Securing Memory Encryption and Authentication Against Side-Channel Attacks Using Unprotected Primitives

Thomas Unterluggauer, Mario Werner, and Stefan Mangard
Graz University of Technology
Email: {firstname.lastname}@iaik.tugraz.at

ABSTRACT

Memory encryption is used in many devices to protect memory content from attackers with physical access to a device. However, many current memory encryption schemes can be broken using Differential Power Analysis (DPA). In this work, we present MEAS—the first Memory Encryption and Authentication Scheme providing security against DPA attacks. The scheme combines ideas from fresh re-keying and authentication trees by storing encryption keys in a tree structure to thwart first-order DPA without the need for DPA-protected cryptographic primitives. Therefore, the design strictly limits the use of every key to encrypt at most two different plaintext values. MEAS prevents higher-order DPA without changes to the cipher implementation by using masking of the plaintext values. MEAS is applicable to all kinds of memory, e.g., NVM and RAM, and has memory overhead comparable to existing memory authentication techniques without DPA protection, e.g., 7.3% for a block size fitting standard disk sectors.

Keywords

side-channel attacks; DPA; memory; encryption; authentication

1. INTRODUCTION

Memory encryption is the standard technique to protect data and code against attackers with physical access to a memory. It is widely deployed in state-of-the-art systems, such as in iOS [2], Android [15], Mac OS X [1], Windows [12], and Linux [26, 32]. Typical encryption schemes employed in these systems are Cipher-Block-Chaining with Encrypted Salt-Sector IV (CBC-ESSIV) [13], Xor-Encrypt-Xor (XEX) [37], and XEX-based Tweaked codebook mode with ciphertext Stealing (XTS) [21]. These schemes successfully prevent attackers from accessing memory content when the device is shut off and the encryption key is not present on the device, e.g., an encryped USB flash drive.

Contrary to that, in many situations in the Internet of Things (IoT), a physical attacker is in possession of a running device, or can turn a device on. In such cases, the attacker can, for example, observe and tamper with data in RAM. As a result, memory encryption and tree-based authentication techniques, e.g., Merkle trees [30], Parallelizable Authentication Trees [18] (PAT) and Tamper Evident Counter [11] (TEC) trees, are increasingly deployed to protect data in RAM. As one prominent example, RAM encryption and authentication was only recently adopted in consumer products with Intel SGX [17]. Similarly, there are efforts to encrypt RAM on AMD [23] and ARM systems [20] as well.

However, whenever a physical attacker has access to a running device, the attacker is also capable of performing side-channel attacks. This means that the attacker cannot just read and tamper with the memory, but is also capable of measuring side-channel information, such as the power consumption of the hardware, during the encryption and authentication of the memory. The attacker can then exploit such side-channel information to learn the secret key used for memory encryption and authentication. In practice, an attacker performing both passive, e.g., bus probing, and active, e.g., data spoofing, attacks on the memory, is also capable of observing side-channel information, e.g., by attaching an osilloscope for measuring the power, during the actual encryption or authentication process. As such, side-channel attacks are realistic for any physical attacker when given access to a running device. One particularly strong class of side-channel attacks is Differential Power Analysis [25] (DPA), which allows successful key recovery from observing the power consumption during the en-/decryption of several different data inputs. DPA attacks effectively accumulate side-channel information about the key being used by observing multiple en-/decryptions under the same key.

However, contemporary memory encryption and authentication schemes that protect memory against physical attackers, e.g., [10, 17, 34, 38, 42, 43], lack the consideration of side-channel attacks and DPA in particular. More concretely, the security of contemporary schemes is build upon the assumption of a microchip that is secure against active and passive adversaries and which does not leak any information about the key via side channels. However, as pointed out before, the assumption that side-channel attacks on microchips are infeasible is too strong. In fact, DPA attacks were quite recently shown to pose a serious threat to memory encryption on general-purpose CPUs. While the DPA presented in [45] breaks many contemporary memory encryption schemes, the practical attacks in [3, 27, 40, 45] document the feasibility of DPA on memory encryption and authentication on state-of-the-art systems.

ASIA CCS '17, April 02 - 06, 2017, Abu Dhabi, United Arab Emirates

ISBN 978-1-4503-4944-4/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3052973.3052985

In principle, there exist techniques to protect cryptographic primitives against DPA attacks. For example, an implementation can be protected by changing the hardware such as by applying masking techniques [8, 16], which use randomization to make the side-channel information independent from the actually processed value. However, protecting implementations of cryptographic primitives against DPA is expensive and a tough problem in an active field of research existing for almost two decades. The massive overheads for DPA-protected implementations range between a factor of four and a few hundred [4, 7, 33, 35] and would thus render current memory encryption and authentication schemes in latency sensitive applications impractical. In contrast, more efficient solutions are in sight when considering side-channel protection throughout the cryptographic design and looking for potential synergies.

Contribution. In this paper, we solve the problem of protecting data in memory against physical attackers in possession of a running device. More concretely, we solve the stringent problem of DPA attacks on memory encryption and authentication without additional memory overhead over conventional schemes.

We approach the topic with a detailed analysis of the security of fresh re-keying [24, 29] as a promising mechanism to prevent DPA on memory encryption. While re-keying completely thwarts DPA on the cryptographic key, our major result here is that re-keying provides merely first-order DPA security for the memory content itself. In particular, we show that the read-modify-write access patterns inevitably occuring in encrypted memory allow for profiled, higher-order DPA attacks that leak constant plaintext data when re-keying is applied to memory encrpytion.

Second, we build on our analysis and present MEAS—the first Memory Encryption and Authentication Scheme secure against DPA attacks. The scheme is suitable for all kinds of memory including random access memory (RAM) and non-volatile memory (NVM). By making use of synergies between fresh re-keying and authentication trees [11, 18, 30], MEAS simultaneously offers security against first-order DPA and random access to all memory blocks. In more detail, MEAS uses separate keys for each memory block that are stored in a tree structure and changed on every write access in order to strictly limit the use of each key to the encryption of two different plaintexts at most. For higher-order DPA security, MEAS performs data masking by splitting the plaintext values into shares and storing the encrypted shares in memory. This allows to flexibly extend DPA protection to higher orders in trade for additional memory. For all DPA protection levels, MEAS does not require DPA-protected implementations of the cryptographic primitives, making MEAS suitable for common off-the-shelf (COTS) systems equipped with unprotected cryptographic accelerators. However, MEAS is also an ideal choice for constructing a DPA-secure system from scratch as engineers do not have to cope with complex DPA protection mechanisms within the cipher implementation.

Finally, we show that memory encryption and authentication using MEAS adds protection against the very powerful DPA attacks, and still features the same memory overhead as state-of-the-art schemes which completely lack side-channel protection. For example, instantiating 4-ary, first-order DPA secure MEAS for standard disk architectures results in a very low memory overhead of 7.3%. Contrary to that, protecting cryptographic implementations against DPA to make use of state-of-the-art schemes would result in massive overheads making memory encryption and authentication infeasible.

Outline. This work is organized as follows. In Section 2, we first state our threat model and requirements, and we then discuss the state of the art on memory encryption and authentication. The state of the art on side-channel attacks and countermeasures is content of Section 3. We analyze the re-keying countermeasure in terms of memory encryption in Section 4 and use the results to present our first-order DPA secure MEAS in Section 5. Section 6 then presents data masking to achieve higher-order DPA security in MEAS. An evaluation of MEAS is done in Section 7 and we finally conclude in Section 8.

2. MEMORY ENCRYPTION AND AUTHENTICATION

The encryption and authentication of memory is an important measure to prevent attackers with physical access from learning and/or modifying the memory content. There are several schemes for memory encryption and authentication available, but none of them takes the risk of side-channel attacks into account.

In this section, we define two threat models: the *non-leaking chip model* restates the state of the art [10, 17, 34, 38, 42, 43], and the extended *leaking chip model* further takes side-channel leakage into account. Moreover, we summarize present techniques for memory encryption and authentication and its requirements.

2.1 Threat Model and Requirements

The *non-leaking chip model* in previous works assumes a single, secure microchip performing all relevant computations, e.g., a CPU. An attacker cannot perform any kind of active or passive attacks against this chip. All other device components outside this chip, e.g., buses, RAM modules and HDDs, are under full control of the adversary. Therefore, a physical attacker can, e.g., probe and tamper with buses, exchange peripherals, or turn the whole device on and off. For off-chip memory, this means that an attacker with physical access is capable of freely reading and modifying the memory content.

While reading can give an attacker access to confidential data stored inside the memory, modification breaks memory authenticity in several ways [10]: In *spoofing attacks*, an attacker simply replaces an existing memory block with arbitrary data, in *splicing attacks*, the data at address A is replaced with the data at address B, and in *replay attacks*, the data at a given address is replaced with an older version of the data at the same address.

Our *leaking chip model* extends the *non-leaking chip model* by considering passive side-channel attacks. It assumes that the microchip performing all relevant computations leaks information on the processed data via side channels, e.g., power and electromagnetic emanation (EM). Physical attackers can observe this leakage and perform side-channel attacks.

Hence, cryptographic schemes protecting the confidentiality and authenticity of off-chip memory in the *leaking chip model* have to fulfill three main requirements.

1. The only information an adversary can learn from memory is whether a memory block (i.e., ciphertext) has changed or not.
2. Prevention of spoofing, splicing, and replay attacks.
3. Protection against side-channel attacks.

In addition, fast random access to all memory blocks, high throughput (fast bulk encryption), and low memory overhead are desired.

2.2 Memory Encryption

Memory encryption schemes usually split the memory address space into blocks of predefined size, e.g., sector size, page size, or cache line size. Each of these blocks is then encrypted independently using a suitable encryption scheme. The partitioning of the address space into memory blocks aims to provide fast random access on block level and fast bulk encryption within the instantiated encryption scheme. Hereby, the chosen block size strongly affects possible trade-offs w.r.t. metadata overhead, access granularity, and speed.

So far, several memory encryption schemes have been proposed in the *non-leaking chip model* and are being used nowadays, e.g., the tweakable encryption modes XEX [37] and XTS [21], CBC with ESSIV [13], and counter mode encryption [38, 42].

2.3 Memory Authentication

Like for memory encryption, memory authentication schemes split the memory address space into blocks and aim for separate authentication of each of these blocks. Several memory authentication schemes have been proposed in the *non-leaking chip model*.

For example, a keyed Message Authentication Code (MAC) using the block address information can protect against spoofing and splicing attacks. However, it still allows for replay attacks. In order to protect against replay attacks, authenticity information must be stored in a trusted environment, e.g., in secure on-chip memory, that an attacker cannot modify. Authentication trees minimize this demand for secure on-chip storage, namely, only the tree's root is stored in secure memory, while the remaining tree nodes can be stored in public memory. Such trees therefore protect against spoofing, splicing, and replay attacks. Authentication trees over m memory blocks with arity a have logarithmic height $l = \log_a(m)$. Three prominent examples of authentication trees are Merkle trees [30], Parallelizable Authentication Trees [18] (PAT), and Tamper Evident Counter [11] (TEC) trees. We give a detailed description of them in Appendix A.

3. SIDE-CHANNEL ATTACKS

Present memory encryption and authentication schemes are designed to protect off-chip memory against adversaries with physical access assuming a microchip that is secure against all active and passive attacks. However, in IoT scenarios, the assumption that the microchip is secure against all passive attacks is often too strong since, in practice, a microchip running an algorithm leaks information on the processed data via various side channels, such as power, timing, and electromagnetic emanation (EM). This allows adversaries perform passive side-channel attacks, which can reveal secret keys that are used in cryptographic implementations. There exist two basic classes of passive side-channel attacks [25]: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). Originally, SPA and DPA have been introduced for the power side-channel, but their basic principle is is applicable to all kinds of side channels such as power, EM, and timing. Therefore, we will use the terms SPA and DPA throughout the paper, but note that our elaborations apply to all kinds of side channels.

3.1 Simple Power Analysis

In SPA attacks, the adversary tries to learn the secret value processed inside a device from observing side channels during a single processing of the secret value to be revealed, e.g., the adversary tries to learn the encryption key from a power trace observed during a single encryption. However, the adversary is allowed to observe the same encryption multiple times to reduce measurement noise. Clearly, an implementation that cannot keep a key secret for a single encryption is worthless. Therefore, bounded side-channel leakage for a single encryption and thus security against SPA attacks is a necessary precondition for any implementation.

3.2 Differential Power Analysis

Quite naturally, the amount of information learned about a secret value from side-channel information increases with the number of different processed inputs under the respective secret. This is exploited in DPA attacks, which use the observation of several different processings of a secret value in a device to learn its value, e.g., the adversary tries to learn the secret key from power traces observed during the en-/decryption of multiple (public) input values.

One important property of DPA attacks is their order. The order d of a DPA [25, 31] is defined as the number of d different internal values in the executed algorithm that are used in the attack. The attack complexity of DPA grows exponentially with its order [8].

3.3 Profiled Attacks

Independently of whether SPA or DPA is performed, side-channel attacks can make use of profiling. Profiling of a side-channel, e.g., the power consumption, means to construct templates [9] that classify the side-channel information of a target device with respect to a certain value processed inside the device. In the actual attack, the templates are matched with the side-channel trace to gain some information on the value processed inside the device. The information learned from template matching can then be exploited in either SPA or DPA manner. Note however that conducting profiled attacks requires much more effort than performing non-profiled attacks. Further note that in many applications it is impossible to perform the required profiling step at all.

3.4 DPA Countermeasures

The effectiveness of DPA attacks has caused a lot of effort to be put into the development of countermeasures to prevent DPA. Two basic approaches to counteract DPA have evolved, namely, (1) to protect the cryptographic implementation using mechanisms like masking, and (2) the frequent re-keying of unprotected cryptographic primitives.

3.4.1 Masking

Masking [8, 16], also called secret sharing, is a technique that can hinder DPA attacks up to certain orders. The idea behind masking is to prevent DPA by making the side-channel leakage independent from the processed data.

In a masked cryptographic implementation, every secret value v is split into $d+1$ shares $v_0, ..., v_d$ in order to protect against d-th order DPA attacks. Thereby, d shares are chosen uniformly at random and the $(d+1)$-th share is chosen such that the combination of all $d+1$ shares gives the actual secret value v. As a result, an adversary is required to combine the

side-channel leakage of all $d+1$ shares to be able to exploit the side channel, i.e., to perform a $(d+1)$-th order DPA.

While the masking operation itself is usually cheap, e.g., XOR, cryptographic primitives typically contain several operations that become more complex in the masked representation. This eventually results in massive implementation overheads. For example, the 1st-order DPA secure threshold implementations of AES in [7,33] add an area-time overhead of a factor of four.

3.4.2 *Frequent Re-Keying*

The success rate of key recovery with DPA rises with the number of different processed inputs. Therefore, frequent re-keying [24,29] tries to limit the number of different processed inputs per key, i.e., the data complexity.

The countermeasure constrains a cryptographic scheme to use a certain key k only for q different public inputs (q-limiting [41]). When the limit of q different inputs is reached, another key k' is chosen. Thus, for a certain key k, an adversary can only obtain side-channel leakage for q different inputs, which limits the feasibility of DPA to recover k.

Therefore, designing schemes and protocols with small data complexity q is one measure to prohibit DPA against unprotected cryptographic implementations. In more detail, it is widely accepted that very small data complexities, i.e., $q=1$ and $q=2$, have sufficiently small side-channel leakage and do not allow for successful key recovery from DPA attacks [5,36,41,44].

Leakage-Resilient Cryptography. Frequent re-keying can be applied to any cryptographic scheme, e.g., an encryption scheme ENC or an authenticated encryption scheme AE, by choosing a new key whenever a new message has to be encrypted and authenticated, respectively. However, in such a re-keying approach, side-channel resistance is also affected by the concrete instance of the cryptographic scheme. In practice, the cryptographic scheme must be able to process arbitrarily long messages using a standard primitive, e.g., AES with 128-bit block size. This situation facilitates DPA in certain modes, such as CBC. Therefore, the cryptographic scheme must be designed with special care.

A generic construction for an encryption scheme ENC that can process arbitrarily long messages without DPA vulnerability is given in Figure 1. For DPA security, it requires a new key k_0 to be chosen for every new message. To securely process an arbitrary number of message blocks, the depicted scheme chains a primitive f that encapsulates the block encryption $c_i = E(k_i; p_i)$ and a key update mechanism $k_{i+1} = u(k_i)$. Hereby, the included key update mechanism $k_i \rightarrow k_{i+1}$ ensures the unique use of each key k_i. The construction can be considered secure against side-channel attacks if the key update mechanism is chosen such that the side-channel leakages of all invocations to f cannot be usefully combined. However, note that given that the key is iteratively derived using f, random access to individual blocks is typically quite expensive.

Exemplary constructions following the principle of Figure 1 to design DPA secure schemes from unprotected primitives are the leakage-resilient encryption schemes in [36, 41, 44] and the leakage-resilient MAC in [35]. Block-cipher based instantiations of these schemes have a data complexity of $q=2$ in order to prohibit successful key recovery via DPA attacks.

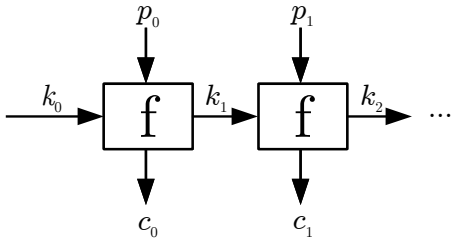

Figure 1: Generic encryption scheme ENC.

4. RE-KEYING FOR MEMORY ENCRYPTION

Frequent re-keying is a mechanism to protect against DPA without requiring that the implementation of the cryptographic primitive uses costly DPA countermeasures such as masking. Simultaneously, there are more and more practical systems being deployed with unprotected cryptographic accelerators by vendors not being aware of side-channel attacks. As a result, re-keying based schemes are an interesting option for protecting memory encryption and authentication against DPA.

In this section, we perform the first investigation of the security of re-keying in the context of memory encryption and authentication. It shows that contrary to other use cases, the re-keying operation itself can be realized without DPA countermeasures when protecting memory. However, we also show that the application of re-keying to memory encryption allows for profiled, higher-order DPA that leaks confidential constants in memory due to read-modify-write operations inevitably occuring in encrypted memory.

4.1 The Re-Keying Operation

Up until now, the principle of re-keying was applied only to communicating parties aiming for confidential transmission. Hereby, constructions following Figure 1 prevent DPA, but require the initialization with a fresh key and thus secure key synchronization between the communicating parties. This synchronization is typically achieved by deriving a fresh key from a shared master secret k and a public, random nonce n [14,29,41]. However, this approach shifts the DPA problem to the key derivation, which thus needs DPA protection through mechanisms like masking.

The encryption and authentication of data stored in memory gives different conditions for the instantiation of re-keying based schemes. In particular, encrypting data in memory means that en- and decryption is performed by the same party, i.e., a single device encrypts data, writes it to the memory, and later reads and decrypts the data. Therefore, key synchronization becomes unnecessary and the cryptographic scheme can be re-keyed using random numbers without the need for any cryptographic primitive or function being implemented with DPA countermeasures.

4.2 Re-Keying and Plaintext Confidentiality

The typical target of DPA attacks is the key being used as key recovery fully breaks a cryptographic scheme. Re-keying based schemes thus thwart such attacks and make DPA on the key infeasible. However, the actual goal of encryption is to ensure data confidentiality. Therefore, protecting the key against DPA is a useful measure, but as our analysis shows,

the application of re-keying to memory encryption can yet result in a loss of memory confidentiality.

The main observation that leads to this conclusion are read-modify-write operations that inevitably occur in any encrypted memory. These take place whenever the write granularity is smaller than the encryption granularity. For example, when a single byte is written to a memory that is encrypted using an 128-bit block cipher, the respective 128-bit encryption block has to be loaded from memory, decrypted and modified according to the byte-wise write access, and then be encrypted again and written back to the memory. In this case, 120 bits of the respective block remain the same. The same phenomenon is observed in encryption schemes that cover multiple encryption blocks $p_0, p_1, p_2, \ldots$. Here as well, one plaintext block, e.g., p_0, might be changing, while others, e.g., p_1, remain constant.

If now re-keying is applied to memory encryption, the constant plaintext parts within read-modify-write operations will be encrypted several times using different keys. This causes constant, secret plaintext parts to be mixed with varying keys. This situation is quite similar to the original DPA setting, where a constant, secret key is mixed with varying plaintexts. For stream ciphers, attackers can easily exploit this mixing operation—the XOR of varying pad and constant plaintexts—in a first-order DPA. Namely, attackers can model the power consumption of the varying pad for each plaintext hypothesis using the observed ciphertexts. Matching the power model with the side-channel observations then eventually reveals the constant plaintext. For block ciphers, a first-order DPA does not work, but a profiled, second-order DPA that is similar to unknown plaintext template attacks [19] can be applied to learn constant plaintexts.

Unknown Plaintext Template Attacks. In [19], the constant key k of a block cipher E is attacked by observing the encryption of several unknown plaintexts with the help of power templates. Hereby, the power templates are used to learn information on the unknown plaintexts $p_0, p_1, \ldots$ and intermediate values $v_0, v_1, \ldots$ in the respective encryption processes $E(k; p_0), E(k; p_1), \ldots$. Exploiting the relation between the information learned on $p_0, p_1, \ldots$ and $v_0, v_1, \ldots$, the key k is recovered. As the attack combines side-channel information from both the unknown plaintexts $p_0, p_1, \ldots$ and the intermediate values $v_0, v_1, \ldots$, the order of this attack is two.

The described attack can be easily applied to a re-keyed encryption scheme (cf. Figure 1). Namely, read-modify-write operations cause a constant plaintext block p_i to be encrypted several times using different keys $k_i, k_i', \ldots$. Changing the roles of plaintext and key in the attack from [19], re-keying allows to learn the constant plaintext block p_i from side-channel information on the varying key $k_i, k_i', \ldots$ and some intermediate value $v_i, v_i', \ldots$, both extracted using power templates. As a result, one plaintext may only be encrypted with one single key for re-keying to completely thwart DPA. This also seems reasonable in the view of leaking more information on a plaintext, the more often it is encrypted under different conditions, i.e., using different keys.

Summarizing, memory encryption inevitably causes read-modify-write operations. These cause re-keyed stream ciphers to become vulnerable to first-order DPA and re-keyed block ciphers to become vulnerable to profiled, second-order DPA. These attacks do not target the actual keys, but the confidential memory content. While these attacks cannot be prevented in the memory scenario, note that the effort and complexity of profiled, second-order DPA attacks is very high in practice. Hence, re-keyed block encryption provides a suitable basis to construct a memory encryption scheme with first-order DPA security. We further pursue this approach in Section 5. To obtain higher-order security, we extend our design in Section 6 and propose masking of the stored plaintext values. This effectively increases the number of values to be recoverd via templates without the need for masking being implemented in the cipher.

5. DPA-SECURE MEMORY ENCRYPTION AND AUTHENTICATION

The analysis in Section 4 showed that frequent re-keying of a block cipher based mode is a suitable approach to construct a memory encryption and authentication scheme with first-order DPA security from unprotected cryptographic primitives. However, one major requirement in Section 2 is to provide fast random access in memory, which is not provided by present re-keying based encryption schemes.

A common way to provide fast random access to large memory is to split the memory into blocks that can be directly accessed. However, encrypting each of these memory blocks by the means of fresh re-keying would render the number of keys to be kept available in secure on-chip storage too high. This problem is quite similar to memory authentication with replay protection, which also requires block-wise authenticity information to be stored in a trusted manner. To tackle this issue, state-of-the-art authenticity techniques (cf. Section 2 and Appendix A) employ tree constructions to gain scalability and to minimize the required amount of expensive on-chip storage.

In this section, we therefore use the synergies between frequent re-keying and memory authentication to present MEAS—a Memory Encryption and Authentication Scheme with first-order DPA security built upon unprotected cryptographic primitives and suitable for all kinds of large memory, e.g., RAM and NVM. Similar to existing memory authentication techniques, MEAS uses a tree structure to minimize the amount of secure on-chip storage. However, instead of hashes or nonces, keys are encapsulated within the tree. In more detail, the leaf nodes of the tree, which store the actual data, are encrypted and authenticated using an authenticated encryption scheme that is provided with fresh keys on every write access. Similarly, the inner nodes of the tree, which store the encryption keys for their respective child nodes, are encrypted with an encryption scheme that uses a fresh key on every write. MEAS is shown secure in the *leaking chip model*, and in particular, its DPA security is substantiated by limiting the number of different processed inputs per key to $q = 2$ such as in [5, 36, 41, 44].

In the following, we first present the construction of MEAS, followed by a security analysis considering authenticity, and side-channel attacks.

5.1 Construction

The construction of MEAS is designed to be secure according to the *leaking chip model*. Therefore, MEAS requires an SPA-secure block encryption scheme ENC and an SPA-secure authenticated encryption scheme AE. Both ENC and AE have to fulfill the common security properties for (authenticated) encryption schemes. Such schemes

can, e.g., be instantiated from unprotected cryptographic implementations using leakage-resilient block encryption [44], a leakage-resilient MAC [35], and the generic composition encrypt-then-MAC [6]. Besides, a random number generator is required for generating the keys.

An example of the tree construction proposed for MEAS is depicted in Figure 2. For the sake of simplicity, this example as well as the following description assumes the use of a binary tree, i.e., arity $a = 2$. However, instantiating the tree with higher arity is easily possible.

The structure of MEAS is as follows. The data in memory is split into m plaintext blocks p_i. Each of these p_i is encrypted and authenticated to a ciphertext-tag pair (c_i, t_i) using the authenticated encryption scheme AE with data encryption key dek_i:

$$(c_i, t_i) = AE(dek_i; p_i) \qquad 0 \leq i \leq m-1.$$

The encryption scheme ENC then encrypts the data encryption keys dek_i to the ciphertexts $c_{l-1,i}$ using key encryption keys $kek_{l-1,i}$. The operator $||$ denotes concatenation.

$$c_{l-1,i} = ENC(kek_{l-1,i}; dek_{2i}||dek_{2i+1}) \quad 0 \leq i \leq \frac{m}{2} - 1.$$

Recursively applying ENC in a similar way to the key encryption keys finally leads to the desired tree.

$$c_{j,i} = ENC(kek_{j,i}; kek_{j+1,2i}||kek_{j+1,2i+1}) \quad 0 \leq i \leq \frac{m}{2^{l-j}} - 1 \quad 0 \leq j \leq l-2$$

While all ciphertexts and tags are stored in public, untrusted memory, the root key $kek_{0,0}$ is stored on the leaking chip.

5.1.1 Read Operation

When reading data (c_i, t_i) from memory, all the keys on the path from the root key $kek_{0,0}$ down to the respective data encryption key dek_i are decrypted one after another. The data encryption key dek_i is then used to decrypt and authenticate the respective memory block (c_i, t_i).

For example in Figure 2, to obtain the plaintext block p_2 stored in (c_2, t_2), the root key $kek_{0,0}$ is used to decrypt $kek_{1,0}$. Then, $kek_{1,0}$ is used to decrypt $kek_{2,1}$, which permits to decrypt dek_2. Finally, dek_2 is used with (c_2, t_2) to authenticate and decrypt the respective plaintext p_2.

Note that the decryption of the encapsulated keys can only be performed sequentially. However, this is not considered a problem since computation is typically much faster than storage (e.g., RAM or HDD). On the other hand, caching of the intermediate nodes (key encryption keys) is supported by MEAS in order to achieve good performance, e.g., small average access latency.

5.1.2 Write Operation

Writing data to the memory is where the actual re-keying is performed. Namely, the process of updating p_i with p'_i involves the replacement of all keys along the path from the root key $kek_{0,0}$ down to the respective data encryption key dek_i with randomly generated ones. On the other hand, the keys for the adjacent subtrees are only reencrypted under the new node keys. This re-keying can be performed in a single pass from the root to the leaf node of the tree.

For example in Figure 2, when block p_5, which is stored in (c_5, t_5), gets replaced, also the keys $kek_{0,0}$, $kek_{1,1}$, $kek_{2,2}$ and dek_5 have to be changed. Therefore, the node $c_{0,0}$ is decrypted to extract $kek_{1,0}$ and $kek_{1,1}$. The new node $c'_{0,0}$ can then be determined by encrypting $kek_{1,0}$ and a new $kek'_{1,1}$ with the new key encryption key $kek'_{0,0}$. The nodes $c_{1,1}$ and $c_{2,2}$ are updated in the same way. The new data block (c'_5, t'_5) is then the result of authenticated encryption of p'_5 under the new data encryption key dek'_5.

Note that it is not necessary to check authenticity when a full block is written to the memory. Only read-modify-write operations on a data block require an authenticity check. This authenticity check is automatically performed when the data is read prior to modification and thus does not inhere any additional costs. Also note that read-modify-write operations require only one single tree traversal, because the data encryption key required for the read operation automatically becomes available in the last steps of the write (and re-keying) procedure.

5.2 Authenticity

The design of MEAS protects data authenticity with respect to spoofing, splicing, and replay attacks using both the authentic root key and the AE scheme. In particular, spoofing and splicing attacks on the leaf nodes are directly detected by the AE scheme since different keys are used for each block. Moreover, the AE scheme indirectly also protects the inner tree nodes for properly chosen schemes AE and ENC. In such case, any tampering with the ciphertext of an intermediate node will lead to a random but wrong key to be decrypted. This tampering will thus propagate down to the leaf node to give an erroneous, random data encryption key and finally an authentication error.

Replay protection for all nodes is the result of the authentic root key, which gets updated on every write to any leaf node, i.e., choosing a new, random root key on every write access ensures that the secure root reflects the current state of the tree in public memory. Vice versa, the authenticitiy tags in the leaf nodes output by the AE scheme reflect the authenticity of the path from the root to the respective data block. Therefore, if the authenticity check of a leaf node fails, any node on the path from the root to the leaf may be corrupted.

5.2.1 Handling corruption

As soon as a corrupted leaf node has been detected, it is required that the authenticity of the tree is restored before any further actions are taken. Otherwise, an adversary may be able to perform DPA attacks on encryption keys by introducing authenticity failures on purpose.

Restoring authenticity of the tree is simple and requires no additional support. It is sufficient to replace all corrupted data (leaf) nodes with random values since regular writes restore authenticity from the root to the respective leaf node. Restoring authenticity in this manner also causes re-keying on all nodes on the path from the root to the leaf to take place. This re-keying procedure effectively thwarts any DPA that otherwise could be performed by malicious modification of stored ciphertexts.

For example in Figure 2, if the authenticity check of the node (c_4, t_4) fails, any of the nodes $c_{0,0}, c_{1,1}, c_{2,2}$ and (c_4, t_4) can be erroneous. Therefore, the plaintext p_4 is replaced with a random plaintext p'_4 in order to restore the authenticity. Hereby, new keys $kek'_{0,0}, kek'_{1,1}, kek'_{2,2}$ and dek'_4 are chosen and the stored values $c'_{0,0}, c'_{1,1}, c'_{2,2}$ and (c'_4, t'_4) are updated accordingly. This procedure restores the authenticity of the

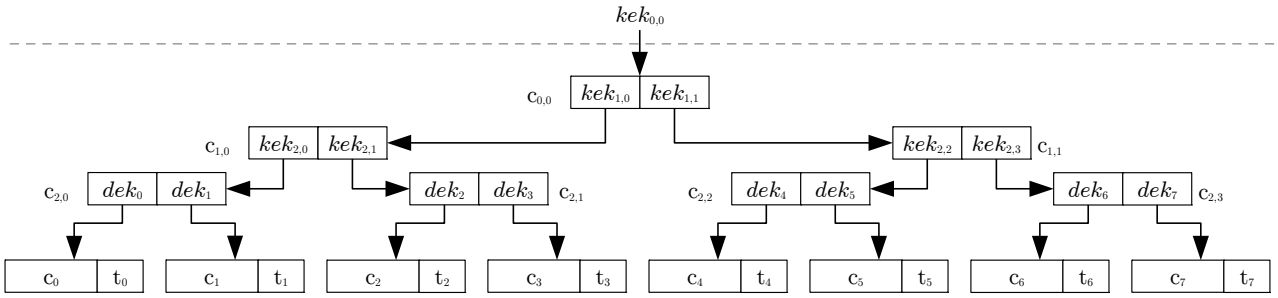

Figure 2: Meas's tree construction for $m = 8$ data blocks and with an arity of $a = 2$.

path from $kek_{0,0}$ to dek_4, but leaves any adjacent subtree intact. Moreover, the choice of fresh keys $kek'_{0,0}, kek'_{1,1}, kek'_{2,2}$ and dek'_4 prevents first-order DPA through adversaries repeatedly modifying $c'_{0,0}, c'_{1,1}, c'_{2,2}$ or (c'_4, t'_4).

5.2.2 *Recovering from corruption*

Depending on the actual application, there are different approaches to deal with the corruption. A straightforward approach, which is suitable for RAM encryption, is to simply reset the tree and start from scratch. The memory encryption engine of SGX [17], for example, follows this approach and requires a system restart to recover. However, applying this idea to block-level disk encryption is impractical since a reset of the tree is equivalent to destroying the data of the whole block device.

Another, more graceful approach is to recover from the corruption when possible. In the case of RAM encryption, it is, for example, possible that the operating system kills (and restarts) only those processes which actually accessed a corrupt data block. In the setting of disk encryption, it can be enough to report which files or directories were destroyed to enable appropriate error handling.

Given a single authentication failure, it is not possible to determine which node is corrupt. However, since corruptions in higher tree nodes lead to authenticity failures in more data blocks, it is possible to identify the subtree which is affected by the data corruption using multiple adjacent reads. This can even be done quite efficiently in a binary search like approach (i.e., $\mathcal{O}(\log m)$ reads), assuming that only a single node has been corrupted.

For example in Figure 2, when the authenticity check of data block 2, i.e., (c_2, t_2), fails, then data block 3 is checked next. If block 3 is authentic, then only block 2 (child of dek_2) is corrupt. Otherwise, either block 0 or block 1 is checked next. If this next block is authentic, then only blocks 2 and 3 (children of $kek_{2,1}$) have been corrupted. In case of another error, a final check in the right subtree (children of $kek_{1,1}$) is needed to determine if only the left subtree (children of $kek_{1,0}$) or the whole tree is corrupt. Note however that locating the corruption requires each authenticity failure to be followed by a re-keying step as described in Section 5.2.1 in order to resist DPA. For example, if data block 2 is read and detected to be corrupt, the path from the root key to data block 2 must be re-keyed. If during the location phase data block 3 is detected to be unauthentic as well, also the path from the root key to data block 3 must be re-keyed. The same procedure applies to all other checks in the location phase.

5.3 Side-Channel Discussion

We discuss the side-channel security using three types of attackers with increasing capabilities. The first type solely uses passive attacks and tries to exploit the side-channel leakage during operation. The second type additionally induces authenticity errors by tampering with the memory and strives for exploiting error handling behavior. The third type further tries to gain an advantage by restarting, i.e., power cycling, the whole system at arbitrary points in time.

Passive Attacks.

The protection of Meas against DPA lies within the re-keying approach. Therefore, every randomly generated key is used for the encryption and decryption of exactly one tree node with one specific plaintext. As soon as the plaintext of a node changes in any way, also a new key for the encryption of the respective node is generated.

For a certain key, a physical attacker who only passively observes Meas can thus at most acquire side-channel traces of one encryption and arbitrarily many decryptions of one single plaintext. Even though the trace number is possibly high, the best an attacker can do is to combine all the traces to a single rather noise free trace of this one key-plaintext pair. To perform a DPA, on the other hand, traces for multiple different plaintexts are required. In the presence of a passive physical attacker, Meas is therefore secure against first-order DPA attacks given that both ENC and AE are SPA secure.

Passive Attacks and Memory Tampering.

An active physical attacker who tampers with the memory content can gain additional information by corrupting the ciphertext of certain nodes. Namely, such tampering gives side-channel information from the decryption of different data for one single key. However, even with such tampering it is only possible to acquire one additional side-channel trace for a specific key. This is due to the fact that every tampering is detected as soon as the leaf node is authenticated. Handling the authentication error involves restoring authenticity and thus re-keying which makes the gathering of further traces impossible. As a result, the number of acquirable traces (i.e., under the same key, but with different ciphertexts) is clearly bounded by two. Given the assumptions in related work on leakage-resilient cryptography [36,41,44], bounding the input data complexity per key by two makes Meas secure against first-order DPA for malicious memory corruption.

Passive Attacks, Memory Tampering and Restarts.

The side-channel security of MEAS relies on the assumption that tree operations are performed atomically. This means that, e.g., once a read operation is started, all steps involved in MEAS, i.a., the MAC verification and the re-keying on authenticity failure, must be performed and completed. This assumption holds true for a running device since physical fault attacks on the leaking chip are outside the threat model. However, restarting the device during operation can break this assumption. In this case, attackers can use a combination of power cycling and memory tampering to collect arbitrarily many side-channel traces and perform a first-order DPA against a non-volatile key. However, this attack is easily prevented when the concrete use case is known.

For the encryption and authentication of RAM, there is simply no reason to maintain persistent keys between system restarts. Similar to SGX, the device generates a new random key on startup which effectively thwarts the attack. For NVM, however, a persistent root key is unavoidable. Yet, there are easy and secure solutions for NVM too. For example, one could store one additional bit on the leaking chip to record whether a presumably atomic operation is currently active. This allows to detect aborted operations in MEAS on startup and thus to take further actions, e.g., counting and storing the number of aborted operations on the leaking chip and appropriate error handling when a certain threshold is reached. Such countermeasures can also be integrated with the transaction/journaling functionality of a file system.

Summarizing, MEAS itself does not contain any mechanism to deal with malicious power cycling. However, for both RAM and NVM simple and cheap solutions are available.

6. HIGHER-ORDER DPA SECURITY

The tree construction presented in the previous section provides memory confidentiality and authenticity in the presence of a first-order side-channel adversary. However, profiled, second-order attacks as outlined in Section 4 still reveal the content of the tree nodes protected by the means of re-keying. Since the loss of confidentiality of a node close to the root would also reveal large chunks of the protected memory, i.e., all child nodes, protection against higher-order DPA is desireable.

In this section, we propose masking of the plaintext values to extend the protection of MEAS to higher-order DPA. The extension works with cryptographic primitives implemented without DPA countermeasures and allows to dynamically adjust the protection order depending on the actual threat.

6.1 Concept

The basic idea to provide higher order DPA security is to add a masking scheme (cf. Section 3.4.1) to MEAS. However, unlike the masking of specific cryptographic implementations, the proposed data masking scheme operates with unprotected primitives. Therefore, the plaintext data in each tree node of MEAS is first masked, and then the masked plaintext and the masks are encrypted separately and both stored in memory. On decryption, both the masked plaintexts and the masks are decrypted and the masks applied to obtain the original plaintext value.

The masking scheme requires new masks to be chosen whenever the key of a tree node is changed. This is the case on every write access to a specific node. As a result, the data being encrypted is randomized. This prevents that constant data is encrypted under different keys. Moreover, it requires adversaries trying to learn a constant plaintext using profiled attacks such as described in Section 4 to additionally extract information on every single mask from the side-channel. Therefore, the order of the attack increases accordingly.

6.2 Masking Details

The following masking approach can be applied accordingly to both the intermediate nodes, which use an encryption scheme ENC, and the leaf nodes, which use an authenticated encryption scheme AE. However, for simplicity we only consider the encryption of an arbitrary tree node using an encryption scheme ENC.

When encrypting a tree node in MEAS, the node's plaintext p is split into $b+1$ blocks $p_0, ..., p_b$ according to the size of the underlying encryption primitive, i.e., 128 bits in case of AES. In order to protect this node against d-th order DPA, $d-1$ random and secret masks $m_0, ..., m_{d-2}$ have to be generated. These masks are then applied to each plaintext block p_i to give random values r_i:

$$r_i = p_i \oplus m_0 \oplus ... \oplus m_{d-2} \qquad 0 \leq i \leq b.$$

In the actual encryption, both the masks $m_0, ..., m_{d-2}$ and the random values $r_0, ..., r_b$ are processed and the respective ciphertext c is stored in memory:

$$c = ENC(dek; m_0||...||m_{d-2}||r_0||...||r_b).$$

Whenever the node has to be read, the ciphertext is decrypted to give $m_0||...||m_{d-2}||r_0||...||r_b$. To obtain the plaintext blocks p_i, the masking is reverted by again xor-ing all masks $m_0, ..., m_{d-2}$ to each block r_i.

6.3 Side-Channel Discussion

The re-keying of the (authenticated) encryption scheme guarantees that adversaries are not capable of building suitable DPA power models from the observation of ciphertexts and thus prevents DPA against the key completely.

To prevent the loss of plaintext confidentiality from the profiled, second-order attacks outlined in Section 4, the proposed masking scheme randomizes the plaintext input using $d-1$ random, secret masks. As a result, the scheme requires adversaries to combine side-channel information from $(d+1)$ different values to recover the plaintext, i.e., to perform a $(d+1)$-th order DPA. In particular, such DPA requires to learn side-channel information on the varying key, an intermediate value in the cipher, and the $d-1$ masks. On the other hand, the masking scheme requires to additionally encrypt $d-1$ masks in each tree node. However, for a properly chosen encryption scheme ENC, these encryption operations cannot be exploited in a DPA, because both the masks and the keys are random and always changed simultaneously on every write access to the respective tree node.

Note, however, that in order for the masking to protect MEAS also in the presence of hardware glitches, the sum of plaintext and the masks must be stored in a register prior to the encryption operation. This is automatically the case if the masking is implemented in software. Hereby, the result is stored in a register and may then, e.g., be further processed in a cryptographic hardware accelerator.

Besides, we also emphasize that profiled DPA attacks such as in Section 4—which are counteracted by the proposed masking scheme—are quite hard to conduct on state-of-the-art systems. For example, while the unknown plaintext

template attack in [19] was performed against software implementations on 8-bit and 32-bit microcontrollers, a profiled DPA will take significantly more effort on hardware implementations embedded in a complex system-on-chip. Moreover, the attack complexity also rises rapidly with the attack order. As a result, small protection orders will already be sufficient for MEAS in practice. However, a detailed analysis of the side-channel leakage of a device implementing MEAS is indispensable for a proper choice of the protection order.

6.4 Implementation Aspects

The definite choice of the implemented protection order allows for various trade-offs influenced by several parameters: the cost for storing the masks, the concrete leakage behavior of the device, and the risk. Hereby, the leakage behavior and the cost for storing the masks are closely coupled.

A DPA is more likely to be successful on a device the more side-channel leakage the device gives. Therefore, a higher protection order is needed the more the device leaks, which leads to higher storage costs for masks. Alternatively, the leakage of the device might be reduced by hiding countermeasures [28] in the implementation, such as shuffling. However, such countermeasures can only be built into newly designed devices. Nevertheless, besides the actual strength of a potential attacker, the actual leakage behavior of the device forms the basis for the choice of the protection order and thus memory cost.

Besides, the choice of the protection order is also strongly influenced by the concrete risk of an attack. In more detail, a trade-off between the protection order and the risk is possible. Namely, the higher the risk of an attack to a specific block, the better should be the protection of the respective block, i.e., the higher should be the protection order. Concretely in MEAS, the tree nodes stored in levels closer to the root are a more interesting target for an attacker since revealing the keys stored in these nodes would allow to decrypt large parts of the memory. Therefore, tree nodes closer to the root are at higher risk and thus need a higher protection order. However, the number of nodes in one tree level decreases the closer the respective level is to the root. As a result, increasing the protection order for tree nodes at higher risk has only little memory overhead in MEAS and thus is an inexpensive improvement of security against higher-order DPA.

7. EVALUATION

MEAS is a novel approach to provide authentic and confidential memory with DPA protection. While there already exist several concepts for memory encryption and authentication (cf. Section 2), all of them lack the consideration of side-channel attacks.

In this section, we compare MEAS with these state-of-the-art techniques regarding security properties, parallelizability, randomness, and memory overhead. Our methodology to assess the overheads is independent of any concrete implementation and allows to precisely state the asymptotic memory requirements of all schemes for any real-world instance. It shows that MEAS can efficiently provide first-order DPA-secure memory encryption and authentication at roughly the same cost as existing authentication techniques, which, on the other hand, completely lack the consideration of DPA at all. Put into numbers, a 4-ary, first-order DPA secure MEAS instance for standard hard disks results in a very low memory overhead of 7.3%. Contrary to that, using DPA-protected implementations in contemporary memory encryption and authentication schemes would be impractical due to their massive implementation overheads.

7.1 Security Properties

Comparing the contestants in Table 1 regarding security properties shows that only MEAS and TEC trees provide both confidentiality and authenticity in the form of spoofing, splicing and replay protection. DPA security, on the other hand, is only featured by MEAS and Merkle trees. However, Merkle trees do not provide confidentiality and their DPA security can be considered a side effect. Namely, the hash functions used in Merkle trees simply do not use any secret material, i.e., keys or plaintexts, which is the common target in DPA attacks.

7.2 Parallelizability

A more performance oriented feature, on which previous tree constructions typically improved on, is the ability to compute the cryptographic operations involved in read and write operations in parallel. Having this property is nice in theory, but is in practice not the deciding factor to gain performance. To make use of a scheme's parallelism, multiple parallel implementations of the cryptographic primitives as well as multi-port memory, to read and write various nodes in parallel, are required. Since these resources are typically not available, a common, alternative approach to improve performance is the excessive use of caches.

In MEAS, due to the key encapsulation approach used to achieve its DPA security, parallelizing the computations within the encryption scheme is not possible. However, this is not necessarily a problem preventing the adoption of MEAS in practice since on-chip computation is very fast compared to off-chip memory accesses. Additionally, like for all authentication trees, caches for intermediate nodes are a very effective and important measure to reduce the average latency. In summary, the performance of any authentication tree (and MEAS) is mainly determined by the tree height, which depends on both the tree arity and the number of blocks in the authenticated memory, and the cache size. As a result, given a concrete implementation of the cryptographic primitive, the actual runtime performance of all authentication trees is expected to be quite similar.

7.3 Memory Overhead

Table 1 further contains the memory overhead formulas that have been derived for each scheme. These formulas take into account the tree arity a, and the sizes for data blocks s_b, nonces s_{nonce}, hashes s_{hash}, tags s_{tag}, and keys s_{key}. The overhead formulas neglect the influence of the actual number of data blocks m given that it vanishes with rising node counts. The overheads therefore have to be considered as an upper bound which gets tight with $m \to \infty$. This approach gives exact and comparable results that are independent of the actual implementation and that are realistic for any memory with more than 128 data blocks.

The different parameters involved may make the overhead comparison seem difficult at first glance. However, it gets quite simple when actual instantiations are considered. Instantiating the trees for a fixed security level with $s_{nonce} = s_{tag} = s_{key}$ and $s_{hash} = 2 \cdot s_{tag}$, for example, shows that Merkle trees, PATs, and TEC trees have identical overhead. The overhead of MEAS, on the other hand, is even

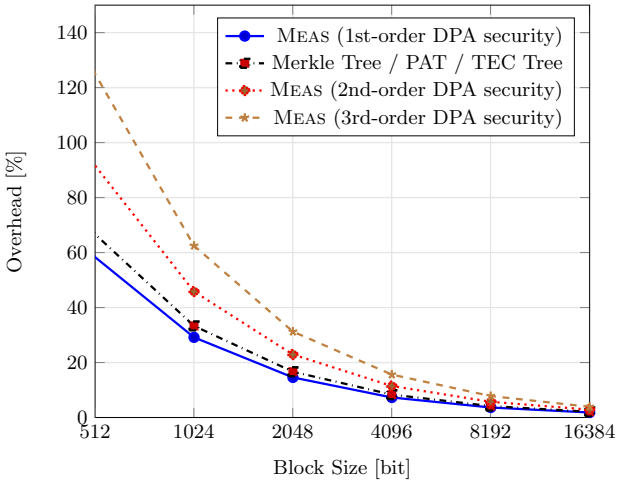

Figure 3: Memory overhead comparison for 4-ary trees depending on protection order and block size with a security level of 128 bits ($a = 4$, $s_{nonce} = s_{tag} = s_{key} = 128$, $s_{hash} = 256$).

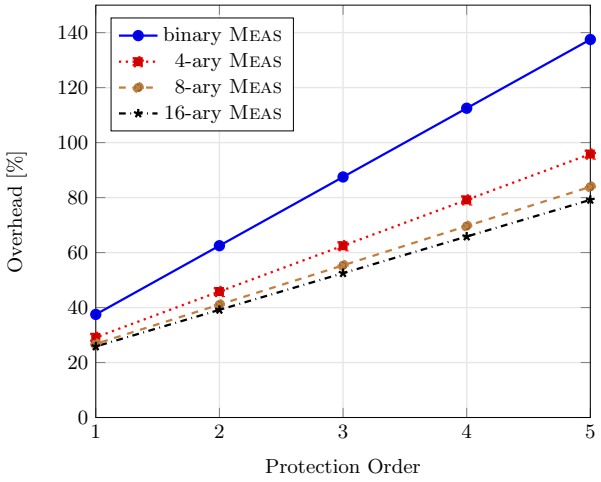

Figure 4: Memory overhead of Meas depending on arity and protection order (1024-bit blocks, 128-bit security).

lower, especially with small arity. This is due to the fact that in MEAS only leaf nodes are directly authenticated. On the other hand, PATs and TEC trees directly protect the authenticity of every tree node.

The memory overhead of MEAS, PATs, Merkle trees, and TEC trees is also visualized in Figure 3 for different block sizes. For practical instantiations, the block size will be chosen according to the system architecture, namely, page size, sector size, or cache line size. Both the sectors of modern disks as well as memory pages in state-of-the-art systems are sized 4096 bytes (=32768 bits). Such large block size is out of scope of Figure 3 as it has negligible memory overhead in any case. Besides, the memory overhead for a block size of 4096 bits (sector size in older hard disks) is also very low, e.g., 7.3% for 4-ary MEAS. However, the memory overhead of MEAS for block sizes fitting nowadays cache architectures is also practical given the security features it provides. While today's typical cache line size is 512 bits, modern CPUs often come with features such as Adjacent Cache Line Prefetch [22], which effectively double the cache line fetches from memory to 1024 bits. In a 4-ary MEAS, for example, such block size results in decent 29.2% memory overhead.

Note that these relatively small overheads—quite similar to existing authentication techniques—in combination with additional and exclusive DPA protection are the main advantage of MEAS. Using existing memory encryption and authentication schemes with DPA-protected implementations, on the other hand, would result in overheads of a factor of four to a few hundred [4, 7, 33, 35] and thus be far more expensive, eventually rendering memory encryption and authentication in many applications impractical.

7.4 Memory Overhead with Masking

The memory overhead of MEAS with higher-order DPA protection additionally depends on the protection order d and the size of the masks s_{mask} which is typically equal to s_{key}. In particular, a generalized version of the limit of the memory overhead as the number of memory blocks approaches infinity is:

$$\frac{a}{a-1} \cdot \frac{s_{key} + (d-1) \cdot s_{mask}}{s_b} + \frac{s_{tag}}{s_b}.$$

Figure 3 contains an evaluation of this memory overhead for a 4-ary tree and 128-bit security, i.e., the keys, the tags, and the masks are sized 128 bits. It shows that masking adds multiplicatively to the memory overhead for all block sizes. However, for larger block sizes, the memory overhead of MEAS becomes negligible regardless of the protection order.

Table 1: Comparison of Meas with other constructions for scalable authentic and/or confidential memory which offer block wise random access.

	Auth.	Conf.	DPA Security	Parallelizable[a]		Memory Overhead
				Read	Write	
MEAS	✓	✓	✓			$\frac{a}{a-1} \cdot \frac{s_{key}}{s_b} + \frac{s_{tag}}{s_b}$
PAT	✓			✓	✓	$\frac{a}{a-1} \cdot \frac{s_{tag}+s_{nonce}}{s_b}$
TEC Tree	✓	✓		✓	✓	$\frac{a}{a-1} \cdot \frac{s_{tag}+s_{nonce}}{s_b}$
Merkle Tree	✓		✓	✓		$\frac{a}{a-1} \cdot \frac{s_{hash}}{s_b}$

[a] Requires multiple cryptographic implementations and multi-port memory in practice.

Note that the protection order stated for MEAS in Figure 3 applies to all nodes in MEAS. If however, and as explained in Section 6.4, different protection orders are used for nodes at different risk, the depicted plots mark the border cases for the actual memory overhead. For example, if low-level tree nodes do not use masking (i.e., having first-order DPA security) and first-order masking is applied to all other nodes (i.e., having second-order DPA security), the actual memory overhead is lower- and upper-bounded by the plot with first- and second-order protection, respectively.

An evaluation of the memory overhead of MEAS over different protection orders and arity is depicted for 1024-bit blocks and 128-bit security in Figure 4. Hereby, it turns out that the memory overhead is strongly influenced by the tree's arity leading to two main observations. First, a higher arity clearly lowers the memory overhead, but for an arity higher than eight, the reduction resulting from another increase of the arity becomes quite small. Second, the memory overhead rises linearly with the protection order, but the increase is stronger the lower the tree's arity is. This is due to the masks for randomization of the plaintext being chosen and stored for each tree node. As a result, higher arity leads to more plaintext blocks sharing such masks in one tree node and thus lower memory overhead due to the masking.

7.5 Randomness

MEAS consumes a considerable amount of randomness. In particular, fresh random keys and masks must be chosen for all nodes on the path from the root to the leaf whenever a write operation is performed. For MEAS with protection order d, this sums up to $(s_{key} + (d-1) \cdot s_{mask}) \cdot (l+1)$ random bits needed on each write operation, where l is the tree height. Implementations of Merkle trees, PATs and TEC trees without consideration of side channels however do not require any random value if all nonces are chosen as counters. Yet, cipher implementations that protect PATs and TEC trees against side-channel attacks rely on significant amounts of randomness too. Namely, implementations with protection order d split its state into $(d+1)$ shares. This demands for at least $d \cdot s_{state}$ random bits per cipher invocation that get necessary for all accessed nodes on both reads and writes.

8. CONCLUSION

Authentic and encrypted memory is a requirement for storing and processing data in hostile environments where attackers have physical access. The consideration of the imminent threat of side-channel attacks against the involved cryptographic primitives is thus the natural next step.

In this work, we therefore presented MEAS, the first Memory Encryption and Authentication Scheme which is secure against DPA attacks. The scheme does not require any DPA-protected primitive, allowing its use in COTS systems. Moreover, MEAS provides fast random access on the configured block level and can be adopted for all kinds of use cases including RAM and disk encryption.

The scheme combines the concept of fresh re-keying with authentication trees by storing the involved keys in an encrypted tree structure. While this prevents first-order DPA, masking of the plaintext values flexibly extends the protection of MEAS to higher-order DPA if required. Compared to existing schemes, MEAS exclusively offers DPA protection by design at roughly the same memory overhead and performance. This is a clear benefit over state-of-the-art memory authentication and encryption techniques, which would face impractical implementation and runtime overheads for DPA-protected implementations if adapted accordingly.

Acknowledgments.

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402). Further, this work has been supported by the Austrian Research Promotion Agency (FFG) under the grant number 845579 (MEMSEC).

9. REFERENCES

[1] Apple Inc. Apple Technical White Paper: Best Practices for Deploying FileVault 2, 2012. http://docplayer.net/281501-Best-practices-for-deploying-filevault-2.html.

[2] Apple Inc. iOS Security, 2015. https://www.apple.com/business/docs/iOS_Security_Guide.pdf.

[3] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede. DPA, bitslicing and masking at 1 ghz. In *Cryptographic Hardware and Embedded Systems - CHES 2015*, pages 599–619, 2015.

[4] S. Belaïd, V. Grosso, and F. Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? Cryptology ePrint Archive, Report 2014/053, 2014.

[5] S. Belaïd, F. D. Santis, J. Heyszl, S. Mangard, M. Medwed, J. Schmidt, F. Standaert, and S. Tillich. Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis. *J. Cryptographic Engineering*, 4(3):157–171, 2014.

[6] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[7] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient AES threshold implementation. In *Progress in Cryptology – AFRICACRYPT 2014*, pages 267–284, 2014.

[8] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO 1999*, pages 398–412, 1999.

[9] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, 2002.

[10] R. Elbaz, D. Champagne, C. H. Gebotys, R. B. Lee, N. R. Potlapally, and L. Torres. Hardware mechanisms for memory authentication: A survey of existing techniques and engines. *Trans. Computational Science*, 4:1–22, 2009.

[11] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli, and P. Guillemin. Tec-tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 289–302, 2007.

[12] N. Ferguson. AES-CBC + Elephant diffuser A Disk Encryption Algorithm for Windows Vista, Aug. 2006.

[13] C. Fruhwirth. New Methods in Hard Disk Encryption, 2005.

[14] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[15] Google Inc. Android Full Disk Encryption, 2015. https://source.android.com/security/encryption/.

[16] L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems - CHES 1999*, pages 158–172, 1999.

[17] S. Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016:204, 2016.

[18] W. E. Hall and C. S. Jutla. Parallelizable authentication trees. In *Selected Areas in Cryptography - SAC 2005*, pages 95–109, 2005.

[19] N. Hanley, M. Tunstall, and W. P. Marnane. Unknown plaintext template attacks. In *Information Security Applications - WISA 2009*, pages 148–162, 2009.

[20] M. Henson and S. Taylor. Beyond full disk encryption: Protection on security-enhanced commodity processors. In *Applied Cryptography and Network Security - ACNS 2013*, pages 307–321, 2013.

[21] IEEE. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE Std 1619-2007, April 2008.

[22] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer Manuals. 325462-058.

[23] D. Kaplan, J. Powell, and T. Woller. AMD memory encryption, 2016. http://developer.amd.com/resources/articles-whitepapers/.

[24] P. Kocher. Leak-resistant cryptographic indexed key update, Mar. 25 2003. US Patent 6,539,092.

[25] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO 1999*, pages 388–397, 1999.

[26] 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.

[27] J. Longo, E. D. Mulder, D. Page, and M. Tunstall. Soc it to EM: electromagnetic side-channel attacks on a complex system-on-chip. In *Cryptographic Hardware and Embedded Systems - CHES 2015*, pages 620–640, 2015.

[28] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[29] M. Medwed, F. Standaert, J. Großschädl, and F. Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In *Progress in Cryptology - AFRICACRYPT 2010*, pages 279–296, 2010.

[30] R. C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy - SP 1980*, pages 122–134, 1980.

[31] T. S. Messerges. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems - CHES 2000*, pages 238–251, 2000.

[32] T. T. Michael Halcrow, Uday Savagaonkar and I. Muslukhov. Ext4 Encryption Design Document. http://goo.gl/qbcZV2.

[33] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology - EUROCRYPT 2011*, pages 69–88, 2011.

[34] E. Owusu, J. Guajardo, J. M. McCune, J. Newsome, A. Perrig, and A. Vasudevan. OASIS: on achieving a sanctuary for integrity and secrecy on untrusted platforms. In *Computer and Communications Security - CCS 2013*, pages 13–24, 2013.

[35] O. Pereira, F. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *Computer and Communications Security – CCS 2015*, pages 96–108, 2015.

[36] K. Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology – EUROCRYPT 2009*, pages 462–482, 2009.

[37] P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004*, pages 16–31. Springer Berlin Heidelberg, 2004.

[38] B. Rogers, S. Chhabra, M. Prvulovic, and D. Solihin. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *IEEE/ACM International Symposium on Microarchitecture - MICRO 2007*, pages 183–196, Dec 2007.

[39] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors OS- and performance-friendly. In *IEEE/ACM International Symposium on Microarchitecture - MICRO 2007*, pages 183–196, 2007.

[40] P. R. Sami Saab and C. Hampel. Side-channel protections for cryptographic instruction set extensions. Cryptology ePrint Archive, Report 2016/700, 2016.

[41] F. Standaert, O. Pereira, Y. Yu, J. Quisquater, M. Yung, and E. Oswald. Leakage resilient cryptography in practice. In *Towards Hardware-Intrinsic Security - Foundations and Practice*, pages 99–134. 2010.

[42] G. Suh, D. Clarke, B. Gasend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *IEEE/ACM International Symposium on Microarchitecture - MICRO 2003*, pages 339–350, Dec 2003.

[43] G. E. Suh, D. E. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *International Conference on Supercomputing - ICS 2003*, pages 160–171, 2003.

[44] M. M. I. Taha and P. Schaumont. Key updating for leakage resiliency with application to AES modes of operation. *IEEE Trans. Information Forensics and Security*, 10(3):519–528, 2015.

[45] T. Unterluggauer and S. Mangard. Exploiting the physical disparity: Side-channel attacks on memory encryption. In *Constructive Side-Channel Analysis and Secure Design, COSADE 2016*, pages 3–18, 2016.

APPENDIX

A. AUTHENTICATION TREES

In the following we describe three prominent examples of authentication trees, namely, Merkle trees [30], Parallelizable Authentication Trees [18] (PAT), and Tamper Evident Counter [11] (TEC) trees. Note however that there are also hybrid variants like Bonsai Merkle trees [39], which use elements from both Merkle trees and PATs. The description assumes binary trees, the operator $||$ denotes concatenation.

A.1 Merkle Trees [30]

Merkle trees use a hash function H to hash each of the m memory blocks p_i:

$$h_{l,i} = H(p_i) \qquad 0 \leq i \leq m-1.$$

These hashes $h_{l,i}$ are recursively hashed together in a tree structure and the root hash $h_{0,0}$ is put on the secure chip:

$$h_{j,i} = H(h_{j+1,2i}||h_{j+1,2i+1}) \qquad 0 \leq i \leq \frac{m}{2^{l-j}} - 1, \quad 0 \leq j \leq l-1.$$

A.2 Parallelizable Authentication Trees [18]

PATs use a nonce-based MAC and a key k to authenticate each of the m data blocks p_i using a tag $t_{l,i}$:

$$t_{l,i} = MAC(k; n_{l,i}; p_i) \qquad 0 \leq i \leq m-1.$$

The nonces $n_{l,i}$ are recursively authenticated in a tree structure using again nonce-based MACs. While the key k and the root nonce $n_{0,0}$ must be stored on the secure chip, all other nonces and the tags are stored publicly in off-chip memory:

$$t_{j,i} = MAC(k; n_{j,i}; n_{j+1,2i}||n_{j+1,2i+1}) \qquad 0 \leq i \leq \frac{m}{2^{l-j}} - 1, \quad 0 \leq j \leq l-1.$$

A.3 Tamper Evident Counter Trees [11]

While Merkle trees and PATs provide memory authenticity, TEC trees additionally provide memory confidentiality. Therefore, TEC trees use Added Rendundancy Explicit Authenticity [13] (AREA) codes. Hereby, each plain memory block p_i is padded with a nonce $n_{l,i}$ and then encrypted with key k using a common block cipher:

$$c_{l,i} = E(k; p_i||n_{l,i}) \qquad 0 \leq i \leq m-1.$$

For verification, a ciphertext $c_{l,i}$ is decrypted to $p'_i||n'_{l,i}$ and $n'_{l,i}$ compared with the original nonce $n_{l,i}$. Hereby, the authenticity is ensured by the diffusion of the block cipher as it makes it hard for the adversary to modify the encrypted nonce $n_{l,i}$. The nonce $n_{l,i}$ is formed from the memory block address and a counter $ctr_{l,i}$ [11]. The nonce counters are recursively authenticated using AREA codes in a tree structure. The key k and the root counter $ctr_{0,0}$ are stored on the secure chip:

$$c_{j,i} = E(k; ctr_{j+1,2i}||ctr_{j+1,2i+1}||n_{j,i}) \qquad 0 \leq i \leq \frac{m}{2^{l-j}} - 1, \quad 0 \leq j \leq l-1.$$