Fresh Re-Keying II: Securing Multiple Parties against Side-Channel and Fault Attacks

Marcel Medwed, Christoph Petit, Francesco Regazzoni, Mathieu Renauld, François-Xavier Standaert

UCL Crypto Group, Université catholique de Louvain.
Place du Levant 3, B-1348, Louvain-la-Neuve, Belgium.

Abstract. Security-aware embedded systems are widespread nowadays and many applications, such as payment, pay-TV and automotive applications rely on them. These devices are usually very resource constrained but at the same time likely to operate in a hostile environment. Thus, the implementation of low-cost protection mechanisms against physical attacks is vital for their market relevance. An appealing choice, to counteract a large family of physical attacks with one mechanism, seem to be protocol-level countermeasures. At last year's Africacrypt, a fresh re-keying scheme has been presented which combines the advantages of re-keying with those of classical countermeasures such as masking and hiding. The contribution of this paper is threefold: most importantly, the original fresh re-keying scheme was limited to one low-cost party (e.g. an RFID tag) in a two party communication scenario. In this paper we extend the scheme to n low-cost parties and show that the scheme is still secure. Second, one unanswered question in the original paper was the susceptibility of the scheme to algebraic SPA attacks. Therefore, we analyze this property of the scheme. Finally, we implemented the scheme on a common 8-bit microcontroller to show its efficiency in software.

Keywords: Side-channel attacks, Fault attacks, Re-keying, Masking, Shuffling

1 Introduction

Ensuring security against physical (e.g. side-channel and fault) attacks is an increasingly important challenge for cryptographic embedded devices. It is specially critical in applications requiring low-cost implementations. Indeed, most solutions that have been introduced in the literature to prevent physical attacks imply significant performance penalties, that may be too high for certain applications. For example, improving security against side-channel attacks is frequently obtained by applying masking [4,10] or hiding [28,29] to the implementations. Additionally, preventing fault attacks requires to include fault detection mechanisms in the circuits [2,12]. In both cases, implementing these countermeasures implies significant area or time overheads. Unfortunately, low-cost devices such as smart cards, RFID tags or sensor nodes are also the ones for which the threat of a physical attack is the most realistic, when operated in hostile environments.

Since directly protecting cryptographic algorithms, such as the AES Rijndael, against side-channel and fault attacks is difficult, an alternative approach is to design encryption mechanisms that can be more easily protected in this case. One important line of research, denoted as leakage-resilient cryptography, aims at combining such new designs with a proof of security, using the formalism of modern cryptography. For example, this approach has been applied to new stream cipher constructions [6,21,31]. But as discussed in [26,27], present proof techniques have limited practical relevance, as they need to rely on assumptions that may be difficult to fulfill by hardware designers. More important for our present focus, these schemes are also quite inefficient, as their initialization implies the execution of a pseudorandom function [5,27], which typically requires the execution of n AES encryptions, with n the bit size of the initialization vector. A more practically-oriented line of research has been trying to embed the block cipher in some protocol that makes it easier to protect. An example of such an approach is the application of "all-or-nothing" transforms [17]. Here, the idea is to modify the plaintexts and ciphertexts according to a (low-cost) mapping, therefore preventing attacks based on known plaintexts/ciphertexts. Whereas the scheme is efficient for long messages, the initialization effort might render it impractical for smartcard and RFID applications. More recently, a fresh re-keying scheme was presented at Africacrypt 2010 [19]. It combines the re-keying used in leakage-resilient cryptography with easy to protect low-cost mappings in order to remove the initialization overhead.

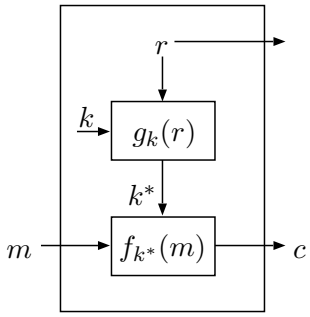

Fig. 1. Original (one party) fresh re-keying scheme.

The basic principle of this fresh re-keying scheme, that we further investigate in this paper, is pictured in Figure 1. It essentially encrypts every message block m under a fresh session key k^*, with a block cipher. The session key is generated from the master key k and a public random nonce r, with a function g. At first sight, it may seem that one just shifts the problem of protecting the block cipher f against physical attacks to the one of protecting the function g. Interestingly, it is argued in [19] that a proper selection of g may lead this scheme to be significantly easier to protect against such attacks than the underlying block cipher

f. Namely, g does not need to be cryptographically strong. It should only ensure a strong diffusion between the master key k and the session keys k^*, while being easy to protect against Differential Power Analysis (DPA). Assuming that these conditions are respected, the only additional requirement for the global scheme of Figure 1 to be secure against side-channel attacks is that the block cipher f resists against SPA (i.e. side-channel attacks exploiting a single measurement) - an easier task than preventing DPA. In addition, the re-keying mechanism naturally prevents Differential Fault Analysis (DFA) and was shown to have limited hardware cost. Following these interesting features, the present paper aims at extending this analysis in three main directions:

1. In view of the difficulty to design leakage-resilient pseudorandom permutations [5], the claim that a simple fresh re-keying scheme could be secure against a wide class of side-channel and fault attacks appears quite provocative. In this respect, one important problem that was left open in the previous work of Africacrypt is to evaluate if the cryptographically weak function g could not be the target of advanced side-channel attacks (such as [20,23]), taking advantage of its simple algebraic expression. We evaluate this concern and suggest that it can be efficiently prevented by shuffling the implementation [11] (which is anyway required to prevent SPA).
2. The original Africacrypt scheme was limited to the protection of one party in a communication protocol. This is because in Figure 1, it is crucial that the random nonce r is chosen inside the protected device and cannot be manipulated from outside. For example, keeping r constant would completely break the re-keying. While there exist many practical scenarios in which such an asymmetric type of security is realistic (e.g. RFID readers can be protected with expensive means, only tags have strong cost constraints), there also exist many where protecting multiple parties is necessary. For instance, trends can be observed in which also constrained devices become readers, e.g. mobile phones in NFC applications [22]. Furhtermore, automotive applications, where many low-cost devices need to communicate securely [13], provide a strong motivation for developing such tools. As a result, we extend the re-keying scheme to multiple parties. In particular, this allows all involved parties to derive a common session key in a side-channel protected manner[1]. We show that the security of the new proposal is similar to the one of the single-party case, while its performances only decrease linearly with the number of parties.

[1] This context can be seen as reminiscent of group key distribution. However, our objectives are different in the sense that group key distribution typically aims at ensuring cryptographic properties such as forward security, whereas our fresh re-keying scheme "only" aims at preventing successful side-channel attacks against the master key. It is an interesting open question to investigate whether one could combine strong physical security guarantees and, e.g. forward secrecy. We note that it would be surprising, as the relatively simple (linear) nature of the g function is central in making it easy to protect against side-channel attacks.

3. Finally, since the use of dedicated solutions for protecting block cipher implementations is mainly justified by strong cost constraints, we evaluate the performances of our proposal in an AVR microcontroller. Our implementations consider different levels of masking and shuffling. We confirm their low-cost nature by comparing them with the masked AES Rijndael software implementation proposed at CHES 2010 [24]. These results nicely complement the ones in [19], where hardware implementations were considered.

2 Background: the Africacrypt 2010 scheme

The original scheme, as depicted in Figure 1, describes a physically secure encryption which is for instance carried out inside an RFID tag. It consists of the re-keying function g and a cryptographically secure encryption function f. At every invocation of the scheme, g uses a symmetric master key k and a fresh random nonce r to obtain a session key $k^* = g_k(r)$. The random nonce r is generated inside the device but is made public afterwards. The session key is then used by f to perform an encryption. Thus, the ciphertext c is obtained as $c = f_{k^*}(m)$. The function f is instantiated with a standardized algorithm, in our examples we use the AES Rijndael. For such algorithms, it is also well known how to protect them against SPA attacks (e.g. by shuffling [7,11]). Thus, the main concern is a careful choice of g. In [19], g was chosen as:

$$\begin{aligned} g : \left(\mathrm{GF}(2^8)[y]/(y^{16}+1)\right)^* \times \left(\mathrm{GF}(2^8)[y]/(y^{16}+1)\right) &\to \mathrm{GF}(2^8)[y]/(y^{16}+1) \\ : (k, r) &\to k \cdot r. \end{aligned}$$

That is, g takes two 128-bit operands, represented by polynomials in y of degree 15 and coefficients in $\mathrm{GF}(2^8)$, and performs a polynomial multiplication to obtain the session key. The key k is constrained to the invertible elements of $\mathrm{GF}(2^8)[y]/(y^{16}+1)$; otherwise any master key k that is a divisor of 0 would only lead to session keys that are also divisors of 0. As shown in the original paper, this choice of g has some very advantageous properties. Most important, it provides sufficient diffusion such that "divide-and-conquer" attacks on the master key become computationally infeasible. Second, the function has homomorphic properties which allows sound protection against higher-order differential attacks [4]. In particular, the key can be split into $t+1$ shares:

$$\left(k_1 = b_1, k_2 = b_2, \cdots, k_t = b_t, k_{t+1} = k \oplus \bigoplus_{i=1}^{t} b_i\right),$$

in order to obtain the session key as:

$$k^* = \bigoplus_{i=1}^{t+1} r \cdot k_i.$$

Such a masking thwarts t^{th}-order attacks[2]. Third, as the arithmetic in the proposed algebra is carry-free, shuffling can be efficiently applied to thwart SPA attacks. Finally, the function can also benefit from hardware countermeasures (such as secure logic styles) at low-costs thanks to its regular structure.

3 Security of g against algebraic SPA

In this section, we analyze the security of the original fresh re-keying scheme from [19] against side-channel attacks that exploit the algebraic structure of the target algorithm. We first show that, if no attention is paid, block ciphers protected by a re-keying scheme can still be vulnerable to algebraic side-channel attacks, due to the simple algebraic structure of the re-keying function g. Next, we suggest that a shuffling of the operations would prevent this kind of attacks (as it prevents most algebraic side-channel attacks, in fact).

In the rest of the paper, we will assume that the encryption function f is the AES Rijndael, and that the side-channel attacks are performed in a known plaintext context. In this setting, the re-keying function g can be written as a system of linear equations over GF(2^8). Let us call $\mathbf{k}$ the vector containing the 16 bytes $k_{.,j}$ ($0 \leq j \leq 15$) of the master key k, and $\mathbf{k_i^*}$ (resp. $\mathbf{r_i}$) the vector containing the 16 bytes $k^*_{i,j}$ (resp. $r_{i,j}$) of the i^{th} session key generated from the same master key (resp. nonce used). The bytes of the nonces are known, the bytes of the master and session keys are unknown. Hence, the system of equations linking a session key k_i^* to the master key k and the nonce r_i is:

$$\mathbf{R_i}.\mathbf{k} = \mathbf{k_i^*} \tag{1}$$

$$\begin{bmatrix} r_{i,0} & r_{i,15} & r_{i,14} & \cdots & r_{i,1} \\ r_{i,1} & r_{i,0} & r_{i,15} & \cdots & r_{i,2} \\ r_{i,2} & r_{i,1} & r_{i,0} & \cdots & r_{i,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{i,15} & r_{i,14} & r_{i,13} & \cdots & r_{i,0} \end{bmatrix} \cdot \begin{bmatrix} k_{.,0} \\ k_{.,1} \\ k_{.,2} \\ \vdots \\ k_{.,15} \end{bmatrix} = \begin{bmatrix} k^*_{i,0} \\ k^*_{i,1} \\ k^*_{i,2} \\ \vdots \\ k^*_{i,15} \end{bmatrix}. \tag{2}$$

This system can be represented using a block matrix:

$$\left[\mathbf{R_i} \mid \mathbf{I}\right] \cdot \left[\frac{\mathbf{k}}{\mathbf{k_i^*}}\right] = \left[0\right], \tag{3}$$

with $\mathbf{I}$ the identity matrix. It is an homogeneous system of 16 linear equations in 32 unknown variables (the bytes of $\mathbf{k}$ and $\mathbf{k_i^*}$). For each additional session key produced from the same master key, we can add 16 new equations in 16 new

[2] In the original scheme, the nonce is shared rather than the key. However, this leads to a first-order leakage if a master key byte is zero, similar as observed in [9].

variables (the bytes of $\mathbf{k^*_{i+1}}$) to the system:

$$\begin{bmatrix} \mathbf{R_1} & \mathbf{I} & 0 & \cdots & 0 \\ \mathbf{R_2} & 0 & \mathbf{I} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{R_n} & 0 & 0 & \cdots & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{k} \\ \mathbf{k^*_1} \\ \mathbf{k^*_2} \\ \vdots \\ \mathbf{k^*_n} \end{bmatrix} = [0] \,. \tag{4}$$

As such, this system is underdefined. We need at least 16 bytes of information about the session keys $\mathbf{k^*_i}$ in order to identify the value of the master key k[3]. The easiest way to solve this system is thus to find 16 additional linear equations, involving one or more bytes of $\mathbf{k^*_i}$. For this purpose, a straightforward approach is to use side-channel leakage in order to learn session key bytes. Usually, side-channel leakages do not provide the exact data processed by a device, but some information about it. For example, one could assume that three leakage points are obtained, for each session key byte, as illustrated in Figure 2. They correspond to (1) the output byte of the fresh re-keying multiplication itself ($k^*_{i,j}$), (2) the XOR operation between this byte and a known plaintext byte ($P_{i,j} \oplus k^*_{i,j}$), and (3) the output of the AES S-box ($\mathsf{S}(P_{i,j} \oplus k^*_{i,j})$). By combining these three Hamming weight values, it is possible to identify a unique valid value for the session key byte in approximately 16% of the cases. As a result, only 11 encryptions are required to get 16 session key bytes with probability higher than 0.99. Naturally, these simple estimations assume that the Hamming weights are perfectly recovered, while actual attacks may be affected by noise. Nevertheless, they show that algebraic attacks must be considered in the analysis of fresh re-keying schemes.

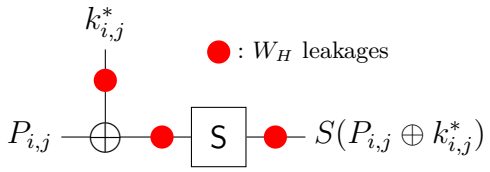

Fig. 2. Three interesting leakage points in the first AES round.

Another possibility is to use Side-Channel Collision Attacks (proposed against the DES in [25], and enhanced in [14]). For example, in [3], the author introduces the notion of *generalized internal collisions* for the AES. A generalized internal collision occurs when two AES S-boxes are evaluated on the same input. These two S-boxes can be located in the same round, in two different rounds or even in two different encryptions. If all the AES S-boxes are implemented in a similar way, the computation of the same value should give rise to similar power consumption traces, making the collision detection possible. If a collision is detected

[3] ignoring the fact that the entropy of k is slightly less than 16 bytes.

between two S-boxes of the first AES round, we translate this information into:

$$P_{i_1,j_1} \oplus k^*_{i_1,j_1} = P_{i_2,j_2} \oplus k^*_{i_2,j_2} \Leftrightarrow k^*_{i_1,j_1} = P_{i_1,j_1} \oplus P_{i_2,j_2} \oplus k^*_{i_2,j_2},$$

Hence, 16 collisions would be sufficient to solve the system of Equation (4). The average number of collisions for n_e encryptions can be computed using the birthday paradox (see [1]): $\mathbf{E}(N(n_e)) = 16n_e - 256 + 256(1 - 1/256)^{16n_e}$. Simulations show that less than 10 encryptions are sufficient to reach the 16 required collisions with high probability.

Summarizing, a straightforward implementation of the re-keying scheme is susceptible to algebraic type of attacks. Importantly, this does not contradict the security analysis in [19], as these attacks can be seen as SPA against the AES function. However, they clearly exhibit the importance of explicit SPA resistance of the block cipher used in the scheme. In a very similar way, fresh re-keying implies executing the AES key scheduling algorithm, in which case Mangard's SPA attack is another possible threat [15] (or similarly [30]).

On the positive side, and as discussed in this previous work, SPA attacks are relatively easy to counteract (compared to DPA). First, they are typically applicable in software implementations with small (e.g. 8-bit) buses. Second, in case small data buses are considered, they are efficiently prevented by shuffling the operations [7,11]. In this respect, it is worth mentioning that, in order to allow secure implementations, the interaction between the f and g functions needs to remain shuffled (which will be ensured in the software implementations of Section 5). Eventually, we also note that the AES Rijndael may not be the most suitable block cipher for efficient shuffling, because of its non-regular and sequential key scheduling algorithm.

4 Extending the Africacrypt scheme to n parties

In this section, we present two possible extensions in order to enable the use of fresh re-keying amongst n parties. The first scheme is a straightforward extension of the original scheme and uses n master keys. The second one allows the same functionality but uses only one master key. We show that their security is similar to the one of the original scheme. This will be done in two steps. First, we demonstrate that the session keys are still uniformly distributed and cannot be biased by an adversary controlling all but one of the nonces involved in the re-keying. Second, we argue that the requirement for the diffusion between k and k^* which is the core of the security argument in [19] is still fulfilled.

4.1 Scheme 1: Using n master keys

The first solution to the extension problem consists in instantiating n original fresh re-keying schemes with independent master keys. Every party possesses all n master keys and serves as master (that is, it generates the nonce) for one instance and as slave (that is, it receives the nonce from outside) for all

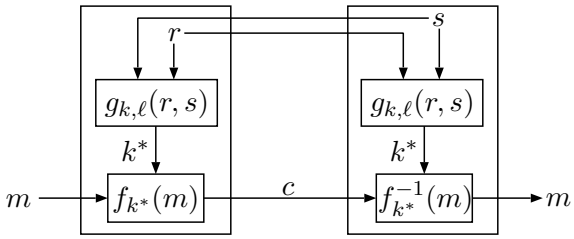

Fig. 3. Basic extension to two parties using two master keys.

other $n-1$ instances. The shared session key is derived as the sum of all the n independent session keys, from the n instances. Like in the original scheme, the master keys are constrained to the invertible elements of $\mathrm{GF}(2^8)[y]/(y^{16}+1)$. Figure 3 illustrates the principle for two master keys k and ℓ. In a first step, the parties generate and exchange the nonces r and s. Next, the session key is computed as $k^* = k \cdot r + \ell \cdot s$.

4.2 Scheme 2: Using a single master key

The second scheme is similar to the first scheme, except that only one master key $k \in \left(\mathrm{GF}(2^8)[y]/(y^{16}+1)\right)^*$ is shared amongst the parties. In particular, every party is enumerated with a unique value $i \in [1, n]$. The session key is derived as:

$$k^* = \bigoplus_{i=1}^{n} r_i \cdot k^i,$$

where r_i denotes the nonce generated by party i and k^i denotes the i^{th} power of the master key. Note, that in this scheme, the order of k needs to be greater n. For two parties the session key is derived as $k^* = k \cdot r + k^2 \cdot s$. As for the i values, we assume that the parties are enumerated statically. Such an assumption already covers many scenarios. In a tag-reader scenario, the roles can be assigned in the specification. In car like scenarios, the devices are rarely replaced and thus can be set up by the manufacturer or a certified garage. However, ad-hoc negotiation of the i values would be interesting to cover arbitrary applications. We leave the evaluation and selection of appropriate negotiation protocols for further research.

4.3 Security model

The challenge when designing an n-party fresh re-keying scheme lies in the fact that each party is provided with $n-1$ nonces from outside. Thus, an adversary potentially has control over $n-1$ out of n nonces. Compared to the single-party case, there are two main properties that need to be verified. First, it should still hold that the adversary controlling the external nonces is not able to significantly bias the session key distributions (e.g. he should not be able to set them to

a constant value). Second, it should also hold that there is a strong diffusion between the master key and the session keys (i.e. it should not be possible to guess one byte of session key, excepted by guessing most of the master key). In the following, we show that these conditions are respected in a model where the adversary can eavesdrop the communications and modify the nonce values received by each party. By contrast, he cannot access the master key and he cannot change the nonce value that is generated internally by the target device.

4.4 Security of Scheme 1

Our analysis will be done in two steps. First, and for illustration, we will consider an adversary trying to set the session key to a constant value. We show in Lemma 1, that this is impossible without already possessing substantial information about the master keys. In other words, in order to fix the session key, the quotient of the two master keys has to be guessed correctly. This gives intuition that biasing the session keys is a difficult problem. Next, we consider a more realistic adversary who is just trying to bias the session key distribution. Lemma 2 shows that, whatever the distribution of the external nonces, the session key is close to uniformly distributed given that the nonce generated within the target device is uniformly distributed.

For simplicity, the following lemmata consider the two-party case. Proofs can be extended easily to the general case. Let r_j and s_j be the nonce values generated during the jth execution of the re-keying, and let k_j^* be the corresponding session key. We prove the security of the party generating s_j.

Lemma 1 *If the adversary is able to keep k^* constant, then he also knows ℓ/k. Reciprocally, if the adversary knows k/ℓ, then he can keep k^* constant.*

Proof: Suppose:

$$k^* := kr_1 + \ell s_1 = kr_2 + \ell s_2.$$

Then assuming $s_1 + s_2$ is an invertible element[4], we get:

$$\ell/k = (r_1 + r_2)/(s_1 + s_2),$$

and the adversary can compute this value since he knows r_1, r_2, s_1, s_2. On the other hand, if the adversary knows ℓ/k, then he can choose:

$$r_2 := r_1 + (s_1 + s_2)\ell/k, \tag{5}$$

in order to keep k^* constant. □

[4] The proof can be slightly adapted when $y + 1$ divides $s_1 + s_2$, either by considering the quotient ℓ/k instead, or by dividing both $r_1 + r_2$ and $s_1 + s_2$ by a common power of $y + 1$. Alternatively since the nonces are randomly chosen and a large majority of them produce an invertible $s_1 + s_2$, the adversary can simply wait until $s_1 + s_2$ is invertible.

More generally, let us now define $\delta := (r_1 + r_2)/(s_1 + s_2)$. Since the adversary knows r_1, s_1 and s_2, choosing r_2 is equivalent to choosing δ (from his point of view). In the following lemma, we show that the distribution of $k_2^* + k_1^*$ cannot be biased by the adversary unless he has some information on the value $\delta + \ell/k$.

Lemma 2 *Let the nonces be uniformly distributed and further let* $0 \leq e < 16$ *be the maximal power of* $y + 1$ *dividing* $\ell/k + \delta$*. Then the value* $k_{12}^* := k_2^* + k_1^*$ *is uniformly distributed in the set:*

$$\mathcal{K}_e := \{(y+1)^e p(y) | \deg(p) < 16 - e, (y+1) \not| p\},$$

given that the nonces s_i *are uniformly distributed.*

PROOF: We have:

$$\begin{aligned} k_2^* &= kr_2 + \ell s_2 \\ &= kr_1 + \ell s_1 + k(r_2 + r_1) + \ell(s_2 + s_1) \\ &= k_1^* + k(\ell/k + \delta)(s_1 + s_2), \end{aligned}$$

hence:

$$k_2^* + k_1^* = (y+1)^e k \frac{\ell/k + \delta}{(y+1)^e}(s_2 + s_1).$$

By definition, $k\frac{\ell/k+\delta}{(y+1)^e}$ is an invertible element of $\mathrm{GF}(28)[y]/(y^{16}+1)$. As a consequence, for any $k_{12}^* = \tilde{k}_{12}(y+1)^e \in \mathcal{K}_e$, we have:

$$k_2^* + k_1^* = \tilde{k}_{12}(y+1)^e \Leftrightarrow s_2 \in \left\{ s_1 + \tilde{k}_{12} k^{-1} \frac{(y+1)^e}{\ell/k + \delta} + (y+1)^{16-e} q(y) | \deg(q) < e \right\}.$$

We see that each k_{12} value corresponds to 2^e values for s_2. Since s_2 is chosen randomly and cannot be controlled by the adversary, we obtain the result. □

Intuitively, Lemma 2 implies that the adversary cannot affect the distribution of $k_2^* + k_1^*$ without affecting the distribution of ℓ/k. Since the adversary has a priori no information at all about ℓ/k, all he can do is to try random values for δ and hope that he guessed ℓ/k up to some large power of $y + 1$. This way, the adversary has no way to decrease the entropy of $k_2^* + k_1^*$. Indeed, $k_2^* + k_1^*$ only belongs to $\mathcal{K}_e$ (that has size $(2^8 - 1)2^{8(15-e)}$) with a probability $2^{-8e} - 2^{-8(e+1)}$.

4.5 Security of Scheme 2

Scheme 2 can be seen as a particular case of Scheme 1 where $\ell = k^2$:

$$k_i^* = kr_i + k^2 s_i.$$

Since the adversary could target both parties, we consider two cases:

1. s is chosen randomly and then r is chosen by the adversary.

2. r is chosen randomly and then s is chosen by the adversary.

The arguments of Section 4.4 also apply to both cases. In the first case, we define $\delta := (r_1 + r_2)/(s_1 + s_2)$ and obtain:

$$k_2^* + k_1^* = k(s_1 + s_2)\,(\delta + k)\,.$$

Using an argument similar to the one in Lemma 2, we have that the adversary cannot modify the distribution of $k_2^* + k_1^*$ without modifying the distribution of $\delta + k$, which requires knowing some information about k. In the second case, we define $\delta := (s_1 + s_2)/(r_1 + r_2)$ and obtain:

$$k_2^* + k_1^* = k^2(s_1 + s_2)\left(\delta + k^{-1}\right).$$

Again by adapting Lemma 2, we see that the adversary cannot modify the distribution of $k_2^* + k_1^*$ without modifying the distribution of $\delta + k^{-1}$, which requires knowing some information about k.

4.6 Security against divide-and-conquer attacks

Most DPA attacks considered in the literature are based on a divide-and-conquer approach. In [19], it is argued that fresh re-keying prevents the application of such a strategy, because every bit of the session key is a sum of half of the master-key bits on average. In addition, since DPA attacks usually require more than one power trace to be successful, this subset changes for every encryption, as long as the nonces are uniformly distributed. Thus, after only a few encryptions, the union of those subsets almost covers the whole master key. In practice, this means that an adversary needs to guess almost all 128 master key bits in order to build 8-bit hypotheses for the session key. More generally, it was shown in [19] that, whatever are the traces selected by the adversary (e.g. those obtained from low Hamming weight nonces), it remains computationally intensive to guess one byte of the session keys. We already showed in the previous section that the control over the external nonces does not allow the adversary to efficiently bias the session keys. As a result, the argument of security against divide-and-conquer attacks for the single party case directly extends to the multi-party case. The best adversarial strategy is to set all the nonces under control to zero and, for the remaining (uncontrolled) nonce, to apply the strategy described in the Africacrypt 2010 paper.

4.7 SCA security of the extended function g

Since the function g has a homomorphic property it can be easily shown that there is no leakage of order smaller $t + 1$ with t as the masking order. Furthermore, as shown in the previous subsection, for the final session key, we can rely on the diffusion property after unmasking. However, what has to be considered separately is the unmasking of the different session key contributions. In particular, if an adversary chooses a nonce s in such a way that only one byte is

non-zero, every byte of the product involving s depends only on one master-key byte. Thus, after unmasking the product, it is possible to directly build hypotheses for a single master key byte. When implementing the full scheme, one would process the uniform nonce first and then accumulate the shares of the biased product to obtain the final session key. For example let $x = r * k$ and $y = s * \ell$ and further let r be the uniformly distributed nonce and s be the biased nonce with only one non-zero byte. Finally, let x be shared as $x = x_1 + x_2 + x_3$. Then after the first multiplication, the session-key register will hold the intermediate values $(1)\, x, (2)\, x + y_1, (3)\, x + y_1 + y_2$, and finally $(4)\, x + y$. This shows that independently of the masking order, if we implement the scheme straightforwardly there will always be a second-order leakage, namely the joint leakage of $(1)\, x$ and $(4)\, x + y$. Furthermore, these leakage samples are only shuffled over 16 positions.

In order to fix this flaw, the unmasking has to be done in an interleaved manner. That is, first all shares with index one are accumulated, afterwards all shares with index two and so on. For the above example, this means that the session key register will hold the values $(1)\, x_1, (2)\, x_1 + y_1, (3)\, x_1 + y_1 + x_2, (4)\, x_1 + y_1 + x_2 + y_2, (5)\, x + y_1 + y_2$, and finally $(6)\, x + y$. In order to attack y an adversary would need to for instance use the joint distribution of $(6)\, x + y$, $(1)\, x_1$, x_2, and x_3, the dimension of which is greater than t.

5 Software implementation in an AVR microcontroller

In this section we will discuss an 8-bit software implementation of the fresh re-keying scheme. As a target platform we chose the AVR microcontroller architecture and in particular the model ATmega128. We target this architecture since it is very common in constrained platforms such as smart cards, which are the ones which will benefit most from our protection scheme.

The selected AVR microcontroller features an 8-bit datapath, 32 general purpose registers and four kilobytes of SRAM. Furthermore, we can rely on four kilobytes of EEPROM and 128 kilobytes of Flash program memory. Finally, most of the instructions finish within a single clock cycle. Unfortunately, our target platform does not feature a hardware random number generator (RNG). Therefore, we state the performance of our implementation in two numbers. First the number of clock cycles assuming that reading a random byte takes 2 clock cycles and second the explicit number of RNG calls needed throughout the execution. This allows a cycle estimation for platforms which do provide such an RNG.

5.1 Multiplication

The basic implementation of the function g relies on a product scan algorithm to implement the polynomial multiplication. That is, every byte c_i of the product $c = a * b$ is calculated as $c_i = \sum_{j=0}^{15} a_{(i-j \mod 16)} * b_j$. The GF($2^8$) multiplication is implemented using two lookup tables, a 256 byte large logarithm table (`LOG`) and the corresponding inverse logarithm table (`ILOG`). In the logarithm table,

$-\infty$ (`LOG[0x00]`) is encoded as `0xff`. In order to get a conditional-branch free multiplication, we rely on Algorithm 1.

Algorithm 1 Branch-free multiplication

Require: a' $\leftarrow$ LOG[a], b' $\leftarrow$ LOG[b] with a, b $\in$ GF(2^8)
Ensure: pp = a $*$ b
1: (carry, c) $\leftarrow$ a$'$ + b$'$
2: pp $\leftarrow$ ILOG [c $-$ carry] // mod 255, ILOG [255] := ILOG [0]
3: zfa $\leftarrow$ ZF(a$'$ $-$ `0xff`), zfb $\leftarrow$ ZF(b$'$ $-$ `0xff`) // ZF ... AVR's zero flag
4: pp $\leftarrow$ pp $\cdot$ (1 $-$ (zfa OR zfb))
5: **return** pp

5.2 Shuffling of the fresh re-keying

The function g needs to be shuffled for two reasons. First, the noise introduced by shuffling is vital for the impact of masking. Second, in order to prevent attacks as described in Section 3, the position of the leakage samples within the power trace needs to be uncertain. Since g relies on carry-free arithmetic, shuffling can be done efficiently. In this section we describe the two different parts of the polynomial multiplication where we apply shuffling and also discuss two different shuffling techniques. Thus, we provide four different levels of shuffling.

The most important part of the algorithm to shuffle is the processing of the product bytes itself, that is the order in which the c_i bytes are computed. This significantly reduces the information an SPA adversary can learn about the product. In particular, it leaves such an adversary with a probability of 1/16 for gaining information about a specific byte. In addition, it is also possible to shuffle the processing of the partial products corresponding to one product byte. Also here, 16 positions can be shuffled, which results in a total randomization of the partial products over 256 positions.

Internally, both parts of the algorithm are implemented as loops where each iteration operates on independent data. Thus, the simplest way of shuffling is to add an offset modulo 16 to the loop counter. For the bytes of the product, we just start from c_r and run through to $c_{(r+15 \mod 16)}$, for a randomly chosen r. For the partial products of one product byte, and a randomly chosen starting offset s, this would mean $c_i = \sum_{j=0}^{15} a_{(i-(j+s \mod 16) \mod 16)} * b_{(j+s \mod 16)}$. From an overhead point of view, shuffling using a random starting index is negligible.

As a second shuffling method, we indirectly address the product bytes and the operand bytes. The bytes of the product are thus addressed by c_{r_i} where r is now a random permutation of the sequence $0, \ldots, 15$. For the partial products of one product byte this would mean $c_i = \sum_{j=0}^{15} a_{(i-s_j \mod 16)} * b_{s_j}$, for a random permutation s. The generation of such a random permutation is significantly more expensive than generating a random starting offset as in the first strategy. In particular, we start with the sequence $0, \ldots, 15$ and sample n random bytes. For every such random byte we take the four MSbs as position one and the four

LSbs as position two. Afterwards, we swap the entries at those positions. In general, the generation of a permutation has a quadratic complexity, thus $n = 16^2$. However, this results in 7 160 clock cycles just to generate one permutation of which we need 17 in total. In order to allow a more flexible tradeoff, we also introduce an intermediate solution which sets n to 16^2 for the first permutation and derives all consecutive ones from the first permutation by swapping 16 random pairs.

For the generation of noise, the two approaches yield similar results, however, if the aim is to prevent SPA attacks, there is a significant difference. In general, if all 16 product bytes need to be recovered within a single trace, the time complexity is 16 for the random starting index and 16! for the full permutation. Since the described SPA attack needs $t > 1$ traces, we can estimate the time complexity of the attacks as 16^t and $16!^t$ respectively. Therefore, for small values of t, a random starting index might not be sufficient to shuffle the product bytes.

5.3 Shuffling of the AES

The AES Rijndael algorithm features a 16 bytes state, thus shuffled implementations usually shuffle these 16 positions plus sometimes additional dummy operations. However, when it comes to the key schedule, only four positions can be shuffled. Therefore, one option in a standard implementation is to store all the round keys in the device instead of computing them on the fly. Unfortunately, we cannot rely on this strategy as we never reuse the same key. As a result we have to introduce three additional key schedules which operate on random dummy data. This allows us again to shuffle over 16 positions and thus achieve the same security level as for the remainder of the algorithm. We implemented a version of the AES which uses fully indirect addressing for all the operations. This allows having complete control over the processed bytes, but negatively affects the performance. In fact our implementation can complete a whole encryption, including one permutation generation, in 30,713 clock cycles.

5.4 Performance results

Table 1 summarizes the different performance numbers for a single polynomial multiplication when implemented using different levels of shuffling. RSI stands for random starting index and RP-n for random permutation, the generation of which used n swap operations. In addition we state the number of RNG calls. In our implementation, those calls are included with a factor of two in the number of clock cycles. It can be seen that the execution time of one multiplication heavily depends on the way the shuffling is implemented. In order to prevent SPA attacks and to add sufficient noise against DPA attacks we rely on the third solution (RP-256 + RSI) for all further evaluations.

Implementing the masking is equivalent to performing $t + 1$ multiplications for t^{th}-order masking. The reason why first-order masking does not take twice as long as the unmasked version is that we reuse the permutation for the product bytes. We settled for this solution as it provides a good security vs. performance

Product bytes shuffling	Partial products shuffling	Clock cycles	Calls to RNG	Code size (bytes)	RAM usage (bytes)
-	-	13,400	16	754	48
RSI	RSI	15,032	33	760	48
RP-256	RSI	22,199	288	904	64
RP-256	RP-16	29,688	528	1008	80
RP-256	RP-256	137,208	4368	1008	80

Table 1. Implementation results for the polynomial multiplication for different levels of shuffling: RSI = random starting index, RP-16 = random permutation generated with 16 swap operations, and RP-256 = random permutation generated with 256 swap operations.

tradeoff. However, the scheme could be implemented using various other tradeoffs, which generally compare favorably with the straightforward protection of a block cipher against side-channel attacks, as will be discussed next.

In order to finalize the two n-party schemes, we need to perform n masked multiplications for Scheme 1 and $2n-1$ multiplications for Scheme 2, if the powers of k are computed on the fly. If the powers of k are precomputed, the execution times of the schemes are equivalent. As for the RAM usage, we need n times the RAM, as the n multiplications need to be performed in parallel to allow interleaving the shares (as discussed in Section 4.7). Only the 16 bytes for the session key accumulation can be shared.

Finally, we compare the total performance of our scheme with the provable secure AES implementation of CHES 2010 [24]. Their results have been recently improved [8] but this paper states only figures for the masked S-box and not for the entire cipher, therefore we still use the figures from the CHES publication. For third-order masking, the implementation of Rivain and Prouff takes 470k clock cycles. Compared to that, our implementation with third-order masking and shuffling takes 125k+31k=156k clock cycles if either Scheme 1 is used or Scheme 2 with precomputed powers of k. This shows the experimental evidence that fresh re-keying is also attractive in software. Especially, because the performance scales linearly with the masking order. For a larger number of involved parties though, it would be favorable to have a hardware polynomial multiplier as in [19].

6 Future Research and Conclusions

In this paper we analyzed and extended the fresh re-keying scheme as introduced at Africacrypt 2010. In particular, we extended their research on three lines. First, we analyzed the scheme's susceptibility to algebraic SPA attacks. We showed that on the one hand, if no precautions are taken, such attacks can be easily applied. On the other hand, the assumptions for such attacks are also non-trivial, thus they can be efficiently prevented by shuffling. Second, we extended the scheme to n parties and showed that the security of the two proposed

masking order	single multiplication	Scheme 1	Scheme 2
w/o masking	22,199	48,398	66,597
1^{st}-order	35,559	71,118	106,677
2^{nd}-order	48,919	97,838	146,757
3^{rd}-order	62,279	124,558	186,837

Table 2. Implementation results for the different masking orders with randomly permuted product bytes (RP-256) and random starting indices for the partial products (RSI).

extensions is similar to the one of the original scheme. From a performance point of view the extensions scale linearly in n and our second scheme does not even need additional key material. Third, we implemented the scheme on an 8-bit microcontoller architecture in order to show its efficiency in software. Thus, the paper shows that the fresh re-keying scheme seems to be an appealing choice to provide side-channel security for automotive applications. That is, applications where many low-cost parties need to communicate in a secure way.

Eventually, while the security versus performance results of the fresh re-keying are pretty strong, it is important to note that the analysis of the scheme combines several components. Namely, the overall resistance against side-channel attacks relies on the provable secure masking for the multiplications, the impossibility of biasing the session key, the interleaved recombination of the session key shares and finally, the SPA security of all components. While present evaluation does not reveal obvious weaknesses, a more formal evaluation of the proposed solution, allowing to precisely understand and argue about the interaction between these components, would be a nice scope for further research. Besides, another interesting open problem would be to study the use of a randomness extractor as re-keying function. As recently discussed in [18], such extractors have interesting properties for leakage resilience, when implemented in hardware. Since they are central elements in proofs of leakage resilience such as [6], their use for improving the formal analysis of fresh re-keying schemes could be investigated as well.

Acknowledgements

The work was partly funded by the 7th framework European project TAMPRES, the Belgian Fund for Scientific Research (FNRS-F.R.S.) and the Walloon Region projects S@T Skywin and SCEPTIC.

References

1. The Department of Computer Science at Duke University, Discrete Mathematics for Computer Science lecture, Chapter 18: Probability in hashing, available online at `http://www.cs.duke.edu/courses/cps102/spring09/Lectures/L-18.pdf`, 2009.

2. G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.
3. A. Bogdanov. Improved Side-Channel Collision Attacks on AES. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2007.
4. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. Y. Dodis and K. Pietrzak. Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2010.
6. S. Dziembowski and K. Pietrzak. Leakage-Resilient Cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
7. M. Feldhofer and T. Popp. Power Analysis Resistant AES Implementation for Passive RFID Tags. In C. Lackner, T. Ostermann, M. Sams, and R. Spilka, editors, *Proceedings of Austrochip 2008, October 8, 2008, Linz, Austria*, pages 1–6, October 2008. ISBN 978-3-200-01330-8.
8. L. Genelle, E. Prouff, and M. Quisquater. Montgomery's Trick and Fast Implementation of Masked AES. In A. Nitaj and D. Pointcheval, editors, *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 2011.
9. J. D. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.
10. L. Goubin and J. Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
11. C. Herbst, E. Oswald, and S. Mangard. An AES Smart Card Implementation Resistant to Power Analysis Attacks. In J. Zhou, M. Yung, and F. Bao, editors, *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.
12. R. Karri, K. Wu, P. Mishra, and Y. Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.
13. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy*, pages 447–462. IEEE Computer Society, 2010.
14. H. Ledig, F. Muller, and F. Valette. Enhancing Collision Attacks. In M. Joye and J.-J. Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2004.
15. S. Mangard. A Simple Power-Analysis (SPA) Attackon Implementations of the AES Key Expansion. In P. J. Lee and C. H. Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2002.
16. S. Mangard and F.-X. Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*. Springer, 2010.
17. R. P. McEvoy, M. Tunstall, C. Whelan, C. C. Murphy, and W. P. Marnane. All-or-Nothing Transforms as a Countermeasure to Differential Side-Channel Analysis. Cryptology ePrint Archive, Report 2009/185, 2009. `http://eprint.iacr.org/`.

18. M. Medwed and F.-X. Standaert. Extractors against side-channel attacks: Weak or strong? In *proceedings of CHES 2011 (to appear)*, pages xxx–yyy.
19. 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 D. J. Bernstein and T. Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010.
20. Y. Oren, M. Kirschbaum, T. Popp, and A. Wool. Algebraic Side-Channel Analysis in the Presence of Errors. In Mangard and Standaert [16], pages 428–442.
21. K. Pietrzak. A Leakage-Resilient Mode of Operation. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.
22. T. Plos and M. Feldhofer. Hardware Implementation of a Flexible Tag Platform for Passive RFID Devices. In *Proceedings of the 14th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2011), Oulu, Finland, August, 2010, Proceedings*, pages xxx–xxx. IEEE Computer Society, August 2011.
23. M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.
24. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In Mangard and Standaert [16], pages 413–427.
25. K. Schramm, T. J. Wollinger, and C. Paar. A New Class of Collision Attacks and Its Application to DES. In T. Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.
26. F.-X. Standaert. Leakage Resilient Cryptography: a Practical Overview. invited talk, ECRYPT Workshop on Symmetric Encryption (SKEW 2011), Copenhagen, Denmark, February 2011. `http://perso.uclouvain.be/fstandae/PUBLIS/96_slides.pdf`.
27. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, and E. Oswald. Leakage Resilient Cryptography in Practice. Towards Hardware Intrinsic Security: Foundation and Practice (book chapter), pp 105-139, Springer, 2010.
28. K. Tiri and I. Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In C. D. Walter, Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2003.
29. K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004.
30. J. VanLaven, M. Brehob, and K. J. Compton. Side Channel Analysis, Fault Injection and Applications - A Computationally Feasible SPA Attack on AES via Optimized Search. In R. Sasaki, S. Qing, E. Okamoto, and H. Yoshiura, editors, *SEC*, pages 577–588. Springer, 2005.
31. Y. Yu, F.-X. Standaert, O. Pereira, and M. Yung. Practical leakage-resilient pseudorandom generators. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 141–151. ACM, 2010.