Exploiting the Physical Disparity: Side-Channel Attacks on Memory Encryption

Thomas Unterluggauer(✉) and Stefan Mangard

Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{thomas.unterluggauer,stefan.mangard}@iaik.tugraz.at

Abstract. Memory and disk encryption is a common measure to protect sensitive information in memory from adversaries with physical access. However, physical access also comes with the risk of physical attacks. As these may pose a threat to memory confidentiality, this paper investigates contemporary memory and disk encryption schemes and their implementations with respect to Differential Power Analysis (DPA) and Differential Fault Analysis (DFA). It shows that DPA and DFA recover the keys of all the investigated schemes, including the tweakable block ciphers XEX and XTS. This paper also verifies the feasibility of such attacks in practice. Using the EM side channel, a DPA on the disk encryption employed within the ext4 file system is shown to reveal the used master key on a Zynq Z-7010 system on chip. The results suggest that memory and disk encryption secure against physical attackers is at least four times more expensive.

Keywords: Memory encryption · Side-channel attack · Power analysis · DPA · Fault analysis · DFA · Ext4

1 Introduction

Many electronic computing devices nowadays contain and process sensitive data in hostile environments. Among two particularly relevant examples, the first are engineering companies whose production machines are shipped around the world. These machines contain high-value intellectual property, e.g., control parameters and source code, that their vendors wish to be protected from unauthorized access and proliferation. Similarly, malicious access and modification must be prevented if usage statistics are used for billing. The second example are employee smart phones or laptops containing corporate secrets. Unattended such devices are a highly interesting target for industrial espionage and therefore need protection mechanisms.

In both examples, adversaries interested in the sensitive data potentially have physical access to the device. To prevent these attackers from simply reading confidential information from main or external memory, e.g., hard disks and memory cards, encryption of memory is well established. Several dedicated encryption

F.-X. Standaert and E. Oswald (Eds.): COSADE 2016, LNCS 9689, pp. 3–18, 2016.
DOI: 10.1007/978-3-319-43283-0_1

modes for memory, such as Cipher-Block-Chaining with Encrypted Salt-Sector IV (CBC-ESSIV) [10], Xor-Encrypt-Xor (XEX) [25], and XEX-based Tweaked codebook mode with ciphertext Stealing (XTS) [1], were proposed to fulfill the special requirements of memory encryption. These successfully prevent a variety of attacks, ranging from simple dumps of memory cards or hard disks to bus probing and cold boot attacks [14], and are thus implemented in an increasing number of real-world applications, such as `dm-crypt`, Mac OS X, Android, and ext4.

However, one important aspect contemporary memory encryption schemes left unconsidered are physical attacks such as side-channel and fault attacks. These allow the adversary to learn about secret key material used during encryption from various side channels, e.g., power, timing, and Electromagnetic Emanation (EM), or from faulty computations due to intentionally induced faults, e.g., clock glitches. Given physical access of the adversary as the motivating threat for memory encryption, physical attacks must not be neglected as these would allow adversaries to learn the encryption key and thus to decrypt confidential data in memory. The consideration of physical attacks is particularly important for permanently running devices that are threatened by attackers without any time constraints, e.g., a corporate customer may be interested in the data and IP of an embedded control unit within a purchased production machine.

This paper therefore investigates contemporary memory encryption schemes and their implementation within `dm-crypt`, Android 5.0, Mac OS X and ext4 in terms of physical attacks. As one main result, our detailed analysis shows that Differential Power Analysis (DPA) and Differential Fault Analysis (DFA) breaks all contemporary memory and disk encryption schemes used in practice. Most prominently, it presents tricks to be applied to DPA and DFA in order to obtain the keys from the tweakable ciphers XEX and XTS. Supporting the analysis results, our second contribution exploits the EM side channel of a Zynq-7010 system on chip in a practical attack on the recently introduced ext4 disk encryption mechanism that completely discloses the confidential disk content. We thus conclude that securing memories against physical adversaries by using contemporary memory encryption requires protected implementations, e.g., [5, 16, 22], that increase the cost of memory encryption at least by a factor of four.

This paper is organized as follows. Section 2 introduces memory encryption and gives an overview on common state-of-the-art implementations. The memory encryption schemes are analyzed with respect to both DPA and DFA in Sect. 3. The practical feasibility of such attacks is evaluated in Sect. 4, and Sect. 5 concludes the paper.

2 Memory Encryption

Memory encryption deals with the encryption of data contained in memory such as RAM, memory cards and hard disks. However, in practice different variants and notations are being used for memory encryption. This Section therefore defines memory encryption and gives an overview on common memory encryption schemes and implementations.

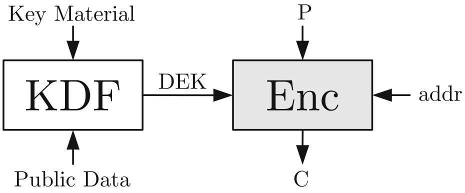


Fig. 1. Generic model of memory encryption.

2.1 Definition

The encryption of memory is usually performed using dedicated memory encryption schemes as these schemes have to fulfill several requirements: (1) ensure random access to all memory blocks, (2) provide sufficiently fast bulk encryption, and (3) the only information an adversary can derive from the encrypted memory is whether a memory block has changed or not.

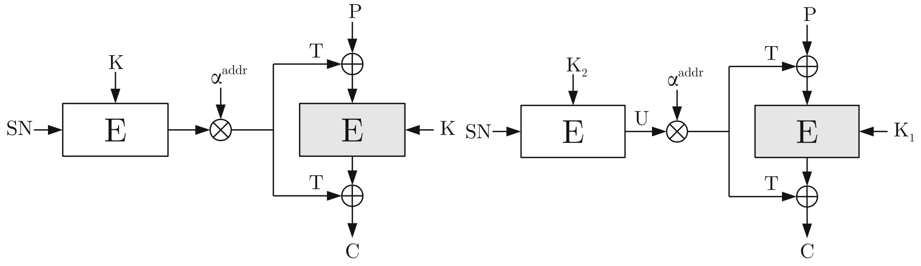
Definition 1. *A memory encryption scheme is an encryption scheme Enc* : $\mathcal{K} \times \mathcal{A} \times \{0,1\}^n \rightarrow \{0,1\}^n$*, which*

- *uses a key K from key space $\mathcal{K}$, and*
- *splits the memory into $s = \lceil \frac{size_{memory}}{n} \rceil$ n-bit memory blocks,*
- *identifies each of the memory blocks by their address in address space $\mathcal{A}$, and*
- *provides address-dependent en-/decryption for each of these memory blocks.*

Definition 1 considers the encryption of a flat memory space and requires the encryption process to incorporate address information. The address information allows memory encryption schemes to fulfill requirement (3) as for this reason each memory block is encrypted differently. Otherwise, it would be easily recognizable if certain data is contained in different memory locations and valid (but encrypted) data could simply be copied to different addresses (*splicing attack* [9]). The requirements (1) and (2) are typically satisfied by splitting the memory space into blocks using two different granularities: the memory is divided into larger sectors (or pages) and each sector (or page) is divided into encryption blocks. The encryption mode then ensures fast bulk encryption within each sector and random access on sector level.

2.2 Memory Encryption in Practice

In practice, memory encryption is often named disk encryption referring to the type of memory used. There are two variants of disk encryption: (1) *block device* or *full disk encryption*, and (2) *file-level disk encryption*. While *full disk encryption* performs encryption directly on the raw memory space of a whole disk, block device, or partition, i.e., beneath a file system, *file-level disk encryption* performs encryption on file level on top of or within a file system. Both variants use the same sort of memory encryption schemes, but apply them to different



(a) XEX mode.

(b) XTS mode.

Fig. 2. Tweakable ciphers for disk encryption.

portions of the memory. Throughout the paper, the term memory encryption thus denotes any of these variants.

Another aspect of practical implementations of memory encryption is that they usually employ a Key Derivation Function (KDF) to derive the Data Encryption Key (DEK) to be used within the memory encryption scheme from, e.g., a user password and public nonces. The combination of such a KDF and a memory encryption scheme leads to the generic model of memory encryption in Fig. 1. The following will use this model to first describe typical schemes for both the KDF and the encryption part, and will then show how these are used in several practical implementations.

Key Derivation Functions. To derive a key from a user password or a PIN, typically a password hashing function such as PBKDF2 [18] or scrypt [23] is used. This password-derived key is then mostly used as a Key Encryption Key (KEK) to decrypt the actual master key MK of the memory using an ordinary block cipher. Depending on the concrete setup, such master key MK is directly used as the DEK for the memory encryption scheme or is used to further derive or decrypt keys, e.g., DEKs for the encryption of single files in file-level disk encryption.

Encryption Schemes. Common implementations exclusively deal with the encryption of external memory, e.g., hard disks. These implementations, e.g., in `dm-crypt`, mainly utilize the modes XEX [25], XTS [1], and CBC with ESSIV [10]. The tweakable block ciphers XEX and XTS are shown in Fig. 2. Both encryption modes apply a tweak T to the cipher E that results from a binary-field multiplication of the encrypted sector number with the memory block address. While XEX uses only one key, XTS uses two different keys for the two instances of the cipher. The CBC mode with Encrypted Salt-Sector IV (ESSIV) is depicted in Fig. 3. ESSIV ensures a secret IV and thus prevents watermarking attacks [27]. It computes the IV as the encryption of the sector number with the hashed key (i.e., salt).

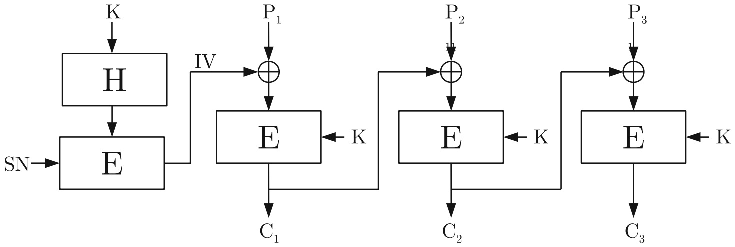


Fig. 3. Disk encryption via CBC and ESSIV.

Differently, research on the design and construction of secure systems further considered the encryption of the main memory. Primarily variants of the counter mode encryption were proposed such as in Fig. 4 [26,29]. The pad is the encryption of a block-specific seed that comprises an Initial Vector (IV), the memory block address, and a timestamp (or counter). It is mostly favored due to the little latency it introduces on the path to the memory.

2.3 State-of-the-Art Implementations

The following presents common implementations within `dm-crypt`, Android, Mac OS X, and ext4, and shows that the memory encryption schemes presented before have high prevalence throughout all of these implementations.

dm-crypt. `dm-crypt` [2] is a disk encryption utility that provides transparent encryption of arbitrary block devices within Linux ≥ 2.6, i.e., block device encryption. `dm-crypt` can be configured to use one of several available encryption modes, i.a., CBC-ESSIV and XTS (default), using different block ciphers, e.g., AES-128 [8]. The utility requires the user to supply the block device DEK when mounting the block device. For more convenient usage, however, Linux Unified Key Setup (LUKS) [11] can be used. LUKS adds a meta-data header to the block device that stores the encrypted DEK. The respective KEK is derived from a user password using PBKDF2.

Mac OS X. Mac OS X from version 10.7 (Lion) onwards provides block device encryption using the tool `FileVault 2` [3,7]. Mac OS X encrypts block devices using XTS and AES-128 with separate DEKs that are chosen randomly upon setup of each encrypted block device. For key storage, Mac OS X uses a three-tier hierarchy of DEKs, KEKs and Derived Key Encryption Keys (DKEK). The DEK is encrypted using a randomly chosen KEK that is encrypted using at least one DKEK. DKEKs can, e.g., be derived from a password or be the public key of a corporate certificate. Both the DEK and the KEK are stored encrypted in a meta-data block on the block device.

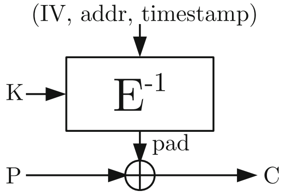


Fig. 4. Counter mode memory encryption.

Android. Android is equipped with full disk encryption for devices such as flash memory. In Android 5.0, encryption of block devices is based on `dm-crypt` that is configured to use AES-128 and CBC-ESSIV [13]. Its DEK is sized 128 bits by default and stored encrypted on the block device. The respective KEK is derived from a user password and a hardware-bound key using scrypt and a signing procedure within a Trusted Execution Environment (TEE).

Ext4. Since Linux 4.1, the ext4 file system offers file-level disk encryption [20,21]. It allows to set up encryption for a specific folder that is assigned a master key derived from a user passphrase and a salt using PBKDF2. While ext4 encrypts file content and names, meta data and file system structure is available in plaintext. Each file uses an individual DEK that is derived from the master key MK and a file nonce N_f using AES-128 in ECB mode, i.e., $DEK_f = E_{N_f}(MK)$. The respective nonce N_f is stored in the file's meta-data section. The file DEK is used to encrypt the file contents using XTS and AES-128.

3 Physical Attacks on Memory Encryption

Physical access as the motivation for memory encryption and the prevalence of the memory encryption schemes from Sect. 2 necessitate their analysis with respect to physical attacks such as side-channel and fault attacks. The following analysis of memory encryption schemes w.r.t. physical attacks shows that both DPA [19] and DFA [4] attacks are easily capable of breaking all the schemes presented, i.e., they reveal the DEK that allows to decrypt all memory content. Most remarkably, it demonstrates how to obtain the AES-128 keys in the tweakable block ciphers XEX and XTS with practical complexity.

3.1 Differential Power Analysis

DPA attacks and its variants, e.g., Correlation Power Analysis [6], are methods that allow recovery of an encryption key based on power measurements or similar, e.g., EM. The typical procedure is to measure the power of n different en-/decryptions for known plain- or ciphertext, to compute certain intermediate values within the en-/decryption based on the n different plain-/ciphertexts and the possible keys, and to map the intermediate values to hypothetical power consumptions according to a leakage model. Correlation of the power traces of the n en-/decryptions with the respective hypothetical power consumptions reduces the key space or determines the key uniquely. The following details DPA attack scenarios on the schemes from Sect. 2.

XEX Mode. The tweak T makes sure that the block cipher behaves differently for each memory address. In spite of this, DPA-style attacks are applicable with little modifications. Therefore, the adversary focuses on one particular memory block, i.e., fixed sector and fixed memory address. For this memory block, the adversary observes ciphertexts and power traces of several encryption processes. The captured power traces are then used twice to attack different rounds of the block cipher shaded gray in Fig. 2a, as the following illustrates for AES-128:

1. From an attacker's point of view, the last round key rk_{10} is blinded with the tweak T. However, for a fixed sector and memory address, the tweak T is constant. A DPA that targets the input of the last round's SBox will thus reveal the last round key xored with the tweak, i.e., $rk_{10} \oplus T$.
2. Knowledge of $rk_{10} \oplus T$ is sufficient to target the input of the second-last round's SBox in a second DPA. It reveals the second-last round key rk_9, which can be used to compute the key K.

Two consecutive DPAs on the same set of traces allow to gain knowledge of the key K. The DPAs disclose the information contained in all memory blocks across all sectors, even though only one particular block in one specific sector is actually attacked.

Note that besides standard DPA, also unknown plaintext template attacks [15] are applicable to directly obtain rk_{10}. However, such attacks require a preceding profiling step to create suitable templates. Alternatively, if the adversary additionally has knowledge of the accessed sector, e.g., from the observation of memory addresses on the bus, the attack generally becomes easier. In this case, the tweak computation that encrypts the sector number can be attacked to immediately learn K from power traces of memory accesses to different sectors. However, depending on the practical circumstances, either of those attacks is more suitable, e.g., the adversary may want to avoid raising suspicion by not probing the memory bus.

XTS Mode. Contrary to XEX, a successful DPA on XTS requires the knowledge of the accessed sector number. It allows to first obtain K_2 from the tweak computation. Once K_2 is known, the tweak T used for encrypting any memory block can be computed which enables a straight-forward attack on the key K_1 by monitoring the power consumption during arbitrary memory accesses.

Counter Mode. Known-plaintext scenarios allow for DPA attacks that recover the key K in counter mode encryption. They facilitate the computation of the encryption pads from both known plain- and ciphertexts and thus DPA on the last round of the cipher. Typically, plaintexts would be assumed to be unknown since memory encryption is applied. However, known-plaintext scenarios will certainly occur in memory encryption. One such case would be publicly known (or observable) data that is sent to a device, e.g., via external interfaces, and that is consecutively encrypted and stored in main memory, e.g., within an input buffer.

If there are insufficiently many known plaintexts, a known input seed also allows for a DPA - one that does not even require any ciphertext. Often, the counters and addresses within the seed will be publicly accessible (or observable). If the IV is public as well, the seed will be fully known and a DPA in the first round of the cipher be possible. The IV will mostly be stored publicly on the disk for disk encryption, but might be chosen randomly at startup and remain inaccessible for encryption of the main memory. Still, the approach in [17], where a DPA is performed on the counter mode of AES without knowledge of the counter value, might be applicable.

CBC Mode with ESSIV. Independently of the initial vector derivation, DPA attacks on the CBC mode are trivially possible through the observation of ciphertexts and power traces of the respective encryption processes. The recovered key K then allows to compute each sector's IV (ESSIV) and hence to obtain any plaintext.

3.2 Differential Fault Analysis

Differential Fault Analysis (DFA) [4] describes techniques that use algebraic properties of ciphers to find out about the key from one correct and one or several faulty cipher invocations with the same input. Various techniques to inject faults into a device exist, e.g., power and clock glitches, laser shots, and electromagnetic pulses. However, the following investigation does not consider how the faults are injected, but elaborates on how faults are exploited in order to obtain the key. It details DFA attack scenarios on the schemes from Sect. 2, and most noteworthy, how to break the tweakable block ciphers XEX and XTS with practical complexity 2^{35} if AES-128 is used.

XEX Mode. The attack procedure of DFA to learn the key K is tightly linked with the employed cipher. Exemplarily, we show how to use DFA to extract the key from AES-128 in XEX mode. The DFA targets the block cipher that is shaded gray in Fig. 2a and consists of two basic steps:

1. An arbitrary byte fault in round 8 is used to extract the xor of round key 10 and the tweak $(rk_{10} \oplus T)$.
2. A byte fault in round 7 and a modified representation of the AES round function lead to round key 9 and thus the key K.

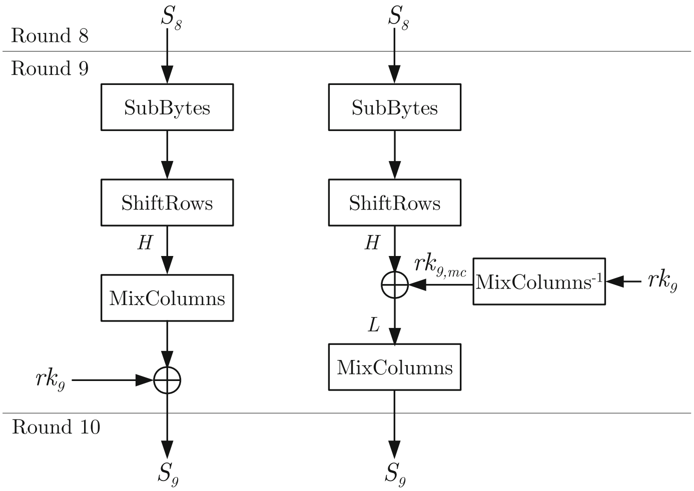


Fig. 5. AES round function (left) and its alternative representation (right).

Learning $rk\_10 \oplus T$. From an attacker's point of view, the last round key rk_10 is blinded with the tweak T. This requires the tweak T to be constant for DFA, i.e., the attack operates on fixed sector and fixed memory block. By forcing reencryption of the same plaintext in the desired block, the adversary gets the chance to inject an arbitrary byte fault during round 8 of the encryption process of the tweakable cipher. Application of a suitable DFA technique, e.g., [24,28], to the pair of right and faulty ciphertext results in the value $rk\_10 \oplus T$.

Learning round key 9. The DFA to learn rk_9 benefits from an alternative representation of the AES round function. As shown in Fig. 5, it is obtained from swapping MixColumns and AddRoundKey. The linearity of MixColumns allows this transformation if the round key is modified accordingly, i.e.,

$$\begin{aligned}\text{MixColumns}(H) \oplus rk_9 &= \text{MixColumns}(H \oplus \text{MixColumns}^{-1}(rk_9)) \\ &= \text{MixColumns}(H \oplus rk_{9,mc}).\end{aligned}$$

In the following, the alternative representation of the round function is used for round 9. The attack starts by injecting a random byte fault during round 7. As the MixColumns operation propagates the fault to the other state bytes, all bytes are affected by the end of round 8. The observed pair of right and faulty ciphertext C, C' and the value $rk_{10} \oplus T$ are used to compute backward to obtain the respective values L, L' in round 9.

Interpreting L, L' as a pair of right and faulty ciphertext, the remaining cipher looks like a round-reduced version of the AES with one inner round missing. The last round consists of AddRoundKey, ShiftRows, and SubBytes and uses the round key $rk_{9,mc}$. The benefit of this approach is that now any DFA technique that targets the last round key of the AES, e.g., [24,28], is suitable to obtain $rk_{9,mc}$ from the pair L, L' and the fault differences at the end of round 8. Round key 9 is then easily computed as $rk_9 = \text{MixColumns}(rk_{9,mc})$.

If the technique in [28] is used to learn $rk_{9,mc}$, the attack has the complexity 2^{34} and thus is clearly possible on nowadays' computers. According to [28], the required faults can be injected by temporal overclocking only.

XTS Mode. Although XTS using AES-128 relies on two 128-bit keys, DFA breaks this mode with total complexity 2^{35}. First, the DFA technique that was just applied to XEX trivially recovers the key K_1 with complexity 2^{34}. Second, the following small trick uses faults in the tweak computation to also learn K_2 with complexity 2^{34}. It determines the faulty tweak T' from the observed faulty ciphertext C' and the correct tweak T.

The procedure to recover K_2 requires the values of K_1, P, and $rk_{1,10} \oplus T$ to be known, where $rk_{1,10}$ denotes round key 10 derived from K_1. These preconditions usually apply if the previous DFA on XEX was utilized to learn K_1. As a result, the tweak T and the intermediate value U (cf. Fig. 2b) can be computed: $U = \alpha^{-addr} \cdot T$. A random fault that is injected in one byte of the state in round 9 of the AES affects four bytes of U. Although the respective faulty U' is not directly observable, it can be brute-forced with complexity 2^{32}. This is done by trying all values for the faulty bytes of U', computing the respective tweaks T', encrypting the original plaintext P using T' and K_1, and matching the result against the faulty ciphertext C'. Once U' is known, four bytes of $rk_{2,10}$ (round key 10 derived from K_2) are revealed using the technique in [24]. Hereby, the possible key space for $rk_{2,10}$ is reduced by the possible differences that can be observed at the output of MixColumns in round 9 that result from a single byte fault during round 9. Similarly, three more faults in different bytes of the state of round 9 recover the remaining 12 bytes of $rk_{2,10}$ and thus K_2.

Counter Mode. DFA on a block cipher operated in counter mode (cf. Fig. 4) requires access to the output of the cipher, i.e., the pad. Since encryption pads must not repeat, consecutive encryptions of plaintexts will not use the same pad and encryption seed. As a result, DFA is limited to the decryption process. If the same ciphertext is loaded from the same memory address several times and the adversary can inject faults during the pad computations and observe the respective plaintexts, the correct and faulty pads can be computed and the master key K be learned via a suitable DFA technique. The required plaintexts may be observed from communication of the device via external interfaces.

CBC Mode with ESSIV. Independently of the initial vector derivation, DFA is trivially possible by restricting analysis to one specific memory block within the CBC chain of one particular sector. Therefore, reencryption of the same plaintext has to be triggered for the desired memory block, e.g., through placing the same message in an input buffer by repeatedly sending the same message to the device. Faults injected during reencryption are directly observable in the resulting ciphertext. This facilitates the application of a suitable DFA technique in order to learn the master key K. Note that for this to work, all memory blocks in the sector prior to the target block must not change during reencryption.

4 EM Attack on Ext4 Encryption

As our analysis points out, contemporary memory encryption schemes are clearly vulnerable to physical attacks. However, it remains to show that such attacks are indeed feasible on contemporary systems. This Section therefore demonstrates a practical attack on the disk encryption scheme incorporated into the ext4 file system. The EM attack conducted on a Zynq Z-7010 system on chip (SoC) reveals the used master key and thus all content by exploiting the leakage of the first round of an AES execution.

4.1 Analysis of Ext4 Disk Encryption

Disk encryption within the ext4 file system works on file level and allows to encrypt arbitrary directories using a specified master key MK. For each file in such directory, the master key MK is used to derive an individual data encryption key DEK_f to encrypt the respective file's content and name. Key derivation is done by encrypting MK with AES-128 in ECB mode using a public file nonce N_f as the key. It starts whenever DEK_f is needed and not already present in main memory. The size of both MK and DEK_f is 512 bits and chosen such as to be able to encrypt files with AES-256 in XTS mode in future versions. However, currently only AES-128 in XTS mode is supported and thus the last 256 bits of DEK_f and MK are not used. The file nonce N_f is stored in an extended attribute of the file's inode.

Clearly, given the master key MK and a public file nonce N_f, the respective file key DEK_f can be derived. However, the key derivation chosen in ext4 also allows to compute the master key MK given any DEK_f and the respective nonce N_f. Therefore, an attacker who wants to learn MK using power analysis can choose between two equivalent targets, namely (1) data encryption of file content, and (2) the derivation of the file key DEK_f. In terms of target (1), the strategy from Sect. 3 can be straight-forwardly applied, but one may need files that are sufficiently large to be able to learn K_2 within XTS. With respect to target (2), one needs to monitor accesses to many different files as such trigger key derivations. To practically verify the feasibility of attacks on disk encryption, we opted for target (2).

4.2 General Attack Flow

The attack we performed assumes an encrypted folder on an SD card using the ext4 file system. It further assumes the attacker is able to trigger the creation of new files within the encrypted folder via external interfaces, e.g., by uploading data via a running web server or writing log files.

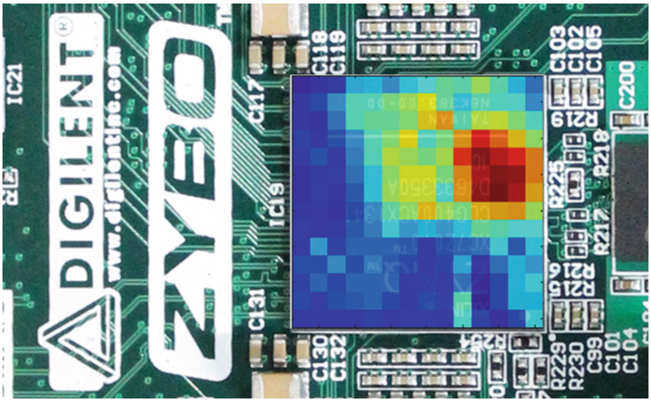


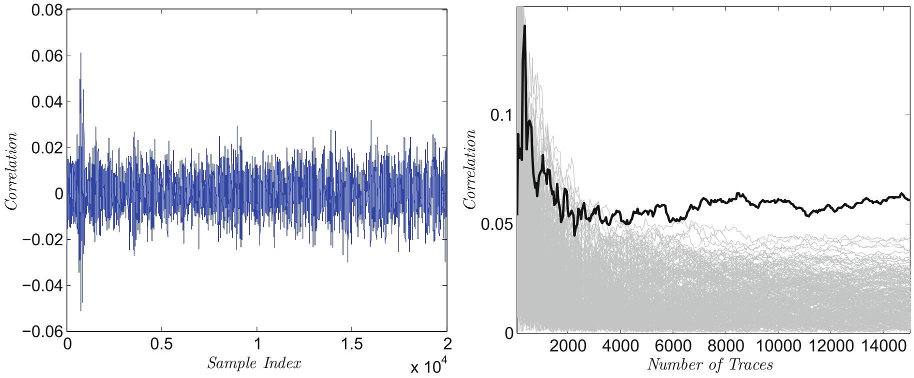
Fig. 6. Distribution of t-test results on the chip surface.

To perform the attack, the attacker first dumps the (encrypted) content of the SD card. They may not be able to read the actual content from such file system dump, but can learn about the directory structure as meta data is not encrypted. Second, the attacker triggers the creation of sufficiently many files on the SD card, observes the EM side channel, and stores the respective EM traces. Third, the attacker again dumps the content of the SD card. By comparing its content with the initial dump from before the measurements, the attacker can learn which files have been created. The meta data of the newly created files allows to both learn the used nonces N_f and their creation date, which in turn allows to map the newly created files on the SD card to the EM traces. In the next step, the attacker creates the power model for the key derivation, i.e., $DEK_f = E_{N_f}(MK)$. Finally, the power model is matched with the EM traces to reveal the master key.

To investigate the encrypted directory in the file system, debugging and forensic tools are highly suitable. We used the tool `debugfs` to find new files in the file system and to learn their creation date and the respective nonces. Note that the access times are also available within the file system, which allows for the described attack also when monitoring arbitrary file accesses.

4.3 Experimental Setup and Results

The feasibility of the attack on ext4 encryption in Sect. 4.2 was verified using the Digilent ZYBO board. The board hosts a Xilinx Zynq Z-7010 SoC, 512 MB of DDR3 RAM, and several IO interfaces, i.a., an SD card slot. The Zynq Z-7010 SoC combines an Artix-7 FPGA and a state-of-the-art hard macro comprising a 650-MHz dual-core ARM Cortex-A9 processor, IO modules, and memory controllers. The measurement devices required to capture the EM traces involved a LeCroy WavePro 725Zi oscilloscope, a Langer RF B 3-2 magnetic field probe, and a Langer PA 303 pre-amplifier.



(a) Time domain (correct key).

(b) Key hypotheses.

Fig. 7. Single-byte correlation results for ext4 key derivation.

The general leakage behavior of the Zynq Z-7010 was examined by running the AES T-table implementation included in the Linux 4.3 kernel in a bare-metal application. Therefore, the EM probe was placed in different locations using a stepper table to evaluate a fixed vs. random t-test. This revealed the spots of high leakage as shown in Fig. 6 and allowed for successful DPA on the bare-metal AES.

The setup for the complete disk encryption scenario was established by configuring the Zynq SoC to use a 350-MHz memory clock and a 625-MHz CPU clock and deploying Linux 4.3 to the ZYBO board. An ext4 file system was created on an SD card and one directory encrypted such that it is only readable by the system running on the ZYBO board. The attack procedure from Sect. 4.2 was executed by repeatedly creating new files via the UART interface. The oscilloscope was triggered to capture an EM trace at 5 GS by setting a GPIO pin just before creating a new file. The SD card content was then analyzed on a PC using `debugfs`, the EM traces aligned, and a DPA performed on the SBox output of the first AES round using the Hamming Weight power model.

The results of the DPA on a single byte of the master key are given in Fig. 7. Using 15,000 EM traces, Fig. 7a clearly presents the correlation of the power model of the correct key guess in the time domain. Moreover, in Fig. 7b the correct key byte (black) is clearly distinguished from the remaining key hypotheses with 5,000 measurements.

In this feasibility study, the Linux kernel was reconfigured to omit symmetric multiprocessing, dynamic frequency scaling, and caches. Moreover, AES executions were highlighted in the captured EM traces through another hardware-triggered signal to help finding AES executions. This however does not affect the applicability of the attack. For example, [12] showed the practicality of attacking a free-running OpenSSL implementation of AES with active caches and frequency scaling on the TI Sitara platform that uses an ARM Cortex-A8.

However, further improvement of both setup and trace processing would definitely be interesting future work.

5 Conclusion

Summarizing, this paper unveiled that contemporary mechanisms that aim to ensure the confidentiality of memory content in the presence of adversaries with physical access are clearly vulnerable to physical attacks. In particular, it showed that all common implementations of memory and disk encryption schemes can easily be broken using DPA and DFA. The attacks are powerful enough to even break the tweakable cipher XTS that is most commonly used. Further, the feasibility of such attacks on state-of-the-art computing systems was verified by exploiting the EM side channel on the Zynq Z-7010 SoC. The attack revealed the master key of the disk encryption scheme incorporated into the ext4 file system and thus all encrypted content.

Our results suggest that if memory encryption is supposed to use current schemes in the future, cipher implementations with appropriate countermeasures must be used. However, the secure cipher implementations proposed so far were mainly designed for the use in embedded devices and might thus not yield the desired throughput for memory encryption. For example, the 1st-order threshold implementations in [5,22] require 246 and 266 clock cycles for one AES execution, respectively. Additionally, these implementations add an area overhead of a factor of four that must hence also be expected for secure memory encryption based on such protected implementations. It thus remains future work to implement memory encryption that fulfills both the requirement for sufficient throughput and security against side-channel adversaries.

Acknowledgments. This work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 845579 (MEMSEC).

References

1. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE Std 1619–2007, April 2008
2. Dm-crypt: Linux Kernel Device-Mapper Crypto Target (2015). http://www.saout.de/misc/dm-crypt/
3. Apple Inc.: Apple Technical White Paper: Best Practices for Deploying FileVault 2 (2012)
4. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
5. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A more efficient AES threshold implementation. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 267–284. Springer, Heidelberg (2014)

6. 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)
7. Choudary, O., Grobert, F., Metz, J.: Infiltrate the Vault: security analysis and decryption of lion full disk encryption. Cryptology ePrint Archive, report 2012/374 (2012). http://eprint.iacr.org/
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer, Heidelberg (2002)
9. Elbaz, R., Champagne, D., Gebotys, C.H., Lee, R.B., Potlapally, N.R., Torres, L.: Hardware mechanisms for memory authentication: a survey of existing techniques and engines. Trans. Comput. Sci. **4**, 1–22 (2009)
10. Fruhwirth, C.: New methods in hard disk encryption. Technical report (2005)
11. Fruhwirth, C.: LUKS On-Disk Format Specification (2011). https://gitlab.com/cryptsetup/cryptsetup/wikis/LUKS-standard/on-disk-format.pdf
12. Longo, J., De Mulder, E., Page, D., Tunstall, M.: SoC it to EM: ElectroMagnetic side-channel attacks on a complex System-on-Chip. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 620–640. Springer, Heidelberg (2015)
13. Google Inc.: Android Full Disk Encryption (2015). https://source.android.com/security/encryption/
14. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM **52**(5), 91–98 (2009)
15. Hanley, N., Tunstall, M., Marnane, W.P.: Unknown plaintext template attacks. In: Youm, H.Y., Yung, M. (eds.) WISA 2009. LNCS, vol. 5932, pp. 148–162. Springer, Heidelberg (2009)
16. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
17. Jaffe, J.: A first-order DPA attack against AES in counter mode with unknown initial counter. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 1–13. Springer, Heidelberg (2007)
18. Kaliski, B.: PKCS# 5: Password-based Cryptography Specification Version 2.0 (2000)
19. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 388. Springer, Heidelberg (1999)
20. Linux Kernel Organization Inc.: Linux Kernel 4.3 Source Tree (2015). https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/log/?id=refs/tags/v4.3
21. Halcrow, M., Savagaonkar, U., Ts'o, T., Muslukhov, I.: Ext4 Encryption Design Document. https://docs.google.com/document/d/1ft26lUQyuSpiu6VleP70_npaWdRfXFoNnB8JYnykNTg
22. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
23. Percival, C.: Stronger Key Derivation via Sequential Memory-Hard Functions. Self-published, pp. 1–16 (2009)
24. Piret, G., Quisquater, J.-J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)

25. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
26. Rogers, B., Chhabra, S., Prvulovic, M., Solihin, D.: Using address independent seed encryption and Bonsai Merkle trees to make secure processors OS- and performance-friendly. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007, pp. 183–196, December 2007
27. Saarinen, M.-J.O.: Encrypted watermarks and Linux laptop security. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 27–38. Springer, Heidelberg (2005)
28. Saha, D., Mukhopadhyay, D., RoyChowdhury, D.: A diagonal fault attack on the advanced encryption standard. Cryptology ePrint Archive, report 2009/581 (2009). http://eprint.iacr.org/
29. Suh, G., Clarke, D., Gasend, B., van Dijk, M., Devadas, S.: Efficient memory integrity verification and encryption for secure processors. In: 36th Annual IEEE/ACM International Symposium on Microarchitecture, Proceedings 2003, MICRO-36, pp. 339–350, December 2003