Power Analysis of Atmel CryptoMemory – Recovering Keys from Secure EEPROMs

Josep Balasch[1], Benedikt Gierlichs[1], Roel Verdult[2], Lejla Batina[1,2], and Ingrid Verbauwhede[1]

[1] K.U.Leuven ESAT/COSIC and IBBT
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be
[2] Radboud University Nijmegen, ICIS/Digital Security Group
Heyendaalseweg 135, 6525 AJ, Nijmegen, The Netherlands
{rverdult,lejla}@cs.ru.nl

Abstract. Atmel CryptoMemory devices offer non-volatile memory with access control and authenticated encryption. They are used in commercial and military applications e.g. to prevent counterfeiting, to store secrets such as biometric data and cryptographic keys, and in electronic payment systems. Atmel advertises the devices as "*secure against all the most sophisticated attacks,* [...] *including physical attacks*". We developed a successful power analysis attack on the authentication step of CryptoMemory devices. Despite the physical security claims by Atmel we found that the devices are not protected against power analysis attacks, except for counters that limit the number of (failed) authentication attempts, and thus power traces, to at most three. We examined the handling of these counters and discovered a flaw that allows us to bypass them, and to obtain power traces from an unlimited number of failed authentication attempts. Our attacks need as few as 100 power traces to recover the secret 64-bit authentication keys. From measurements to full key extraction, the attacks can be carried out in less than 20 minutes on a standard laptop. Once the keys are known, an adversary can read protected contents, clone devices, and manipulate the memory at will, e.g. to set the balance of an electronic wallet. To our knowledge, this is the first power analysis attack on Atmel CryptoMemory products reported in the literature.

Keywords: Atmel CryptoMemory, power analysis.

1 Introduction

In the past years, many commercial devices with security functionalities such as the KeeLoq-based remote keyless entry systems [19,22], the contactless Mifare DESFire MF3ICD40 card [29], or the FPGA bitstream encryption used by Xilinx Virtex FPGAs [26], were shown to be susceptible to power analysis attacks, e.g. DPA [23].

O. Dunkelman (Ed.): CT-RSA 2012, LNCS 7178, pp. 19–34, 2012.

The first "*secure memories with authentication*" [11,12] manufactured by Atmel appeared in 1999 under the name SecureMemory. These devices are mainly formed by a piece of EEPROM memory with access control logic and a cryptographic unit. A valid authentication grants read/write permissions to the memory. Failed authentications are recorded by authentication attempt counters (AACs), effectively locking the device after four or more attempts. A newer version of Atmel secure memories, commonly known as CryptoMemory, was released in 2002 as part of the Atmel AT88SCxxxxC series [7]. The AT88SCxxxxCRF [9] family, referred to as CryptoRF, appeared in 2003 providing the same features as CryptoMemory but with a contactless RF interface.

At the core of all these devices lies a proprietary stream cipher with secret specification, that we refer to as Atmel cipher. This cipher is employed during the mutual authentication protocol, in which a host and a CryptoMemory device authenticate each other by proving knowledge of a 64-bit shared secret key. A session key resulting from the mutual authentication can be further used to provide authenticated encryption of the communication channel.

CryptoMemory's security features, low cost, and ease of deployment have allowed these devices to be widely used in plenty of commercial and military applications [10]. A typical example is the use of the AT88SCxxxxC series to store High Bandwidth Digital Content Protection (HDCP) keys in products such as NVIDIA graphics cards [28], Labgear digital satellite receivers [4], or the Microsoft Zune player [30]. CryptoMemory devices are also used to strengthen security in systems vulnerable to counterfeiting schemes. They are for instance used in printers and printer cartridges manufactured by Dell, Ricoh, Xerox, and Samsung [1]. Other potential applications include "*appliances with smart batteries, set top boxes, video game consoles, video game cartridges, PDAs, GPS, and any system with proprietary algorithms or secrets*" [13], and storage of "*biometric information*" [10]. In smart card form, CryptoMemory devices are mainly used in vendor-specific electronic payments, e.g. in laundromats or parkings [2]. Further applications recommended by the manufacturer include "*ID and access cards, health care cards, loyalty cards, internet kiosks, energy meters, and e-government*" [14].

Contribution. We show that the secret authentication keys used by a CryptoMemory device can be extracted using basic, well-understood power analysis techniques. We devise a practical, non-invasive approach to exploit a flaw in the handling of AACs and to collect any number of power traces corresponding to failed authentication attempts. We have fully implemented and tested our attack on CryptoMemory devices from Atmel development kits and real world products, both in smart card and packaged IC form. Our attacks need as few as 100 power measurements for key extraction and can be carried out, including trace collection, in less than 20 minutes on a standard laptop. We also discuss how to fix the flaw in a way that future revisions of this (or similar) products are not vulnerable to such practical attacks, while remaining backwards compatible[1].

[1] We have informed Atmel about the vulnerabilities found in this work prior to its publication.

Paper Organization. In Section 2 we provide background information about CryptoMemory devices and their advertised security. We summarize previous work and we briefly sketch their security mechanisms. In Section 3 we develop an attack path for DPA attacks, and we provide the details and results of our attacks in Section 4. We discuss the implications of our findings as well as potential countermeasures in Section 5. We briefly conclude in Section 6.

2 Background

CryptoMemory devices are available either in plastic IC packages or as smart cards. The former communicate via a synchronous Two-Wire serial Interface (TWI), while the latter use the asynchronous T=0 protocol standardized in ISO 7816-3 [3]. CryptoMemory devices provide from 1 Kbit to 256 Kbits of memory divided into several user zones that are individually configurable with different access control policies. A separate configuration zone, customizable during the device personalization phase, is used to store such policies. A set of security fuses can be blown after the personalization phase in order to lock the configuration zone. CryptoMemory offers a total of three different security policies:

a) In password mode, a host simply needs to provide a 24-bit password to gain access to the zone. This mode offers a limited level of security, as all exchanged data (including passwords) is transmitted in the clear, i.e. it is vulnerable to eavesdropping and replay attacks.

b) In authentication mode, up to four 64-bit keys (in the following denoted by k) can be set during the personalization phase as shared secrets between host and device. An 8-bit AAC associated to each key controls the number of failed authentication attempts. The protected user zone(s) become inaccessible once AAC is set to x00[2]. Data transmitted to/from protected memory in this mode is in the clear but replay attacks do not apply.

c) In encryption mode, the communication channel between host and device is protected by authenticated encryption. A 64-bit shared session key K_s that is updated after each run of the mutual authentication protocol is used. Entering encryption mode requires the device to be already in authentication mode.

Regardless of the security policy, memory contents in the user zones are stored in the clear, i.e. CryptoMemory does not encrypt the data during the storage process.

Security of CryptoMemory devices. Atmel claims that CryptoMemory devices "*can secure data against all the most sophisticated attacks, including algorithmic attacks, systematic attacks, and physical attacks.*" [6]. In particular, physical attacks are counteracted by the use of tamper-proof features including metal shield layers over the active circuitry, encrypted internal buses, high-security test procedures, and defenses against timing and power supply attacks (to be understood as active tampering attacks, e.g. fault injection via glitching).

[2] In the most restrictive mode the maximum number of authentication attempts is set to four. The decreasing values of AAC are (xFF,xEE,xCC,x88,x00). A correct authentication automatically restores the value of AAC to xFF.

CryptoMemory also provides anti-tearing functionalities, i.e. in the event of a power loss during an EEPROM writing cycle, data can be recovered at the next power-up and written to the intended address. This feature is however optional, and it needs to be requested by the host prior to a write operation. A typical scenario in which this mechanism enhances security is payment systems, e.g. a malicious customer could try to remove a CryptoMemory-based card from the terminal slot before a decreased balance has been written to memory.

Surprisingly, we could not find a single reference to countermeasures against power analysis attacks in Atmel's marketing material and technical documentation. However, given the effort made to protect against invasive probing attacks and non-invasive tampering attacks, it is reasonable to assume that also power analysis attacks were considered. After all, it is claimed that "*CryptoMemory is designed to keep contents secure, whether operating in a system or removed from the board and sitting in the hacker's lab.*" [5]. In our view, it is likely that Atmel relies on the secrecy of the Atmel cipher and the AACs as countermeasures against basic power analysis attacks.

Previous Work. As it has previously occurred with other products using proprietary cryptographic algorithms [16,20,27], the security of CryptoMemory devices took a hit in 2010 when Garcia et al. [21] reverse-engineered the Atmel cipher and the authentication protocol used in the CryptoMemory family. The authors also cryptanalyzed these mechanisms and showed that an adversary could recover CryptoMemory authentication keys in 2^{52} cipher ticks using 2640 eavesdropped keystream frames. Biryukov et al. recently proposed an improved attack [15] that requires 30 eavesdropped keystream frames and 2^{50} cipher ticks to recover authentication keys. Other attacks against systems using CryptoMemory are known [24,32], but they exploit weaknesses in poorly designed protocols and mistakes during deployment rather than vulnerabilities of CrypoMemory devices.

Atmel Cipher. In the following, and for the sake of completeness, we briefly sketch the main functionality of the Atmel cipher. For a more formal and complete specification we refer the reader to [21].

Figure 1 depicts the inner structure of the Atmel stream cipher. The cipher state s is an element of $\mathbb{F}_2^{117}$ composed by a total of 4 shift registers. We denote these elements as left register l, middle register m, right register r, and feedback register f. In particular:

1. left register: $l = (l_0, l_1, ..., l_6) \in (\mathbb{F}_2^5)^7$

2. middle register: $m = (m_0, m_1, ..., m_6) \in (\mathbb{F}_2^7)^7$

3. right register: $r = (r_0, r_1, ..., r_4) \in (\mathbb{F}_2^5)^5$

4. feedback register: $f = (f_0, f_1) \in (\mathbb{F}_2^4)^2$

At each tick, the cipher state $s = (l, m, r, f)$ is transformed into a successor state $s' = (l', m', r', f')$ going through an intermediate state $\hat{s} = (\hat{l}, \hat{m}, \hat{r}, \hat{f})$. During this whole process the cipher takes a single input $a \in \mathbb{F}_2^8$ and produces an output keystream nibble $output(s') \in \mathbb{F}_2^4$.

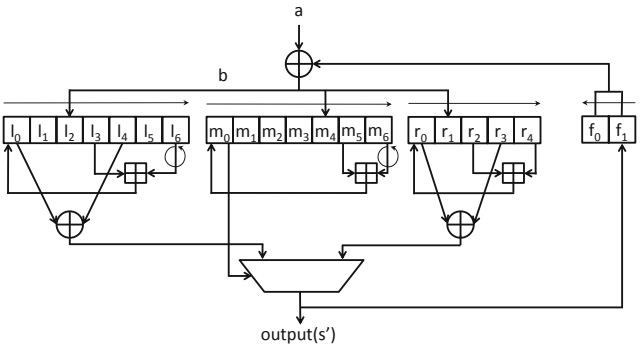


Fig. 1. Atmel cipher

Mutual Authentication Protocol. The mutual authentication protocol between a CryptoMemory device and a host is illustrated in Figure 2. Let $n_r \in (\mathbb{F}_2^8)^8$ be a host nonce, $n_t \in (\mathbb{F}_2^8)^8$ be a CryptoMemory device nonce, and $k \in (\mathbb{F}_2^8)^8$ a shared secret key. We denote a_r and $a'_r \in (\mathbb{F}_2^8)^8$ the host challenge authenticators, and a_t and $a'_t \in (\mathbb{F}_2^8)^7$ the device challenge authenticators. Both values are computed after feeding the values (n_r, n_t, k) into the Atmel cipher.

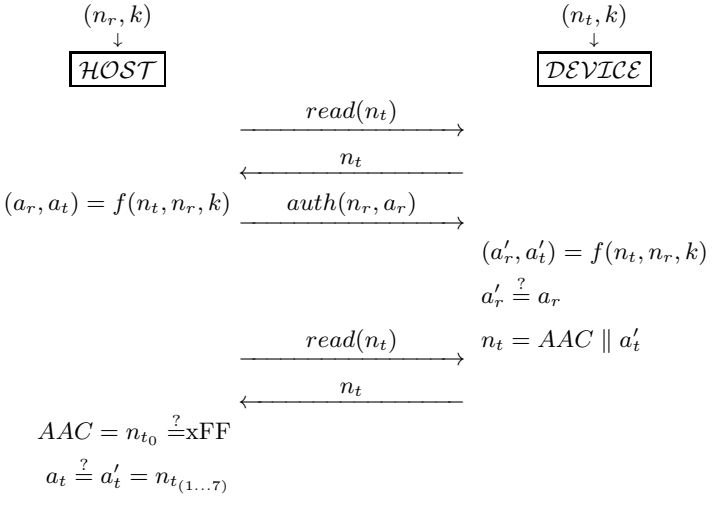


Fig. 2. CryptoMemory mutual authentication protocol

In the first phase of the protocol, the host reads the device randomness n_t and uses it, together with its own randomness n_r and the shared key k, to compute the authenticators (a_r, a_t). In the second phase, the host sends an authentication command to the device, namely $auth(n_r, a_r)$, including the value n_r and the first authenticator a_r. The device computes its own authenticators (a'_r, a'_t), and checks whether the provided a_r equals the calculated a'_r. If the check fails, the

State	Input bytes							Output nibbles						
(s_{0-6})	n_{t_0}	n_{t_0}	n_{t_0}	n_{t_1}	n_{t_1}	n_{t_1}	n_{r_0}	-	-	-	-	-	-	-
(s_{7-13})	n_{t_2}	n_{t_2}	n_{t_2}	n_{t_3}	n_{t_3}	n_{t_3}	n_{r_1}	-	-	-	-	-	-	-
(s_{14-20})	n_{t_4}	n_{t_4}	n_{t_4}	n_{t_5}	n_{t_5}	n_{t_5}	n_{r_2}	-	-	-	-	-	-	-
(s_{21-27})	n_{t_6}	n_{t_6}	n_{t_6}	n_{t_7}	n_{t_7}	n_{t_7}	n_{r_3}	-	-	-	-	-	-	-
(s_{28-34})	k_0	k_0	k_0	k_1	k_1	k_1	n_{r_4}	-	-	-	-	-	-	-
(s_{35-41})	k_2	k_2	k_2	k_3	k_3	k_3	n_{r_5}	-	-	-	-	-	-	-
(s_{42-48})	k_4	k_4	k_4	k_5	k_5	k_5	n_{r_6}	-	-	-	-	-	-	-
(s_{49-55})	k_6	k_6	k_6	k_7	k_7	k_7	n_{r_7}	-	-	-	-	-	-	-
(s_{56-62})	0	0	0	0	0	0	0	-	-	-	-	a_{r_0}	a_{r_1}	-
(s_{63-69})	0	0	0	0	0	0	0	-	-	-	-	a_{r_2}	a_{r_3}	-
(s_{70-76})	0	0	0	0	0	0	0	-	-	-	-	a_{r_4}	a_{r_5}	-
(s_{77-83})	0	0	0	0	0	0	0	-	-	-	-	a_{r_6}	a_{r_7}	-
(s_{84-90})	0	0	0	0	0	0	0	-	-	-	-	a_{r_8}	a_{r_9}	-
(s_{91-97})	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{10}}$	$a_{r_{11}}$	-
(s_{98-104})	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{12}}$	$a_{r_{13}}$	-
$(s_{105-111})$	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{14}}$	$a_{r_{15}}$	-
$(s_{112-118})$	0	0	0	0	0	0	0	a_{t_0}	a_{t_1}	a_{t_2}	a_{t_3}	a_{t_4}	a_{t_5}	a_{t_6}
$(s_{119-125})$	0	0	0	0	0	0	0	a_{t_7}	a_{t_8}	a_{t_9}	$a_{t_{10}}$	$a_{t_{11}}$	$a_{t_{12}}$	$a_{t_{13}}$
$(s_{126-132})$	0	0	0	0	0	0	0	K_{s_0}	K_{s_1}	K_{s_2}	K_{s_3}	K_{s_4}	K_{s_5}	K_{s_6}
$(s_{133-139})$	0	0	0	0	0	0	0	K_{s_7}	K_{s_8}	K_{s_9}	$K_{s_{10}}$	$K_{s_{11}}$	$K_{s_{12}}$	$K_{s_{13}}$
$(s_{140-141})$	0	0						$K_{s_{14}}$	$K_{s_{15}}$					

Fig. 3. Generation of authenticators (a_r, a_t) and K_s given inputs (n_t, n_r, k)

value of AAC is decreased; otherwise, it is set to xFF. The device also updates the value of n_t by concatenating the 8-bit AAC with the 56-bit authenticator a'_t. In the final phase, the host reads the recently updated value of n_t. It first checks whether the authentication was successful (i.e. whether AAC holds the value xFF), and later compares the authenticator a_t with the provided a'_t. If all checks are correct, then the mutual authentication protocol succeeds.

The procedure to compute the authenticators (a_r, a_t) resulting of feeding the values (n_t, n_r, k) into the Atmel cipher is intuitively depicted in Figure 3. Each of the states s_i indicates one tick of the cipher for which an input byte a is given and an output nibble $output(s')$ is obtained. At the start of the protocol all registers of the internal state are initialized to zero. In a first phase, ranging from states s_0 to s_{55}, the three parameters (n_t, n_r, k) are scrambled into the cipher state. The output keystream nibbles generated by the cipher are for the moment ignored. In a second phase, from states s_{56} to s_{125}, all input bytes are set to zero. The output nibbles obtained after some of the ticks are used to form the authenticators a_r and a_t. In the final phase, the session key K_s is computed during states s_{126} to s_{141}.

3 Developing an Attack Path

Attack Goal. The aim of an adversary targeting CryptoMemory devices may vary depending on the deployment setting, but typically the objective will be

to either read protected information (e.g. to manufacture clone chips) or to overwrite memory contents (e.g. to increase the balance in payment scenarios).

In the following we will assume an adversary that possesses a CryptoMemory device configured either in authentication mode or encryption mode. Note that the first security operation in either of these modes is always the mutual authentication protocol. In other words, their security relies on the secrecy of the authentication key (or keys) k. If an attacker knew k, he could easily compute session keys K_s and encrypt/decrypt valid communications. Therefore, we define the goal of the adversary as the recovery of k.

Attack Approach. Previous work has already pointed out cryptographic weaknesses in the Atmel cipher, but the most efficient attack still requires substantial computational effort, e.g. two to six days computation on a cluster with 200 cores [15]. We focus on power analysis attacks as they are often more practical. Note that in the case of deployed CryptoMemory devices, eavesdropping the communication line for later cryptanalysis requires physical access to the target device. Thus, physical access for power analysis does *not* imply an additional requirement.

Target Devices. We have tested three different models of the CryptoMemory family, namely the chips AT88SC0404C, AT88SC1616C, and the newer AT88SC0808CA. These devices differ in the amount of user zone slots and/or size available to the end application, but the protocol to carry out the mutual authentication is identical for all of them.

Given that CryptoMemory devices do not contain a microcontroller, we can assume that the cryptographic unit, including the Atmel cipher, is fully implemented as a hardware module. We further assume that the implementation computes one cipher tick in one or two clock cycles, which gives us an idea of the type of pattern to look for in the power traces.

In the following we describe the experimental setup used in our analysis and the steps followed to evaluate the smart card versions of CryptoMemory. We summarize the slightly different approach required for CryptoMemory ICs, leading to the same results, at the end of the section.

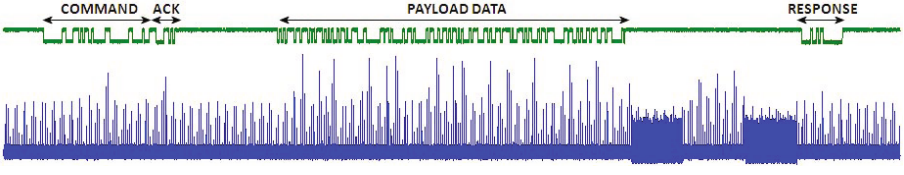
3.1 Experimental Setup

The main element of our experimental setup is a Virtex-II FPGA [33] that communicates with a PC (user interface) and a target device (CryptoMemory). The FPGA can be configured to communicate with either of the CryptoMemory device forms, i.e. as a T=0 compatible smart card reader or as a TWI master device. Apart from the communication line, the FPGA fully controls all external signals provided to the target device. For instance, the reset signal (RST), the external clock (CLK), and even the power supply (VCC) can be manipulated at will and at any point in time. A 50 Ohm resistor is inserted in series in the ground (GND) line of the target device. We use a Tektronix DPO7254 [31] oscilloscope to measure the voltage drop over this resistor, thus obtaining a trace proportional to the device's power consumption. Note that we do not exhaust

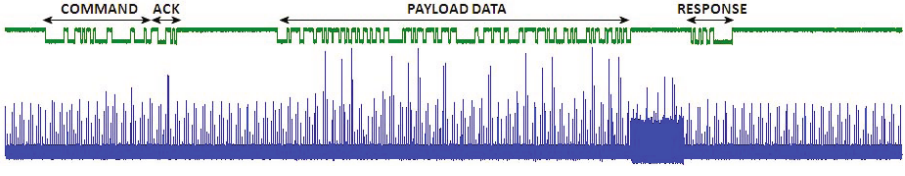
the capabilities of the oscilloscope and that a low-cost model could be used just as well. For all experiments we used a sampling rate of 100 MS/s and 20 MHz bandwidth. In addition to the power consumption we also use the oscilloscope to monitor RST, CLK, and I/O lines.

3.2 Initial Investigation of Power Traces

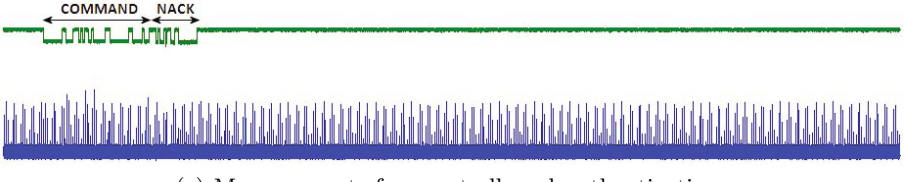
The first step in a power analysis attack is to determine in which part of the protocol the secret key is used by the target device, i.e. at which points in time are the key bytes fed as input to the Atmel cipher. An overview of the I/O and a power trace obtained during the processing of a mutual authentication command is shown in Figure 4. Figure 4(a) represents a successful authentication, Figure 4(b) represents a failed authentication, and Figure 4(c) represents an authentication attempt when the AAC is set to x00.



(a) Measurements for a successful authentication.



(b) Measurements for a failed authentication.



(c) Measurements for a not allowed authentication.

Fig. 4. I/O and power trace for a successful authentication (a), a failed authentication (b), and a not allowed authentication (c)

The leftmost part of the figures start with the host sending 5 bytes corresponding to an authentication command. After receiving these bytes, the card can reply either with an acknowledgement ACK, indicating that it accepts the command, or with a negative acknowledgement NACK, refusing to process the command. The latter is shown in Figure 4(c), where the CryptoMemory device has locked access to the associated user zones. If the device acknowledges the

command then the host sends the payload data, i.e. 16 bytes corresponding to the values n_r and a_r. Upon reception, the card performs a series of operations to determine whether the authenticator provided by the host is correct. Finally, the card sends a response to the host indicating the outcome of the authentication attempt.

The most interesting part of the figure corresponds to the card calculations upon reception of the payload data. One can clearly notice that the calculation interval in Figure 4(b) is significantly shorter than in Figure 4(a). Further, a clearly distinguishable pattern with higher power consumption appears twice for the valid authentication and only once for the invalid authentication. This pattern corresponds to EEPROM writings in configuration memory: the first one, present for both valid and invalid attempts, corresponds to decreasing the value of AAC; the second one, only present for the valid authentication, corresponds to writing the new value of n_t and the session key K_s. Note that writing a new value n_t implies a valid authentication, and thus AAC is restored back to xFF.

Although not visible in these overview plots, we noticed that the cryptographic unit appears to run at a clock frequency slower than that we provided externally. Further investigations showed that the device internally derives this slower clock signal by dividing the external clock by approximately a factor 200.

The card's calculation of the values (a'_r, a'_t) must happen before the second EEPROM write operation, simply because the n_t cannot be updated without having computed a'_t beforehand. Our experiments have shown that the device computes the authenticators (a'_r, a'_t) on-the-fly, i.e. while the payload data (n_r, a_r) is being received from the host. Similarly, the calculation of the session key K_s is performed between the two EEPROM writings, only after the device has authenticated the host by verifying that a_r and a'_r are equal.

Figure 5 shows a zoomed version of the I/O and power traces during the transmission of the value $n_r = (n_{r_0}, \ldots, n_{r_7})$ to the card. A clearly distinguishable peak in the power trace is visible at the end of each byte transmission, while a total of six high peaks are also identifiable during the transmission of a byte. As explained in Section 2 the calculation of (a'_r, a'_t) requires 126 cipher ticks. The pattern in the power traces has a perfect mapping with the cipher behavior illustrated in Figure 3 considering a hardware implementation. In fact, the first peak in Figure 5 corresponds to the state s_6, i.e. when the value n_{r_0} is fed to the cipher. The following six peaks correspond to states s_7 to s_{12}, in which the card uses its own randomness (values n_{t_2} and n_{t_3}) as inputs. Once n_{r_1} has been received over the I/O line, a new peak corresponding to state s_{13} appears in the power trace, and the pattern is repeated for every transmitted byte. In summary, each of the 50 peaks highlighted in Figure 5 corresponds to a cipher state ranging from s_6 to s_{55} in Figure 3.

Since the key k is scrambled into the cipher states s_6 to s_{55}, the device might leak sensitive information through its power consumption. We could not immediately identify a visible pattern in the power consumption that could relate to k (SPA leak) but we also did not expect that from a hardware implementation of a stream cipher. Therefore, we focused our attention on attacks that use statistical

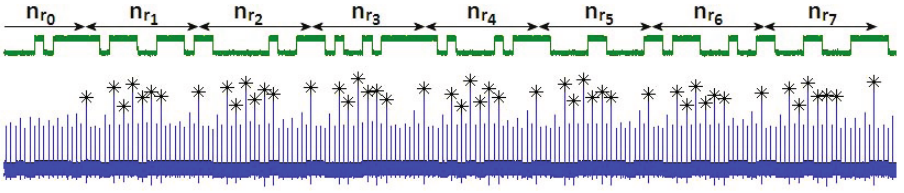


Fig. 5. I/O and power traces during the transmission of n_r in authentication command. Interesting peaks are marked with *.

post-processing of the collected power traces. Recall that DPA attacks require the processing of multiple power traces corresponding to multiple authentication attempts. For each attempt the device must use the same (unknown) k and varying input data.

3.3 Overcoming Authentication Attempt Counters

Even though we have identified the parts in the power traces that correspond to processing of the secret key k, an important practical issue still remains. In our adversarial model, the adversary possesses a CryptoMemory device that is *already* configured. Thus, he does not know the secret key k. In order to run the mutual authentication protocol, an adversary needs to provide an authenticator a_r to the device. However, as the attacker cannot compute this value correctly, the CryptoMemory device will not authenticate the host and, as a consequence, it will decrease the associated AAC. Given that the user zones become inaccessible to the host after four failed authentication attempts, an adversary can collect at most three power traces before permanently locking the device. The issue is that three traces are clearly not sufficient to carry out a successful DPA attack.

There are several ways to try to deal with this limitation. If the application under attack were to use the same key in all deployed CryptoMemory devices, then it would suffice to collect power measurements from several devices. One could also try to take many measurements from a single device, effectively sacrificing it. But as can be seen in Figure 4(c), once the AAC value is set to x00 the device no longer computes the authenticators and measurements would thus be worthless. Therefore, an adversary could obtain at most four power traces per device tested. However, a scenario in which all devices share a single key k seems unlikely to be found in secure deployments[3].

An alternative could be to use template attacks as introduced by Chari et al. [18]. The devices provided by Atmel in evaluation kits, completely configurable by the user, could be used to build such templates. We expect template attacks to require very few power traces from the target device, but it is not clear if three traces would suffice, due to the large cipher state. We did not investigate this approach further as we were interested in more simple attack paths.

[3] Atmel actually recommends to diversify keys in all CryptoMemory deployments by combining the configurable device ID with a unique master key [8], e.g. using a hash function.

We followed a more intuitive approach to overcome the limitation imposed by the AACs. Recall from Figure 4 that *all* the information required to perform DPA, i.e. key-dependent information in power measurements, is obtained *while* the value n_r is being sent to the card. In other words, the information is available to the attacker *before* the card actually decreases the value of AAC. Our approach consists in injecting a negative pulse in the RST signal of the device before the new value of AAC is written into configuration memory. Doing so forces the device to reset its state before beginning the EEPROM write operation.

A successful implementation of this simple procedure is shown in Figure 6. Besides the I/O and power traces, the figure also shows the RST signal input to the device. Injecting a pulse on the RST line right after sending the payload data successfully resets the device. This can be observed on the I/O line, as the device sends its Answer-To-Reset (ATR) value to the host device right after the rising edge in RST. Note also that the first EEPROM write pattern indicating AAC being decreased does not appear in the power trace.

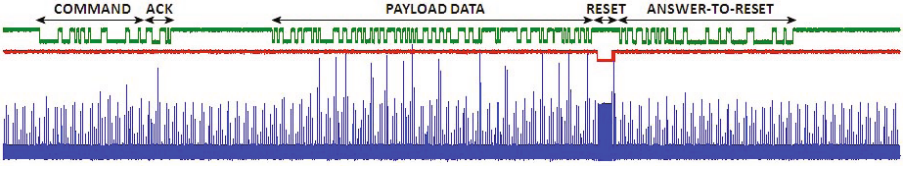


Fig. 6. I/O, RST, and power traces during interrupted authentication

The timing of the RST pulse is far from critical as the adversary can inject it at any point after the reception of n_r and before the first EEPROM writing, i.e. during the transmission of the value a_r. Recall that the transmission of a bit in the I/O line requires 372 clock cycles, and a byte transmission requires a total of 12 bits (1 start bit, 8 data bits, 1 parity bit, and 2 stop bits) [3]. Taking that into account, the adversary has an interval of 35 712 clock cycles in which the RST pulse can be sent to the card.

Note that resetting the device before the EEPROM writings implies that the value n_t can never be updated. As a consequence, all power measurements collected using this approach will correspond to the same device randomness n_t. Although in other scenarios this characteristic could be problematic, in our case it does not limit the success or applicability of DPA attacks. This is because an attacker can still provide varying values of n_r for each authentication attempt, which is fed into the Atmel cipher some ticks before the key k.

Differences with CryptoMemory Packaged ICs. Besides some particularities caused by the use of TWI instead of T=0, the overall behavior of CryptoMemory packaged ICs resembles what we have presented until now. Most important is the fact that the calculation of the parameters (a'_r, a'_t) is done in exactly the same way as in the smart card, i.e. on-the-fly while the host sends the values (n_r, a_r). It is thus possible to identify which zones of the power

measurements correspond to which states of the Atmel cipher during the feeding of the values (n_t, n_r, k).

The main physical difference between CryptoMemory packaged ICs and smart cards is that the former do not have an external RST pin. This could have been a problem, but an "equivalent" mechanism to overcome the AACs consists in cutting the supply voltage (VCC) before the counters are decreased. Similarly to the RST mechanism the timing accuracy to cut the voltage is not critical, and the adversary has plenty of time to perform it[4].

Once the power traces are obtained, the attacks on CryptoMemory in smart card form and in IC form are identical and lead to very similar results.

4 Power Analysis Attack

We obtained a set of 1000 power traces sampled during executions of the mutual authentication protocol, for which we provided random nonces n_r to the device and, each time, reset it as described above.

We processed the traces with a simple routine that extracts the peaks highlighted in Figure 5, yielding a new set of highly compressed and well aligned traces. Contrary to typical attacks on block cipher implementations, the key bytes can not be recovered independently but should be recovered in the order in which they are fed into the Atmel cipher implementation, i.e. first k_0, then k_1, etc.

We first investigated the feasibility of basic DPA attacks that recover k one byte at a time. We used a Hamming distance power model on the full cipher state, i.e. the total number of bit flips in the transition from state s to state s', and Pearson's correlation coefficient as distinguisher [17].

As explained in Section 2, each key byte is fed into the cipher three times in consecutive cipher ticks. Our basic attack worked best when we attacked the last cipher tick that fed in a given key byte, e.g. for byte k_0 the transition from state s_{29} to s_{30}.

Figure 7 exemplarily shows the results we obtained when attacking k_6 (the worst case). The left part of the figure shows the correlation coefficients for all 256 hypotheses, plotted over "time", computed using all 1000 measurements. The right part of the figure shows the evolution of the maximum and minimum correlation peaks per key hypothesis, plotted over the number of traces used. We verified that all key bytes can be recovered in this way using less than 500 traces.

Our slightly more elaborate attack additionally exploits two simple facts. First, we know exactly which sample in the power traces corresponds to which cipher tick. Thus, the attack focuses on the correct sample in the traces and ignores all other samples that appear as noise. Second, each key byte is fed into

[4] CryptoMemory packaged ICs require the host to perform acknowledge polling after sending a mutual authentication command. If the host does not send any polling command the IC is effectively idle, which gives enough room for an adversary to switch off the supply voltage.

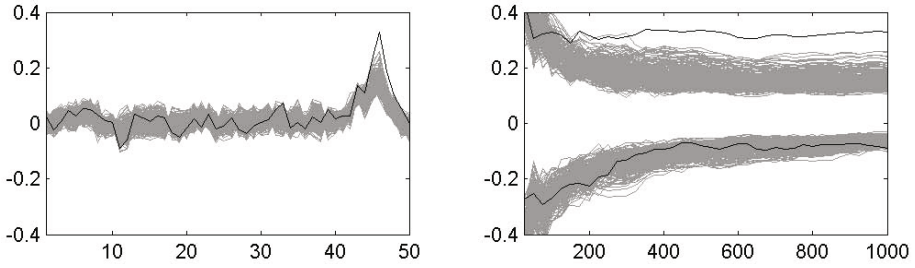


Fig. 7. Results of basic DPA attack. Correlation coefficients per key hypothesis (left), and evolution of correlation peaks per key hypothesis (right).

the cipher three times. Thus, the attack targets all three transitions and adds up the obtained correlation coefficients, per key hypothesis.

In addition, the attack further exploits that the dependence of the cipher state on k grows only slowly and byte per byte. Similar to the strategy described in [19], the attack does not aim to immediately identify the exact value of a key byte, but it maintains a list of the best candidate values and re-evaluates them when attacking the next key bytes.

We verified that this enhanced attack recovers the correct key k using less than 100 measurements. We note that both attacks, from measurements to full key recovery, can be carried out in less than 20 minutes on a standard laptop. Algebraic side-channel analysis would perhaps allow to work with even fewer traces, but it requires to build templates.

5 Implications and Countermeasures

We have shown that an adversary can easily extract the secret authentication key(s) k from CryptoMemory devices using basic, well-understood power analysis techniques. As a consequence, the adversary can perform all actions that any authenticated host could perform. This includes reading the memory, which allows to clone the device, and manipulating its memory contents at will.

The success of our attack is due to two design flaws in CryptoMemory. First, the implementation of the Atmel cipher is not protected by countermeasures against power analysis attacks, except for the AACs that limit the number of traces that can be obtained from a device before it locks itself to at most three. And second, an inadequate handling of the AACs that allows an adversary to bypass this limitation and to obtain any number of measurements from a given device without locking it.

A simple way to prevent our attack would be to modify the handling of the AACs. As learned from our experiments, CryptoMemory performs the authentication procedure as follows: compute the authenticators, decrease the value of AAC, compare the authenticators, and update the value of AAC according to success/failure when writing n_t in memory. This sequence can also be extracted

from Figures 4(a) and 4(b). Atmel explicitely states that this procedure is used "*to prevent attacks*" [7]. In fact, the method protects only the comparison operation and is often used e.g. in SIM cards during PIN verification. Our attacks do, however, not target the comparison operation but the time instants when the secrets are manipulated. Since CryptoMemory manipulates the secret key(s) *before* it decreases AAC, the counters can be easily bypassed with a reset. A more secure handling of the AACs could be to decrease the counter right upon reception of a mutual authentication command, and *prior* to the reception of the payload data and computation of the authenticators.

Protecting against more sophisticated attacks than ours may require to implement some of the well-known countermeasures against power analysis attacks, e.g. noise generators and power filters at the hardware level, or masking, random delays, and shuffling at the circuit level [25].

6 Conclusions

CryptoMemory is advertised to be used as a secure element in larger systems, in a way that only authorized hosts with knowledge of the correct authentication keys can have access to the protected memory contents. It is ensured that CryptoMemory "*permanently lock these values* [secret keys] *after device personalization and guarantee these values can never be read*" [8].

In this work we have shown how to extract such keys by using well-understood power analysis techniques. CryptoMemory uses fuses to lock access control policies, access control to protect memory contents, and AACs to strengthen access control. Therefore, a large part of CryptoMemory's security relies on the AACs, that we identified as a not sufficiently protected point of failure.

Acknowledgments. This work was supported in part by the European Commissions ECRYPT II NoE (ICT-2007-216676), by the Belgian States IAP program P6/26 BCRYPT, and by the Research Council K.U. Leuven: GOA TENSE (GOA/11/007). Josep Balasch and Roel Verdult are funded by a PhD grant within the covenant between the universities K.U. Leuven and R.U. Nijmegen. Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

References

1. AT88SC0204 ChipResetter, `http://chipreset.atw.hu/6/index61.html`
2. Coinamatic, `http://www.coinamatic.com`
3. ISO/IEC 7816-3: Identification cards - integrated circuit(s) cards with contacts - part 3: Electronic signals and transmission protocols (1997)
4. Labgear HDSR300 High Definition Satellite Receiver. User Guide, `http://www.free-instruction-manuals.com/pdf/p4789564.pdf`
5. Anderson, D.: Understanding CryptoMemory - The World's Only Secure Serial EEPROM, `http://www.atmel.com/atmel/acrobat/doc5064.pdf`

6. Atmel. CryptoMemory features, http://www.atmel.com/microsite_cryptomemory/iwe/index.html?source=tout_other2
7. Atmel. CryptoMemory Specification, http://www.atmel.com/atmel/acrobat/doc5211.pdf
8. Atmel. CryptoMemory Powerful Security at Low Cost, http://www.atmel.com/atmel/acrobat/doc5259.pdf
9. Atmel. CryptoRF Specification, http://www.atmel.com/atmel/acrobat/doc5276.pdf
10. Atmel. News Release, http://www.cryptomemorykey.com/pdfs/AtmelPR.pdf
11. Atmel. Secure Memory with Authentication AT88SC153, http://www.atmel.com/atmel/acrobat/doc1016.pdf
12. Atmel. Secure Memory with Authentication AT88SC1608, http://www.atmel.com/atmel/acrobat/doc0971.pdf
13. Atmel Corporation. Plug-and-Play Crypto Chip for Host-Side Security, http://www.atmel.com/dyn/corporate/view_detail.asp?ref=&FileName=Cryptocompanion_2_26.html&SEC_NAME=Product
14. Benhammou, J.P., Jarboe, M.: Security at an affordable price. Atmel Applications Journal, 29–30 (2004)
15. Biryukov, A., Kizhvatov, I., Zhang, B.: Cryptanalysis of the Atmel Cipher in SecureMemory, CryptoMemory and CryptoRF. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 91–109. Springer, Heidelberg (2011)
16. Bogdanov, A.: Linear Slide Attacks on the KeeLoq Block Cipher. In: Pei, D., Yung, M., Lin, D., Wu, C. (eds.) Inscrypt 2007. LNCS, vol. 4990, pp. 66–80. Springer, Heidelberg (2008)
17. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
18. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
19. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Shalmani, M.T.M.: On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 203–220. Springer, Heidelberg (2008)
20. Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Schreur, R.W., Jacobs, B.: Dismantling MIFARE Classic. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 97–114. Springer, Heidelberg (2008)
21. Garcia, F.D., van Rossum, P., Verdult, R., Schreur, R.W.: Dismantling SecureMemory, CryptoMemory and CryptoRF. In: Keromytis, A., Shmatikov, V. (eds.) Proceedings of ACM CCS 2010, pp. 250–259. ACM Press (2010)
22. Kasper, M., Kasper, T., Moradi, A., Paar, C.: Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 403–420. Springer, Heidelberg (2009)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
24. Lee, J., Pahl, N.: Bypassing Smart-Card Authentication and Blocking Debiting: Vulnerabilities in Atmel CryptoMemory based Stored-Value Systems. DEFCON 18 (2010)
25. Messerges, T.: Power analysis attack countermeasures and their weaknesses. In: CEPS Workshop (2000)

26. Moradi, A., Barenghi, A., Kasper, T., Paar, C.: On the Vulnerability of FPGA Bitstream Encryption against Power Analysis Attacks Extracting Keys from Xilinx Virtex-II FPGAs. In: Danezis, G., Shmatikov, V. (eds.) Proceedings of ACM CCS 2011, pp. 111–124. ACM Press (2011)
27. Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: Proceedings of USENIX 2008, pp. 185–193. USENIX Association (2008)
28. NVIDIA. Checklist for Building a PC that Plays HD DVD or Blu-ray Movies, ftp://download.nvidia.com/downloads/pvzone/Checklist_for_Building_a_HDPC.pdf
29. Oswald, D., Paar, C.: Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 207–222. Springer, Heidelberg (2011)
30. Hearst Electronic Products. Microsoft Zune HD 16GB, what's inside, http://www2.electronicproducts.com/Microsoft_Zune_HD_16GB-whatsinside_text-89.aspx
31. Tektronix. DPO7000C Oscilloscope Series, http://www.tek.com/products/oscilloscopes/dpo7000/
32. Viksler, H.: Web Laundry (In)Security, http://ihackiam.blogspot.com/2010/09/web-laundry-insecurity.html
33. Xilinx. XUP Virtex-II Pro Development System User Manual, http://www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf