). It can be seen that the traces belonging to cache hit and cache miss are clearly distinguishable. Therefore, a trace-driven cache attack is possible. In other words, by comparing a couple of traces the attacker would be able to detect whether each Sbox call caused a cache miss or not.

For an attack scenario let us consider two consecutive Sbox calls $S(p_1 \oplus k_1)$ and $S(p_2 \oplus k_2)$. If by observing the power traces the attacker detects a cache hit during the second Sbox call, it can be directly concluded that the two Sbox calls accessed nearby memory locations. Therefore, the attacker can gain certain information about $\Delta k = k_1 \oplus k_2 = p_1 \oplus p_2$. With respect to the underlying cache architecture, i.e., 64-bit lines (8 bytes), the five most significant bits of Δk can be recovered by the attacker. This is true, if the Sbox table starts at a location in FRAM corresponding to the first byte of a cache line. In other words, the first entry of the Sbox table needs to be stored in a location with address `xx...xx000`. Otherwise, the recovered bits of Δk is reduced to the four most significant bits.

As a proof of concept we developed a scenario to perform such an attack. In such a scenario we collected 256 power traces $T_{i \in \{0,\ldots,255\}}$ where the first plaintext byte p_1 is constantly set to an arbitrary value, e.g., 0, and the second one $p_2 = i$. The rest of the plaintext bytes can be arbitrarily selected. By observing the power traces and detecting a cache hit with $p_2 = p'_2$, a part of $k_1 \oplus k_2 = p'_2$ is recovered. Indeed, it is not required to collect all 256 traces; once a cache hit is detected the process can be terminated. For the second phase of the attack, again at most 256 traces are collected with plaintext bytes $p_1 = 0$, $p_2 = p'_2$, and $p_3 = i$. The same process is repeated to find the colliding case for $p_3 = p'_3$ and recovering a part of $k_1 \oplus k_3 = p'_3$. The selection of $p_2 = p'_2$ is necessary to avoid replacing the part of the cache filled during the first Sbox call. This process is repeated for the other plaintext bytes. At the end of the attack, the key space is limited to $2^5 \cdot 2^{3\times16} = 2^{53}$ or to $2^4 \cdot 2^{4\times16} = 2^{68}$ depending on the location the Sbox table is stored in. However, the attack can be extended to the second round and recover more relations to again shrink the key space.

An important issue, which should be mentioned, is that since the program code is also stored in FRAM, the execution of the instructions (fetching them from FRAM) by the MCU also affects the cache misses/hits. In the above-given example all 16 Sbox calls are trivially performed in a loop. If this is not the

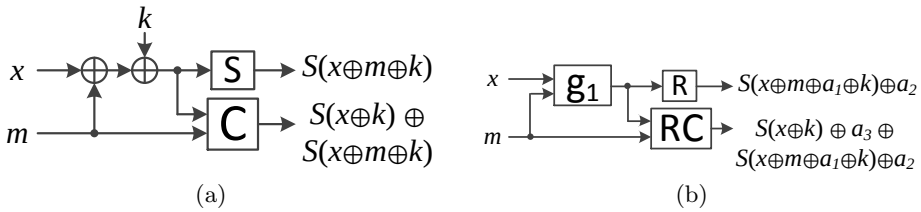


Fig. 6. (a) classical Boolean masking, (b) the scheme of [7]

case, the instructions performed between two consecutive Sbox calls can already replace the interesting line of the cache and avoid any cache hit by the second call. The presented attack scenario is only an example of a common scheme in the presence of a cache. It should also be emphasized that since the cache is shared between the program and data memory, exploiting the leakage by timing attacks (as a time-driven cache attack) is not trivial. In general, we show the leakage sources which should be taken into account when using such an MCU in a security-related application.

3.5 Internal Architecture

As the last case study we considered a masking scheme, which has been developed to provide security against side-channel attacks at any order. The scheme, which is based on the work presented in [14] has been implemented on an FRAM-based MCU as a proof of concept [7]. Therefore, we could easily integrate the same program code on our platform and perform the evaluations.

With respect to Figure 6, we restate the underlying scheme. In a classical first-order Boolean masking (Figure 6(a)), x and m (resp. input and random mask) are given to the device, which generates two outputs as $S(x \oplus m \oplus k)$ and $q : S(x \oplus m \oplus k) \oplus S(x \oplus k)$ as a shared representation of $S(x \oplus k)$. Such a scheme is certainly vulnerable to a second-order attack combining the leakages associated with the output shares. The concept followed in [14] and [7] is to involve more random data in the computations in such a way that the look-up tables g_1, R, and RC are precomputed based on the predefined key k and random data a_1, a_2, and a_3 in a secure environment (see Figure 6(b)). During the operation (similar to the classical Boolean masking) x and m are given to the device, and all the operations are performed by the aforementioned look-up tables. As each of the look-up tables involves a random a_i which is independent of the others, the adversary should not be able to recover any information by combining the leakage of the look-up tables and/or the output shares.

After a random selection of a_1, a_2, and a_3, the precomputations (Algorithm 1) are supposed to be performed in a secure environment, i.e., no side-channel measurement is permitted. After finishing all operations in the operational phase, e.g., for an Sbox as shown in Algorithm 2, $g_2(\cdot, \cdot)$ can be applied on (s_1, s_2) to obtain the unmasked result (in this case $S(x \oplus k)$).

Algorithm 1: Look-up Table Precomputation

input : k, a_1, a_2, and a_3

output: $g_1(\cdot,\cdot)$, $R(\cdot)$, $RC(\cdot,\cdot)$, and $g_2(\cdot,\cdot)$

$\forall i,j;\ g_1(i,j) = \quad i \oplus j \oplus a_1$
$\forall i;\quad R(i) = \quad S(i \oplus k) \oplus a_2$
$\forall i,j;\ RC(i,j) = S(i \oplus k) \oplus a_2 \oplus S(i \oplus j \oplus k \oplus a_1) \oplus a_3$
$\forall i,j;\ g_2(i,j) = \quad i \oplus j \oplus a_3$

Algorithm 2: Operation

input : x, m, $g_1(\cdot,\cdot)$, $R(\cdot)$, and $RC(\cdot,\cdot)$

output: (s_1, s_2)

$g = \ g_1(x,m)$; /* $: x \oplus m \oplus a_1$ */
$s_1 = R(g)$; /* $: S(x \oplus m \oplus a_1 \oplus k) \oplus a_2$ */
$s_2 = RC(g,m)$; /* $: S(x \oplus m \oplus a_1 \oplus k) \oplus a_2 \oplus S(x \oplus k) \oplus a_3$ */

A simplified and reduced version of the LED cipher [6] has been considered in [7] as an example to be implemented by the above-restated scheme. This reduced LED consists of only four rounds (cf. Figure 7) and a 16-bit (4×4) data width (i.e., it works only on the first column of the full LED state). We integrated the corresponding code and ran the MCU at a frequency of 8 MHz. It should be noted that before each encryption the look-up tables g_1, g_2, R, and RC are recomputed, i.e., there is no mask reuse in the whole scheme. By means of appropriate trigger signals as well as inserting a large enough gap between the precomputation phase and the operational phase we made sure to measure the power consumption of the MCU only during the operational phase. As we planned to perform a *non-specific* t-test, we collected 150 000 traces while a fixed value (as 0) or a random one – in a randomly interleaved fashion – was given to the MCU as plaintext. The result of both first- and second-order univariate t-tests are shown in Figure 8. Unexpectedly, the tests report both first- and second-order leakages.

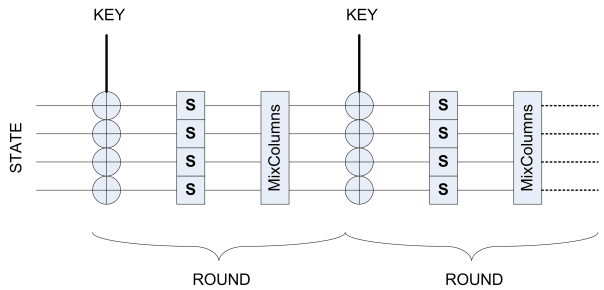


Fig. 7. Reduced LED (taken from [7])

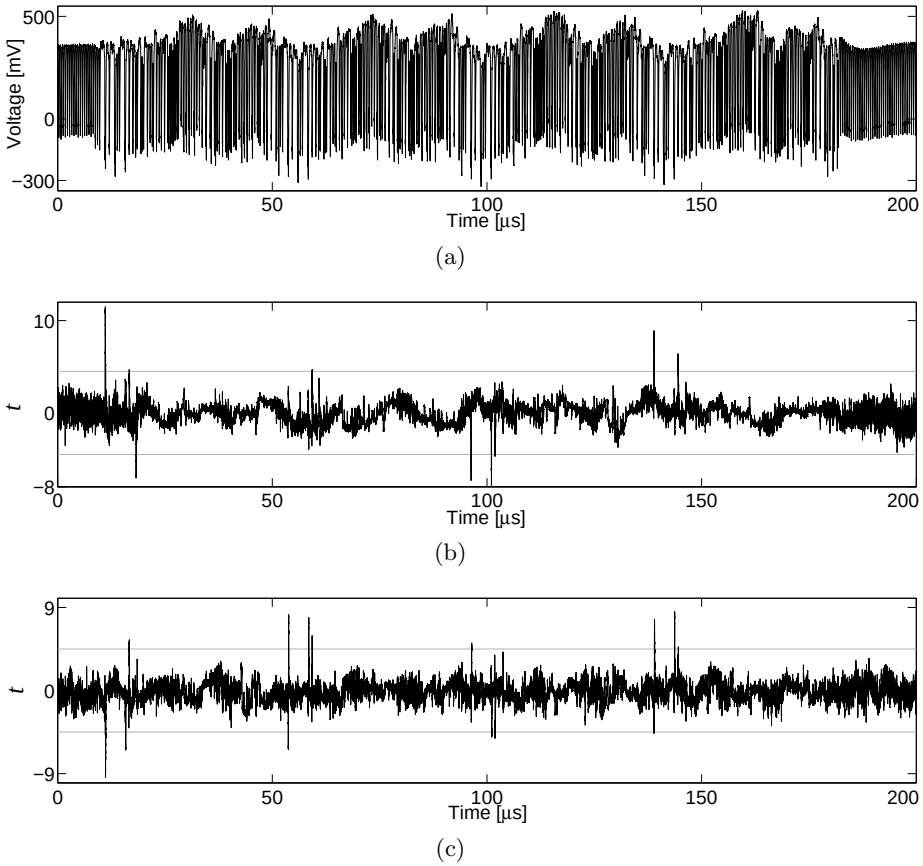


Fig. 8. Reduced LED @ 8 MHz: (a) sample trace (operational phase), (b) and (c) t-test results first- and second-order respectively using 150 000 traces

As stated, the scheme is supposed to provide resistance against the attacks at any order. The exploitable leakage, that we presented, is not due to the underlying scheme. In other words, we do not report any flaws in the algorithm or in the implementation of [7]. Instead, we show that even theoretically-sound countermeasures can fall into failure because of the internal architecture of the underlying platform. By slightly changing the program code and performing many measurements and analyses, we found out that the instructions which perform the table look-ups are the source of the observed leakages. More precisely, the exemplary instruction

```
1  mov.b @pointer, m0
```

which implements the call to the look-up table $RC(g, m)$ (see Figure 6(b) and Algorithm 2) causes such a leakage. Since the details of the MCU architecture are not publicly available, the reason of the observed leakage cannot be easily

pinpointed. Further, although we showed an exploitable leakage, the evaluation we performed (non-specific t-test) cannot give a clear assessment on the hardness of an attack exploiting such leakage.

We give an example to show the strong effect of the MCU's internal architecture on the side-channel vulnerability. We observed that if a random value is written to a location in FRAM the leakage associated with this write operation depends on the value which has been previously stored in that location. In other words, suppose that x has been stored at location `address`. A write operation, which stores a random value m at location `address`, leads to a leakage associated with the value of x as well. We observed such leakage during the evaluation of the above-expressed reduced LED implementation. At one point in the code (during the operational phase) a masked intermediate value is stored at a location where the unmasked plaintext had been stored before (at lines 4 and 7 of the below code):

```
1  mov   #STATE, pointer
2  rlam  #4, st0
3  add   st1, st0
4  mov.b st0, 0(pointer)
5  rlam  #4, st2
6  add   st3, st2
7  mov.b st2, 1(pointer)
```

In order to avoid such a strong leakage (shown in Figure 9) we cleared the contents of this location during the precomputation phase by:

```
1  mov   #STATE, pointer
2  mov.b #0x00,0(pointer)
3  mov.b #0x00,1(pointer)
```

This indeed is an evidence to the statement given above. We believe that such leakage is due to the FRAM architecture as well as the way a write operation is performed.

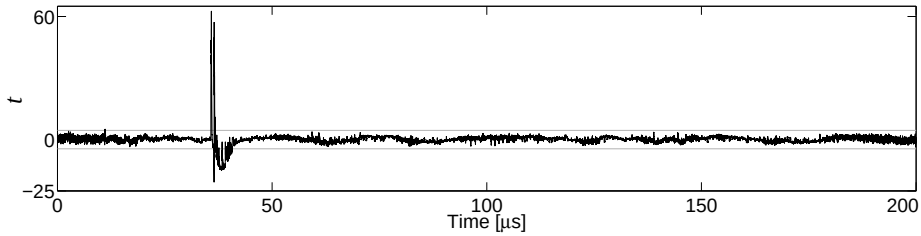


Fig. 9. Reduced LED (uncleared #STATE) @ 8 MHz: first-order t-test result using 35 000 traces

4 Conclusions

In this work we have extensively examined the side-channel vulnerability of FRAM-based MCUs of Texas Instruments as a platform for cryptographic applications. The motivation of this work is related to the relevant announcements dealing with the ultra-low-power feature of such MCUs and the claims on the hardness of power analysis attacks. Hence, we focused only on the power consumption of the underlying device and presented the corresponding evaluation results. The covered targets include the embedded AES accelerator (hardware), ordinary instructions of the MCU (software), FRAM cache, and the MCU internal architecture. In short, by means of practical investigations we confirm the hardness (but still feasibility) of the attacks on the embedded AES accelerator compared to similar targets of such attacks, e.g., the embedded AES core of Atmel's XMEGA MCUs. Such a hardness is mainly due to its low-power technology which leads to a high noise level in the measurements. However, when a cryptographic algorithm is implemented by the general-purpose MCU instructions, our practical results showed feasibility of straightforward and common DPA attacks without any serious difficulties. The cache, which has been integrated to accelerate the FRAM accesses, also comes with known security issues. Since an FRAM read must be followed by an FRAM write with the same value due to its destructive nature, an FRAM access consumes much more energy compared to a cache access. Hence, cache hit/miss can be clearly distinguished by observing the power traces. Although the cache is shared between the program and data memory (in MSP430FR5xxx family), we have shown that the trace-driven cache attacks (which exploit the sequence of cache misses/hits) are expectedly feasible. We also took a masking scheme into account, that has been developed in particular for platforms with a large non-volatile memory, e.g., FRAM-based MCUs. The scheme is based on precomputed randomized look-up tables and is expected to provide security against side-channel attacks of any order. Although there are no theoretical flaws in its developments, we have demonstrated that its implementation cannot pass a general leakage assessment test. The reason for such a failure lies in the details of the implementation platform (the MCU) regardless of the soundness of the underlying masking scheme.

On the one hand, the results we presented here are more or less expected as we targeted an unprotected platform where side-channel analysis should be feasible. On the other hand, this work gives an overview about the feasibility of exploiting various leakage sources of the underlying platform. Such information spreads awareness of the available leakage sources, and is certainly useful for cryptographic engineers, who deal with such a platform for security-related applications.

Acknowledgment

The authors would like to thank Stéphanie Kerckhof and François-Xavier Standaert from Université catholique de Louvain for their kindness in providing the source code of the masked implementation of the reduced LED of [7].

References

1. J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede. Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs. In *CT-RSA 2012*, volume 7178 of *LNCS*, pages 19–34. Springer, 2012.
2. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 326–343. Springer, 2014.
3. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
4. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In *CRYPTO 2008*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.
5. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop, Nara*, 2011.
6. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.
7. S. Kerckhof, F. Standaert, and E. Peeters. From New Technologies to New Solutions - Exploiting FRAM Memories to Enhance Physical Security. In *CARDIS 2013*, volume 8419 of *LNCS*, pages 16–29. Springer, 2014.
8. I. Kizhvatov. Side channel analysis of AVR XMEGA crypto engine. In *WESS 2009*. ACM, 2009.
9. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
10. Microsemi. Security: Protect Your Intellectual Property. `http://www.microsemi.com/products/fpga-soc/security`.
11. A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs. In *CCS 2011*, pages 111–124. ACM, 2011.
12. A. Moradi, D. Oswald, C. Paar, and P. Swierczynski. Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering. In *FPGA 2013*, pages 91–100. ACM, 2013.
13. D. Oswald and C. Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *CHES 2011*, volume 6917 of *LNCS*, pages 207–222. Springer, 2011.
14. F. Standaert, C. Petit, and N. Veyrat-Charvillon. Masking with Randomized Look Up Tables - Towards Preventing Side-Channel Attacks of All Orders. In *Cryptography and Security 2012*, volume 6805 of *LNCS*, pages 283–299. Springer, 2012.
15. Texas Instruments. Introducing advanced security to low-power applications with FRAM-based MCUs. `www.ebv.com/fileadmin/design_solutions/php/download.php?path=uploads%2Ftx_highlightcampaign%2FWolverine_Security_Whitepaper_01.pdf`, 2012.
16. Texas Instruments. Embedded Processing & DSP Resource Guide. `http://www.ti.com.cn/cn/lit/sg/sprt285f/sprt285f.pdf`, 2014.
17. Texas Instruments. User's Guide: MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family. `http://www.ti.com.cn/cn/lit/ug/slau367f/slau367f.pdf`, 2014.
18. Xilinx. UltraScale Architecture. `http://www.xilinx.com/products/technology/ultrascale.html`.