Side-Channel Plaintext-Recovery Attacks on Leakage-Resilient Encryption

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

***Abstract*—Differential power analysis (DPA) is a powerful tool to extract the key of a cryptographic implementation from observing its power consumption during the en-/decryption of many different inputs. Therefore, cryptographic schemes based on frequent re-keying such as leakage-resilient encryption aim to inherently prevent DPA on the secret key by limiting the amount of data being processed under one key. However, the original asset of encryption, namely the plaintext, is disregarded.**

This paper builds on this observation and shows that the re-keying countermeasure does not only protect the secret key, but also induces another DPA vulnerability that allows for plaintext recovery. Namely, the frequent re-keying in leakage-resilient streaming modes causes constant plaintexts to be attackable through first-order DPA. Similarly, constant plaintexts can be revealed from re-keyed block ciphers using templates in a second-order DPA. Such plaintext recovery is particularly critical whenever long-term key material is encrypted and thus leaked. Besides leakage-resilient encryption, the presented attacks are also relevant for a wide range of other applications in practice that implicitly use re-keying, such as multi-party communication and memory encryption with random initialization for the key. Practical evaluations on both an FPGA and a microcontroller support the feasibility of the attacks and thus suggest the use of cryptographic implementations protected by mechanisms like masking in scenarios that require data encryption with multiple keys.

***Index Terms*—differential power analysis, side-channel attack, leakage-resilient encryption, re-keying**

I. Introduction

Side-channel attacks are a serious threat to cryptographic implementations. Such attacks allow to learn a secret value, e.g., a key, processed inside a cryptographic device from observing physical properties like the power consumption or the electromagnetic emanation (EM). One particularly strong variant of side-channel attacks is differential power analysis (DPA). DPA attacks effectively accumulate information about the secret value from multiple observations done during the processing of different inputs. While DPA attacks on embedded microcontrollers are often capable of recovering the secret value from less than 100 observations using distinct inputs [8], DPA has recently been shown to be a serious threat to state-of-the-art desktop systems too [16]. Therefore, there is a strong need for countermeasures.

Today, there basically exist two approaches to counteract side-channel attacks. The first approach is to secure the cryptographic implementation using mechanisms like masking [15]. Hereby, the data is randomized to make the side-channel information independent from the actual processed value. The second approach is to design cryptographic protocols such that they do not allow for certain classes of side-channel attacks at all. Examples for this approach are frequent re-keying [9], [10] and leakage-resilient cryptography [13], [14], [18], [19], which aim to inherently prevent DPA on the secret key.

The idea of fresh re-keying and leakage-resilient encryption is to limit the number of distinct inputs processed under one key. Therefore, a fresh key is chosen for every new encryption. This results in every key being used only once, rendering DPA on the secret key infeasible. Still, these schemes are required to resist side-channel attacks that succeed with a single observation, i.e., simple power analysis (SPA). Therefore, many of the leakage-resilient schemes are additionally proven to leak a bounded amount of information on the secret key material through side channels [13], [14], [18].

The approach followed by re-keying and leakage-resilient encryption has the advantage of successfully preventing side-channel attacks on the secret key without the need for dedicated DPA countermeasures in the cryptographic implementations. Yet, these schemes do not make any statements with respect to other confidential data. In particular, the actual goal of an encryption scheme is to ensure plaintext confidentiality. The fact that leakage-resilient encryption provides bounded leakage on key material does not imply bounded leakage of the plaintexts. Yet, the effect of re-keying and leakage-resilient encryption on the plaintext's side-channel security has not been investigated before.

A. Contribution

In this paper, we show that frequent re-keying as it occurs within leakage-resilient encryption is vulnerable to plaintext recovery using side-channel attacks. More concretely, we show that encrypting a constant plaintext multiple times with different keys facilitates DPA to recover the constant plaintext. Leakage-resilient stream ciphers such as in [13], [14], [18] thus leak a constant plaintext through a plain, first-order DPA and, moreover, a second-order, template-based DPA can be utilized to learn a constant plaintext that is the input of a block cipher that is protected by re-keying as in [9], [10], [19]. We verified both presented attacks on an FPGA and a microcontroller to emphasize their practicality.

However, we stress that our attacks are not limited to side-channel countermeasures such as leakage-resilient encryption, but are also relevant for any other scenario where the same data

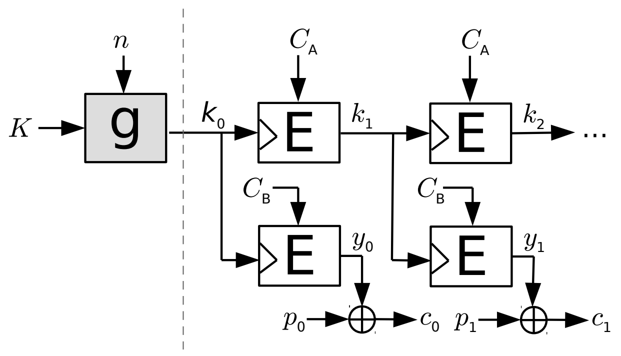

Fig. 1. Leakage-resilient stream cipher.

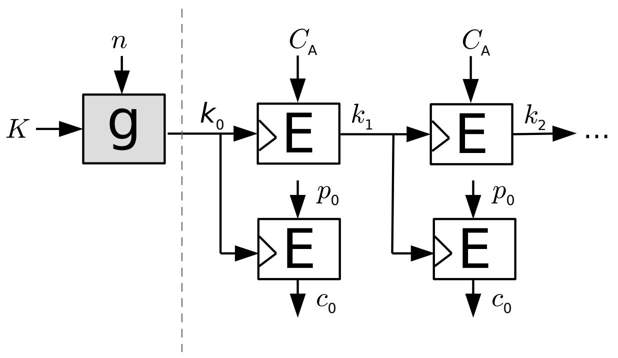

Fig. 2. Leakage-resilient block cipher encryption.

is encrypted using different keys. In particular, we show that the presented attacks are applicable to several scenarios in practice. For example, the encryption of random access memory (RAM) such as in Intel SGX [4] is often initialized using a random key, resulting in the same data being encrypted using a different key on every startup. Another prominent example are messages that are sent to multiple users that each use a different key.

We emphasize that our plaintext recovery attacks are a both realistic and serious threat. In particular, whenever long-term keys are encrypted, for example, when they are loaded into an encrypted memory, plaintext recovery implies the leakage of sensitive key material. As a consequence, we suggest the use of cryptographic implementations with dedicated DPA countermeasures in all these settings that encrypt the same data multiple times using different keys.

This paper is organized as follows. Section II gives background information on side-channel attacks and countermeasures. Section III presents side-channel plaintext-recovery attacks and its applications. Section IV elaborates on the attacks' practical verification and Section V concludes this work.

II. Background

In this section, we provide background information on side-channel attacks, the re-keying countermeasure, and leakage-resilient encryption.

A. Side-Channel Attacks

The execution of a cryptographic implementation leaks information on the processed data via various side-channels, such as power and EM. This information leakage is exploited by attackers in so-called side-channel attacks in order to learn secret information such as the key. Independent of the concrete source of side-channel leakage, there are two basic types of side-channel attacks: simple power analysis (SPA) and differential power analysis (DPA). While SPA tries to recover the secret key of a cryptographic implementation from observing the power consumption (or any equivalent side channel) during en-/decryption of one single input, DPA uses observations during the en-/decryption of many different inputs. Hereby, DPA is particularly effective as there is more side-channel leakage available, the more data is processed under one single key. One important property of DPA attacks is their *order*. The order d of a DPA [6], [11] is defined as the number of d different internal values in the executed algorithm that are used in the attack.

Both SPA and DPA attacks can further make use of *profiling*. Profiling a side-channel, e.g., the power consumption, means to classify the side-channel leakage of a specific device with respect to a certain value processed inside the device. The resulting *templates* [3] can then be used to attack an implementation by matching the templates with the observed side-channel trace in order to learn about the value processed inside the device.

B. Re-Keying and Leakage-Resilient Encryption

The probability for key recovery via DPA to be successful rises with the number of side-channel observations for different inputs. Therefore, one approach to counteract DPA is frequent re-keying [7], [10]. Hereby, the goal is to design a cryptographic scheme such that for a certain key k, the number of different inputs to the underlying cryptographic primitive is upper-bounded by some small number q (q-limiting [18]). As soon as the limit of q different inputs is reached, another key k' is selected. This limits the data complexity per key k. Thus, for a certain key k, the cryptographic implementation can only generate the side-channel leakage for q different inputs, which effectively limits the feasibility of DPA to recover k. As a result, the implementation of the cryptographic primitive is only required to resist SPA attacks.

One prominent example for this re-keying approach is leakage-resilient encryption, such as depicted in Fig. 1-2. Hereby, the re-keying operation itself is implemented using a leak-free initialization step to derive a session key k_0 from a pre-shared master secret K and a nonce n, i.e., via a both DPA- and SPA-secure re-keying function $g : (K, n) \mapsto k_0$. Therefore, a fresh nonce n has to be chosen for every encryption. The session key k_0 is then used in a leakage-resilient mode of operation that guarantees bounded data complexity per key when performing the actual en-/decryption. Therefore, leakage-resilient encryption modes utilize a key update step $k_i \mapsto k_{i+1}$ to provide a different key for the encryption of each plaintext block p_i. For example, the modes in Fig. 1-2 compute the next block's key k_{i+1} by encrypting a constant value C_A using an encryption primitive E, e.g., a block cipher, with the current block's key k_i.

As shown in Fig. 1-2, current leakage-resilient encryption modes employ two basic variants to perform the encryption of the single plaintext blocks p_i. The majority of modes uses a stream cipher approach [13], [14], [18] similar to Fig. 1. These modes use the current block's key k_i and a constant C_B to compute a value $y_i = E_{k_i}(C_B)$ that is used to pad the plaintext p_i as $c_i = p_i \oplus y_i$. On the other hand, there are also proposals for a block-cipher based approach [19] such as in Fig. 2, which directly use the current block's key k_i to encrypt p_i as $c_i = E_{k_i}(p_i)$. Both variants clearly limit each key's data complexity during encryption by two to make DPA on the keys k_i infeasible. Yet, both variants must be implemented such that they resist SPA-like attacks. Therefore, such schemes must have bounded leakage of the key material, i.e., it must be hard to usefully combine the leakages of the single k_i in order to learn the key stream. Streaming modes such as in Fig. 1 thus often come with a proof for bounded leakage of the key material given bounded leakage of the primitive E. Contrary to that, schemes such as in Fig. 2 currently lack this feature.

III. Side-Channel Plaintext-Recovery Attack

Fresh re-keying effectively prevents DPA on the key and often even allows to prove bounded leakage of the key material as in leakage-resilient encryption. However, even though encryption schemes have the goal to protect confidential data, leakage-resilient encryption currently lacks the consideration of plaintext confidentiality for side-channel adversaries.

In this section, we show that frequent re-keying is vulnerable to plaintext recovery using DPA. In particular, encrypting constant data using different keys allows to attack leakage-resilient stream ciphers using a first-order DPA and re-keyed block cipher encryption using a second-order template attack. While this setting is inherent with the re-keying countermeasure, we also show its practical relevance in several other applications.

A. Stream Cipher Mode

Frequent re-keying allows to perform a first-order DPA to recover a constant plaintext that is encrypted using a leakage-resilient stream cipher such as in Fig. 1. For illustration, we consider the encryption of a single, constant plaintext block p_i.

The choice of a fresh nonce n upon every encryption results in different key streams and thus in the plaintext p_i being encrypted using different pads y_i, y_i', y_i''. As a result, an attacker will, for the same plaintext p_i, observe different ciphertexts c_i, c_i', c_i'' and the respective power consumptions of the implementation. This observation facilitates DPA on the constant plaintext. Namely, the varying pad y_i allows to distinguish correct from wrong guesses of the plaintext p_i. In the DPA, the attacker can therefore model the power consumption of the stream cipher implementation as $HW(y_i) = HW(c_i \oplus p_i)$ for all observed ciphertexts c_i, c_i', c_i'' and for all guesses of p_i, where HW denotes the Hamming weight. Applying an appropriate statistical distinguisher, e.g., correlation, to the power model and the observed power consumption then reveals the correct plaintext p_i.

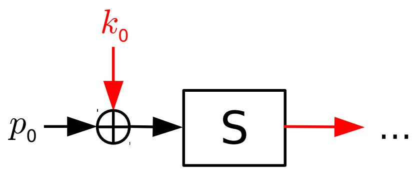

Fig. 3. UPTA-like plaintext recovery attack on one plaintext byte in the first AES round. Two templates, one on the whitening key k_0 and one on the S-box output $S(p_0 \oplus k_0)$, have to be trained.

Note that this DPA targets the linear XOR operation. We also stress that this kind of DPA is not limited to leakage-resilient stream ciphers as in Fig. 1, but applicable to stream ciphers in general. Namely, for cryptographic security, the pad of a stream cipher (and thus its initial value) must not repeat over different plaintexts. Therefore, producing different key streams, i.e., re-keying, is mandatory for any stream cipher implementation. The discussed attack scenario will thus appear every time when a plaintext is encrypted with a stream cipher more than once.

B. Block Cipher Mode

The leakage-resilient block cipher encryption mode in Fig. 2 is not subject to the previously discussed first-order DPA attack. This is due to the fact that a full block cipher, i.e., the bottom ciphers in Fig. 2, separates the known ciphertext from the constant plaintext, and therefore, without knowledge of the random key, we cannot verify any plaintext hypothesis. However, a second-order template attack targeting the first round of the data-encryption block cipher can be mounted on the construction instead.

The idea of the attack against the block cipher used for data encryption is similar to the idea behind an unkown plaintext template attack (UPTA) [5]. Such an UPTA recovers the secret key of a block cipher without having access to both the plain- and the ciphertext. Hereby, two templates are used to recover the required information instead. Combining the leakage information from multiple block cipher invocations in a DPA-like manner then permits to recover the secret key. Re-keying effectively prevents key recovery via such UPTA attacks since the key for the block cipher is constantly changed. However, an UPTA-like plaintext-recovery attack can still be mounted on the block cipher given that the plaintext is constant. In this case, the role of the key and the plaintext are simply swapped.

In more detail, an UPTA-like plaintext recovery attack on the initial round of a block cipher consists of two phases. In *phase one*, two templates are trained using labeled training traces. The first template provides combined information about both the key and the plaintext. It is therefore typically trained on the intermediate value after the first nonlinear function. The second template provides information about the secret key. None of the two templates is expected to determine the respective values unambiguously. However, the combination of the information provided by both templates should be descriptive.

Attacking, for example, the first round of the AES block cipher with such an UPTA-like plaintext recovery attack results in the setup shown in Fig. 3. The first template has to be trained on the S-box output $S(p_0 \oplus k_0)$ and provides information about the plaintext and the whitening key. The second template is trained on the whitening key k_0 itself and is supposed to learn information about the actual key value.

In *phase two*, the templates are matched with all target traces. For every trace, matching the first template provides attackers with a matrix of probabilities for all possible key and plaintext hypotheses. However, these matrices can not be combined across different target traces. Therefore, the second template is used to weight the key hypotheses within the afore mentioned matrices. As a result, the attacker is able to reduce the matrices with plaintext/key probabilities to vectors of plaintext probabilities. These vectors can then be easily combined across all target traces.

C. Implications and Applicability

The principle of the presented attacks on leakage-resilient encryption seems quite natural when taking into consideration that more information about certain data is leaked the more often it is processed in different ways. Namely, the vulnerability exploited in these schemes arises from mixing a varying component, the key, with a fixed component, the plaintext, in the process of re-keying. While such attacks that exploit the multiple encryption of fixed data using different keys seems implausible at first glance, there are indeed several practical use cases where such settings occur. Quite noteworthy, these settings are not limited to re-keying as a DPA countermeasure.

1) Communication: One example in practice are communication protocols. In SSL, for example, each session agrees on a new session key. As a result, sending the same data via multiple SSL sessions leads to the encryption of this data using different session keys and therefore enables side-channel plaintext-recovery attacks. One practical situation where this happens is an (embedded) web sever that receives multiple download requests for a certain file that is thus leaked using the presented attacks. Yet, for the cipher modes nowadays employed in such communication protocols it seems easier to use DPA to directly recover the key. However, using re-keying or leakage-resilient schemes as a countermeasure still allows to recover constant plaintexts.

Note that key wrapping does not solve the problem of plaintext recovery attacks in multi-party communication settings. Even though key wrapping guarantees that the plaintext itself is encrypted only once, the used data encryption key still needs to be encrypted for all communicating parties with their respective long-term keys. Therefore, side-channel plaintext-recovery attacks on the key wrapping procedure, where the data encryption key is the plaintext input, are still possible.

Another example in the area of communication is the handling of transmission errors. Many embedded devices do not have a full network stack covering error cases and therefore require the manual re-transmission of data when an error occurs. Due to constrained resources, the ciphertexts will not be held in memory, but the data will be encrypted again. Using re-keying as a DPA countermeasure for such devices will thus allow for recovering the plaintext information as shown before.

2) Memory Encryption: Re-keying nowadays also occurs in the setting of memory encryption. For example, in implementations of transparent RAM encryption, like recently introduced by Intel with SGX [4], it is common to choose a new random RAM encryption key on every system reboot. Therefore, every time (confidential) data is loaded into memory after startup, it is re-encrypted using a different key. This facilitates the attacks presented in this paper to learn the plaintext data. This can be particularly critical if there are long-term keys being loaded into the RAM.

Similar to the communication example, the application of re-keying and leakage-resilient encryption does not close this vulnerability. Namely, memory is typically encrypted using a block-wise granularity. Therefore, updating small amounts of data, e.g., a single byte, will cause a read-modify-write operation on the respective block to take place. In order for re-keying to work properly, the whole block must then be re-encrypted using a freshly chosen key, triggering the re-encryption of the unchanged data in the block. As a result, the presented attacks will not just work across several system boots, but also within a single session of the system.

We stress that a similar effect also occurs for RAM encryption using the counter mode such as in [4]. Hereby, the nonce input to the block cipher is composed of the block address and the respective block's counter. Therefore, whenever some data block is copied or written back to the memory, either the address or the respective block's counter changes, leading to a different pad, and thus re-encryption of the same data, again allowing for side-channel plaintext-recovery attacks.

D. Remarks and Countermeasures

Re-keying successfully prevents key recovery through DPA attacks without the need for DPA countemeasures in cryptographic implementations. Even more, leakage-resilient encryption nicely allows to give proofs of security in the presence of side-channel adversaries. However, the presented attacks also make clear that the re-keying approach may facilitate new attack scenarios that have been left unconsidered so far. This work gives an example of such scenario that designers and implementers need to be aware of.

The best countermeasure to the presented attack would be to avoid the re-encryption of constant data at all. However, multiple encryption of the same data using different keys frequently appears in practice. Therefore, contradicting the original intention of re-keying, adding countermeasures to the cryptographic implementation is one possible solution for these use cases. While these countermeasures cannot prevent the attack scenario completely, mechanisms like masking [15] can at least increase the attack order to render the attack complexity [2] for plaintext recovery too high.

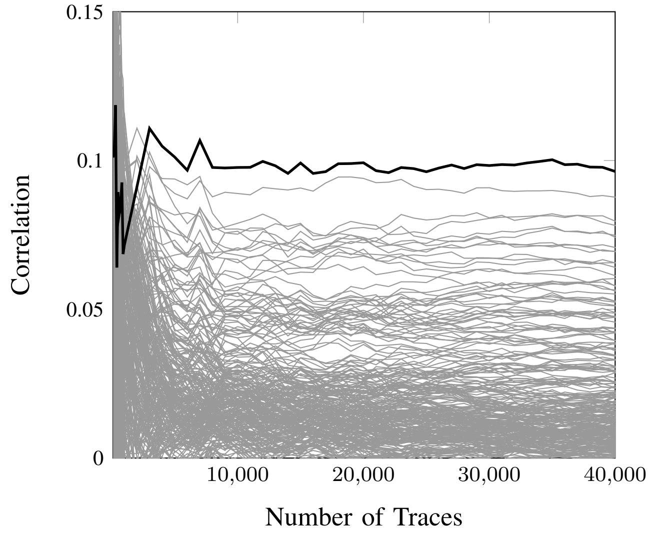

Fig. 4. Single plaintext byte result of a 1st-order DPA on the stream cipher. The correct plaintext byte is highlighted in bold.

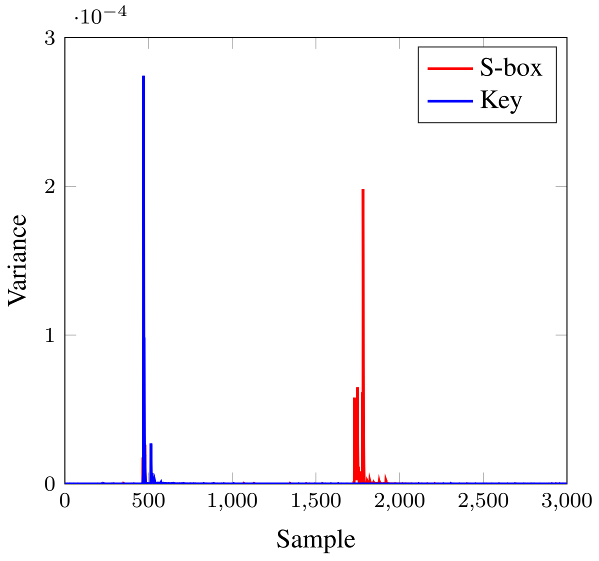

Fig. 5. Points of interest detection for the S-box and the key template. The main key leakage is located at sample 470. The S-box output leaks the most at sample 1782.

IV. Practical Evaluation

In this section, we practically verify the described attacks and present our results on both an FPGA and a microcontroller.

A. Stream Cipher Mode

We implemented the stream cipher depicted in Fig. 1 on the Sakura-G [17] side-channel evaluation board featuring a Xilinx Spartan 6 LX75 FPGA. The implementation uses a single AES-128 core as the encryption primitive E that computes one AES round per cycle and that is shared between the key update procedure and the pad computation. Therefore, the implementation natively processes plaintext blocks p_i of 128-bit size. Once the pad is computed and the input data block is ready, the pad is applied fully in parallel to the input. The implementation communicates with the host PC via the USB interface that emulates a virtual COM device. To ease the attack setup, the implementation also provides a dedicated signal to trigger the power measurements.

The implementation was operated at 24 MHz. For the power measurements, we sampled the signal at measurement point J3 using a LeCroy WP725Zi oscilloscope at 250 MS. The measurement point J3 gives an amplified signal of the voltage drop over a shunt resistor on the VCC line. We then performed the attack as described in Section III-A by using plaintext byte hypotheses and correlation as the statistical distinguisher. The attack could successfully identify all plaintext bytes in less than 10,000 traces. For example, Fig. 4 shows the results for a single plaintext byte that could be recovered using 3,000 traces already. However, to improve the results both measurement setup and trace processing are possible starting points.

B. Block Cipher Mode

The practical evaluation of the UPTA-like plaintext-recovery attack on the block cipher mode was performed using a ChipWhisperer-Lite [12] side-channel evaluation board, sampling at 29.5 MHz. As target board, an Atmel XMEGA128D4-U microcontroller, clocked at 7.4 MHz, was used. As in the original UPTA paper, the Hamming weight leakage model is well suited for this processor. On the software side, a byte-oriented C implementation of AES-128 from the AVR-Crypto-Lib [1] was chosen.

The actual attack was performed following a divide-and-conquer approach, where every plaintext byte is attacked in isolation. However, only a single template trace set as well as a single target trace set was used to attack all bytes.

In the first phase of the attack, byte-wise templates to classify the Hamming weights of both the key and the S-box output were trained using 30,000 power traces recorded during random encryptions with known plaintext and key. For each template, the points of interest were chosen by selecting the samples of a trace with the highest variance between the means of all Hamming weight classes of the respective template. The variance for the key and the S-box template is visualized for a single state byte in Fig. 5. The points of interest that contribute the most information to the desired templates can be clearly seen in the 3,000-sample long traces. In total, 50 points of interest were selected for each template.

In the second phase, every byte's key and S-box template was matched with every target trace. This results in probabilities for the modeled Hamming weights at the template positions. For each trace, the probabilities of a single byte's S-box template were then used to compute the probabilities for all potential plaintext and key values for the respective byte. Afterwards, the key dependency was removed by weighting the probabilities based on the result of the key template matching. As a result, probabilities for the different plaintext values remain which can be combined for all traces.

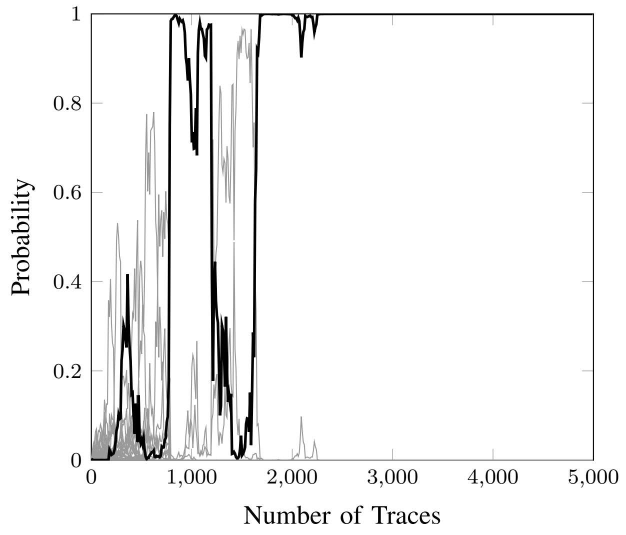

Fig. 6. Plaintext probabilities of an UPTA-like attack on one plaintext byte. The correct plaintext value is highlighted in bold black.

An exemplary development of these plaintext value probabilities is visualized for one byte of the plaintext in Fig. 6. In this figure, roughly 2,000 target traces are sufficient to uniquely determine the correct plaintext value. Across all plaintext bytes, most of the plaintext values could be determined with less than 5,000 traces.

Note that the presented attack is only supposed to prove that UPTA-like plaintext-recovery attacks are indeed possible and practical. However, the required number of traces should not be taken as reference for the expected attack complexity. Optimizing the attack would easily be possible using a more sophisticated measurement setup or by exploiting the leakage from additional samples within the traces.

V. Conclusion

In this paper, we investigated the side-channel security of frequent re-keying and leakage-resilient encryption. While such schemes have several advantages such as inherently preventing DPA on secret key material and giving provable leakage bounds without the need for a protected cryptographic implementation, we showed that schemes based on frequent re-keying do not sufficiently protect confidential plaintexts from DPA. In particular, whenever confidential data, e.g., a long-term key, is (re-)encrypted multiple times using different keys, the cryptographic device generates additional leakage on this data that an attacker can exploit. As a result, constant plaintexts encrypted using leakage-resilient stream ciphers are recovered using a standard, first-order DPA, and a template-based, second-order DPA can reveal plaintexts that are encrypted multiple times with a block cipher using different keys.

The consideration of plaintext confidentiality in the presence of side-channel adversaries thus reveals a weakness of current re-keying based schemes that designers and implementers need to be aware of. This issue is emphasized by several sensible applications where care has to be taken as these inherently perform re-encryption of constant plaintexts, e.g., multi-party communication and RAM encryption. We therefore conclude that, as opposed to the original idea of fresh re-keying and leakage-resilient encryption, cryptographic implementations with DPA countermeasures such as masking [15] are needed to avoid the leakage of plaintexts in all of these use cases inherently encrypting data using multiple keys.

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 numbers 845579 (MEMSEC) and 845589 (SCALAS).

References

[1] AVR-Crypto-Lib, "AVR-Crypto-Lib," 2016. [Online]. Available: https://trac.cryptolib.org/avr-crypto-lib

[2] 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*, 1999, pp. 398–412.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, 2002, pp. 13–28.

[4] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptology ePrint Archive*, vol. 2016, p. 204, 2016.

[5] N. Hanley, M. Tunstall, and W. P. Marnane, "Unknown plaintext template attacks," in *WISA 2009*, 2009, pp. 148–162.

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

[7] P. Kocher, "Leak-resistant cryptographic indexed key update," Mar. 25 2003, uS Patent 6,539,092. [Online]. Available: https://www.google.com/patents/US6539092

[8] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks – Revealing the secrets of smart cards*. Springer, 2007.

[9] M. Medwed, C. Petit, F. Regazzoni, M. Renauld, and F.-X. Standaert, "Fresh re-keying II: securing multiple parties against side-channel and fault attacks," in *CARDIS 2011*, 2011, pp. 115–132.

[10] M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni, "Fresh re-keying: Security against side-channel and fault attacks for low-cost devices," in *AFRICACRYPT 2010*, 2010, pp. 279–296.

[11] T. S. Messerges, "Using second-order power analysis to attack DPA resistant software," in *CHES 2000*, 2000, pp. 238–251.

[12] NewAE Technology Inc., "ChipWhisperer," 2016. [Online]. Available: https://newae.com/tools/chipwhisperer/

[13] O. Pereira, F.-X. Standaert, and S. Vivek, "Leakage-resilient authentication and encryption from symmetric cryptographic primitives," in *CCS 2015*, 2015, pp. 96–108.

[14] K. Pietrzak, "A leakage-resilient mode of operation," in *EUROCRYPT 2009*, 2009, pp. 462–482.

[15] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *EUROCRYPT 2013*, 2013, pp. 142–159.

[16] P. R. Sami Saab and C. Hampel, "Side-channel protections for cryptographic instruction set extensions," Cryptology ePrint Archive, Report 2016/700, 2016, http://eprint.iacr.org/2016/700.

[17] Satoh Lab./UEC, "Sakura G," 2014. [Online]. Available: http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html

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

[19] 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*, vol. 10, no. 3, pp. 519–528, 2015.