On the Security of Fresh Re-keying to Counteract Side-Channel and Fault Attacks

Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, and Florian Mendel

IAIK, Graz University of Technology, Austria

Abstract. At AFRICACRYPT 2010 and CARDIS 2011, fresh re-keying schemes to counter side-channel and fault attacks were introduced. The idea behind those schemes is to shift the main burden of side-channel protection to a re-keying function g that is easier to protect than the main block cipher. This function produces new session keys based on the secret master key and random nonces for every block of message that is encrypted. In this paper, we present a generic chosen-plaintext key-recovery attack on both fresh re-keying schemes. The attack is based on two observations: Since session key collisions for the same message are easy to detect, it is possible to recover one session key with a simple time-memory trade-off strategy; and if the re-keying function is easy to invert (such as the suggested multiplication constructions), the attacker can use the session key to recover the master key. The attack has a complexity of about $2 \cdot 2^{n/2}$ (instead of the expected 2^n) for an n-bit key. For the typically employed block cipher AES-128, this would result in a key-recovery attack complexity of only 2^{65}. If weaker primitives like 80-bit PRESENT are used, even lower attack complexities are possible.

Keywords: side-channel attacks, fresh re-keying, key-recovery attack

1 Introduction

The design of efficient and effective countermeasures against side-channel and fault attacks is a very challenging task. In fact, more than 15 years ago a kind of an arms race between attackers and designers of countermeasures started and still has not come to an end. In the early years, the main goal of designers of embedded systems was to engineer systems in such a way that they do not leak side-channel information at all [18], or to randomize the power consumption by masking techniques [2]. However, over the years it has become more and more clear that such countermeasures are very expensive to implement for settings with high security requirements. An overview of costs for countermeasures can for example be found in [11].

The main driver for these costs is the fact that in typical settings an attacker can observe the execution of a cryptographic algorithm multiple times with the same key. A good example for such a setting is the mutual authentication of two communicating parties via a challenge-response protocol. In such a setting, the attacker can send an arbitrary number of challenges to a device in order to

obtain an arbitrary number of side-channel measurements or to induce faults to generate pairs of faulty and correct ciphertexts. During each execution of the algorithm, the attacker learns information about the secret key and accumulates this information. This is the basic idea of differential power analysis (DPA) [9] as well as differential fault attacks (DFA) [3].

In [12,13], Medwed et al. propose a re-keying scheme that prevents DPA and DFA attacks by preventing multiple executions of an algorithm with the same key. The basic idea of this re-keying scheme is to never use a long-term key k directly in a cryptographic algorithm, but to derive a fresh session key k^* from k upon each invocation of the algorithm. In fact, for each invocation a random nonce is generated and used in a key derivation function g to generate a session key k^* that is then used by the cryptographic algorithm. This construction prevents an attacker from performing differential attacks on the cryptographic algorithm and this reduces the effort for countermeasures significantly. However, while the cryptographic algorithm needs less protection in this construction, it is clear that it is possible to mount differential attacks on the key derivation function g. Hence, this re-keying approach obviously only pays off in practice if it is significantly easier to protect g against differential attacks than it is to protect the original algorithm.

Medwed et al. provide several arguments for this in [12,13]. In fact, they argue that it is not necessary to have a cryptographic algorithm for the key derivation and propose to use a modular multiplication for the key derivation. A modular multiplication can be protected against differential attacks in a straightforward and efficient way by using blinding techniques [11].

The proposal of Medwed et al. triggered several follow-up works. In fact, several articles [1, 4, 7] treat the question of how to construct a key derivation function that can be implemented efficiently and that at the same time provides a high level of protection against differential attacks. Finding such a function is a central research question in the field of side-channel attacks and countermeasures. This research question is in particular relevant for low-cost systems, such as RFIDs, that typically rely on communication protocols based on symmetric key cryptography. A key derivation function that can be protected efficiently allows to do authentication and session-key derivation based on a symmetric cipher without the need to protect it against differential side-channel and fault attacks. Such a construction can also be used to negotiate a session key that is then used in a leakage-resilient mode of operation [16] for communication.

Our contribution. In this paper, we show that the requirements for the key derivation function that have been formulated in [12, 13] are not sufficient. In fact, we present a simple key-recovery attack on the fresh re-keying schemes proposed in [12, 13]. The basic idea of the attack is that since the scheme changes the block cipher key for every encrypted message block, a time-memory trade-off strategy is possible. An adversary can recover a session key by requesting multiple encryptions of the same message (under different unknown session keys) and searching for collisions with a table of pre-computed encryptions under known

session keys. We also demonstrate how knowledge of one or a few session keys allows to recover the master key for the proposed re-keying functions.

This chosen-plaintext key recovery attack has a complexity as low as $2 \cdot 2^{n/2}$ with similar memory requirements, while the complexity should ideally be 2^n. Due to the properties of the function g (that derives the session key from the master key and random nonces) proposed for these schemes, the master key can be recovered out of one or more recovered session keys. Our attack allows a free trade-off between memory (precomputation) and time/number of queries (online phase), and as such can be tailored to different attack scenarios. In all variants, it is significantly more efficient than Hellman's generic time-memory trade-off.

Outline. The remainder of the paper is organized as follows. We describe the generic construction of the fresh re-keying scheme by Medwed et al. in Section 2. We present our generic key-recovery attack in Section 3 and discuss how the scheme might be fixed in Section 5. Moreover, we give a brief outline to the application of the presented attacks to other fresh re-keying schemes in Section 6. Finally, we conclude in Section 7.

2 Fresh re-keying schemes of Medwed et al.

The basic idea of the re-keying schemes described in this section is to perform every encryption under a new session key k^* to limit the available side-channel information during the encryption for one key. By doing so, the requirements to limit the leakage of side-channel information for the cipher in use can be relaxed. In this section, we describe the two re-keying schemes presented in [13] and [12].

2.1 Basic re-keying scheme (AFRICACRYPT 2010)

The scheme from AFRICACRYPT 2010 [13] targets scenarios where one of the two communication parties only allows limited support for side-channel protection mechanisms. Such a scenario is the communication between an RFID tag and a reader. RFID tags are low cost and low performance devices. Therefore, no overly expensive countermeasures can be included in the RFID tag's block cipher implementation, whereas the protection mechanisms on the more expensive reader can be more complex.

Fig. 1 shows the working principle of the re-keying scheme. This scheme uses two functions: the re-keying function $g(k, r)$ to derive new session keys, and the block cipher $E(k^*, m)$ to encrypt message blocks. For every message block m, a new public nonce r has to be randomly generated on the tag. From this nonce r and the secret master key k, a new session key k^* is then generated via $k^* = g(k, r)$. The session key k^* is then used to encrypt one message block $c = E(k^*, m)$. With the help of the publicly known r and the master key k, the reader is able to decrypt c to $m = E^{-1}(k^*, c)$.

Since the reader cannot contribute to the nonce, an attacker that impersonates the tag can hold the nonce r constant for several different decryptions and

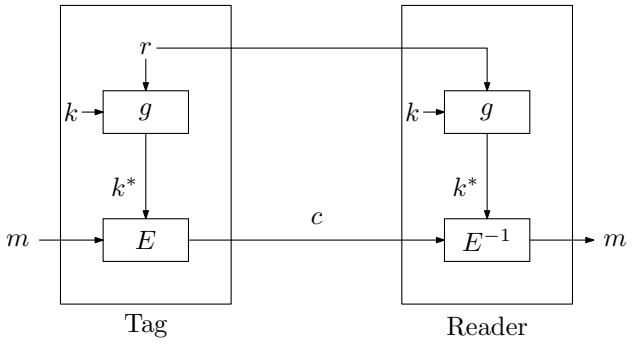

Fig. 1. Structure of the basic re-keying scheme from AFRICACRYPT 2010 [13].

increase the available side-channel information for the reader's implementation. This means that we need different levels of protection for the block cipher implementation E on the reader and on the tag. Medwed et al. argue [13] that both g and E have to be protected against side-channel attacks in the reader's implementation. However, for the tag, only g needs full protection, whereas E does not need to be protected against differential power analysis.

An open question for this re-keying scheme is how to find a suitable function g. In [13], Medwed et al. list six required properties for g:

1. Good diffusion of k.
2. No need for synchronization between parties, i.e., g should be stateless.
3. No additional key material, i.e., k and k^* should be the same size.
4. Little hardware overhead.
5. Easy to protect against side channel attacks.
6. Regularity.

As we show in Section 3, adding another property to the list is necessary:

7. Hard to invert, i.e., it should be hard to recover k from k^*, r.

Medwed et al. [13] propose the following modular multiplication as a specific instance of g:

$$g \,:\, (\mathbb{F}_{2^8}[y]/p(y))^2 \to \mathbb{F}_{2^8}[y]/p(y), \quad (k, r) \mapsto k \cdot r,$$

where $\cdot$ denotes polynomial multiplication in $\mathbb{F}_{2^8}[y]$ modulo $p(y)$. The polynomial $p(y)$ is defined as $p(y) = y^d + 1$ with $d \in \{4, 8, 16\}$ for 128-bit master keys k (typically $d = 16$).

Since $\mathbb{F}_{2^8}[y]/p(y)$ is not a field, but only a ring, and zero divisors exist, $g(k, \cdot)$ it not necessarily bijective for any $k \neq 0$. Master keys k that are zero divisors (not co-prime to $p(y)$) can be considered weak keys since they generate a smaller key space for k^*. Medwed et al. state in [13] that only a fraction of all possible keys k are such weak keys, and that the reduction of the key-space if weak keys are

excluded can be neglected. The same holds true for the nonce r, and randomly generated values for r are unlikely to be 'weak nonces'.

Note that if r is co-prime to $p(y)$, r^{-1} can be calculated easily. Now we can define g', the inverse function to g, easily via $k = g'(k^*, r) = k^* \cdot r^{-1}$. Thus, the master key k can be calculated from a known session key k^* and the corresponding nonce r. We will make use of the function g' in the attack of Section 3.

2.2 Advanced re-keying scheme for multiple parties (CARDIS 2011)

The basic scheme Medwed et al. proposed at AFRICACRYPT 2010 [13] (Section 2.1) only allows low cost side-channel countermeasures for one of the two communication parties. To overcome this drawback, Medwed et al. proposed a second scheme at CARDIS 2011 [12]. This scheme is suitable for multi-party communication (with a common, shared secret key) and allows cheaper side-channel countermeasures for all parties. For clarity, we focus on two-party communication, but the attack can easily be generalized for n parties.

Fig. 2 illustrates the scheme for two-party communication. In contrast to the scheme of Section 2.1, both communication parties are involved in the generation of the session key k^* by contributing a randomly generated nonce.

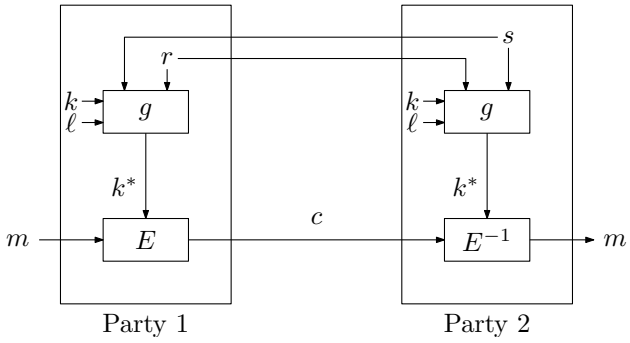

Fig. 2. Structure of the multi-party re-keying scheme from CARDIS 2011 [12] for two parties.

In [12], Medwed at al. propose two different re-keying schemes. The first one uses n different (common, secret) keys for an n-party communication. Each party contributes a random nonce, which is combined with one of the n keys. In the case of a two-party communication, the session key is $k^* = k \cdot r + \ell \cdot s$ (see Fig. 2), where k and ℓ are the secret master keys. r is the public nonce randomly generated by party 1, and s is the public nonce randomly generated by party 2. The ring operations + and $\cdot$ are defined over $\mathbb{F}_{2^8}[y]/p(y)$, as in the AFRICACRYPT paper.

The second proposed scheme [12] uses only one master key k, and expands this to n keys by using powers of k. For the two-party case, the session key is computed as $k^* = r \cdot k + s \cdot k^2$. In general, the n nonces are used as coefficients of a polynomial that is evaluated in k to derive the session key k. In both schemes, the used master keys are restricted to the invertible elements in $\mathbb{F}_{2^8}[y]/p(y)$.

3 Generic key-recovery attack

In this section, we describe simple key-recovery attacks on the encryption scheme of [13] and [12]. In both schemes, the session key k^* is new for every single new encrypted block. In Section 3.1, we show that we are able to recover one of the used session keys k^* with a complexity as low as $2 \cdot 2^{n/2}$ for an n-bit key. Since the function g to derive the session key is easy to invert for both re-keying schemes, we are able to compute the secret master key k out of recovered session keys and the corresponding nonces. We present attacks on the basic re-keying scheme presented at AFRICACRYPT 2010 [13] in Section 3.3 and on the multi-party re-keying scheme from CARDIS 2011 [12] in Section 3.4. Note that similar attacks have been recently published on several authenticated encryption schemes [5,14].

Throughout this section, k is the n-bit master key, k^* an n-bit session key, and r an n-bit nonce.

3.1 Recovery of the session key

As a first step of the attack, we want to recover one of several used session keys. This step consist of two phases: an offline (precomputation) phase and an online (query) phase. The attack is a chosen-plaintext attack with a time complexity of about $2 \cdot 2^{n/2}$. The complexity in memory and number of queries is $2^{n/2}$. Different trade-offs between the memory complexity and the number of queries are possible, at the cost of a higher overall complexity. Chosen-plaintext attacks are not unlikely to be practically applicable if, for instance, protocols based on challenge-response techniques are used.

The basic idea of our attack is to recover the session key k^* from collisions with pre-computed keys. The encryption scheme changes the session key k^* for every block of plaintext that is encrypted. By keeping the plaintext message input to the block cipher fixed, the adversary can apply a basic time-memory trade-off strategy to recover one of the session keys. We will demonstrate in Section 3.3 that this is already enough to also recover the master key k if no additional precautions are taken.

Let $E(k^*, m)$ denote the raw block cipher encryption operation with key k^* and plaintext m. Then the attack strategy is given in Algorithm 1, where m is a fixed message throughout.

A match between an entry in list L and a received ciphertext c gives a candidate session key k^* and the according nonce r. Since there is on average only one possible session key k^* that maps m to c, the possibility of false positives

Algorithm 1 Recover a session key k^*

Fix a message block m.

I. **Offline Phase** (Precomputation)
   Repeat t times:
   1. Guess a new value for k^*.
   2. Compute $c = E(k^*, m)$ and save the pair (c, k^*) in a list L.

II. **Online Phase** (Queries)
   Repeat $t' = 2^n/t$ times:
   1. Request ciphertext c and random nonce r for an encryption of m.
   2. If list L contains an entry (c, k^*) for some k^*, return r and k^*.

is negligible. (If in doubt, the candidate k^* and the derived master key k can be verified with a few additional queries.)

The number of iterations is such that the success probability of finding at least one collision is $\geq 1 - \frac{1}{e} \approx 63.21\,\%$, where e is Euler's number. We assume that no key candidate k^* in the offline phase is selected twice (drawing without replacement), but duplicates may occur in the online phase. Then, the probability of failure is $\left(1 - \frac{t}{2^n}\right)^{t'} = (1 - t')^{t'}$, which increases monotonically from 0 up to $\frac{1}{e}$ as the number t' of online queries grows (while t decreases accordingly). Since the expected number of false alarms is small, we can state that the algorithm finds a correct used session key k^* with high probability with a total complexity of t offline encryptions plus $2^n/t$ online chosen-plaintext queries. The best overall complexity of $2 \cdot 2^{n/2}$ is achieved for $t = 2^{n/2}$.

Sometimes, an attacker wants to recover more than only a single master key. In this case, only the second phase of the attack has to be repeated, while the precomputation phase has to be done only once. In such settings, in particular if the number of attacked keys is large, other values of t might result in a better overall complexity. In Table 1, we give the complexities and memory requirements for different choices of t.

Table 1. Complexities and memory requirements for both phases of the attack with different choices of t.

$\log_2(t)$	offline phase	online phase	memory	total
$n/4$	$2^{n/4}$	$2^{3n/4}$	$2^{n/4}$	$2^{3n/4}$
$n/3$	$2^{n/3}$	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$
$n/2$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2 \cdot 2^{n/2}$
$2n/3$	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{2n/3}$
$3n/4$	$2^{3n/4}$	$2^{n/4}$	$2^{3n/4}$	$2^{3n/4}$

3.2 Memoryless session key recovery

In practice, the memory requirements are typically the most significant restriction for this attack. Unfortunately, since the values of the online phase are not under the attacker's control, standard memoryless collision search techniques are not directly applicable. If the attacker could additionally choose the nonce r for online queries, memoryless cycle finding algorithms would reduce the memory requirements to constant or logarithmic while only marginally (by a small constant factor) increasing the necessary number of online queries.

There are two possible modifications to the attack that allow this. Both attack the reader instead of the tag in the basic scheme of Section 2.1. The first assumes that the reader can also send messages to the tag by requesting a new nonce from the tag and then encrypting under this nonce. This would require the tag to remember the nonce until it receives the corresponding encrypted message. Then, the attacker can send chosen nonces r together with the fixed message m for encryption, and apply the memoryless algorithm described below.

The other variant does not make any such assumptions, but simply attacks decryption instead of encryption in a chosen-ciphertext setting. Instead of a fixed plaintext m, a fixed ciphertext c is sent to the reader together with a chosen nonce r. The collision target, then, is the received plaintext m.

Either of these two versions can be used for memoryless session key recovery as follows. We construct a helper function $f : \{0,1\}^n \to \{0,1\}^n$:

$$f(x) = \begin{cases} D(x, c) & \text{if the last bit of } x \text{ is } 0 \quad \text{(offline, session key guess } x\text{)}, \\ D(g(k, x), c) & \text{if the last bit of } x \text{ is } 1 \quad \text{(query with nonce } x\text{)}, \end{cases}$$

where $D(k^*, c)$ denotes decryption of a fixed ciphertext c. A collision $f(x_1) = f(x_2)$ for f will give us a session key $k^* = x_1$ and corresponding nonce $r = x_2$ with a probability of $\frac{1}{2}$ (otherwise, we have to repeat the procedure).

Now, we can consider the sequence generated by $x_i = f(x_{i-1})$ and apply a standard cycle finding algorithm to determine the periodicity of this sequence and derive a collision. For example, using Brent's algorithm [6], the expected number of evaluations of f to find a collision (for a random mapping) is

$$\sqrt{\frac{\pi}{8}} \cdot \left(\frac{3}{\log 4} + 2 \right) \cdot 2^{n/2} \approx 2.6094 \cdot 2^{n/2}.$$

Since the expected necessary number of collisions to recover a session key is 2, the overall complexity of this approach is slightly higher than before, but the memory requirements are negligible.

For minimizing the overall complexity for a fixed given memory size, better trade-offs are achieved by distinguished-point searches and similar methods. Examples include Quisquater and Delescaille's [17] or van Oorschot and Wiener's [15] algorithms. In particular, the latter is useful if multiple collisions are required.

3.3 Master key recovery for the basic AFRICACRYPT 2010 scheme

For the attack on the basic re-keying scheme of AFRICACRYPT 2010 [13], we can directly apply the standard or memoryless collision searches from Section 3.1. Assume we successfully recovered one session key k^* and the corresponding nonce r. The re-keying function used in this scheme is $k^* = g(k, r) = k \cdot r$.

As already discussed in Section 2.1, the majority of the nonces r is coprime to y^d+1 and the inverse r^{-1} exists. Therefore, we can define the inverse function $k = g'(k^*, r) = k^* \cdot r^{-1}$ and simply derive the master key k in use. The overall complexity of this attack is dominated by the session-key recovery complexity of $2 \cdot 2^{n/2}$.

3.4 Master key recovery for the CARDIS 2011 multi-party scheme

Two different functions g for re-keying are proposed in [12]. We first consider the version with $k^* = r \cdot k + s \cdot \ell$ in the two-party case. Recall that k and ℓ are the two master keys, and r and s are nonces chosen freshly by the two communicating parties. We attack the device during the online phase, where the attacker has control over the nonce s. For simplicity, the nonce s is kept constant during the whole online phase, although the attack works just as well for random nonces. Now, we need to recover two session keys k_1^* and k_2^* with two corresponding nonces r_1 and r_2 to determine the master key. We can then set up the following equations:

$$k_1^* = r_1 \cdot k + s \cdot \ell,$$
$$k_2^* = r_2 \cdot k + s \cdot \ell.$$

By combining them, we get

$$k_1^* - k_2^* = (r_1 - r_2) \cdot k.$$

If the inverse of $(r_1 - r_2)$ exists (which holds with overwhelming probability), we can calculate the first master key k as

$$k = (r_1 - r_2)^{-1} \cdot (k_1^* - k_2^*)$$

As s is also invertible (trivially if we control s, with high probability otherwise), we also get ℓ:

$$\ell = s^{-1} \cdot (k_1^* - r_1 \cdot k).$$

As nearly every difference of $(r_1 - r_2)$ is coprime to $y^{16}+1$, the complexity of this attack is determined by finding the two necessary session keys. By increasing the precomputed table size and the number of online queries to $t = t' = \sqrt{2} \cdot 2^{n/2}$, we can achieve this with an overall complexity of $2\sqrt{2} \cdot 2^{n/2} \approx 2.8284 \cdot 2^{n/2}$ encryptions and a success probability of about $1 - \frac{3}{e^2} \approx 59.40\,\%$. Note that the memoryless version cannot realistically be used in this scenario, since we are unlikely to be able to control both parties' nonces.

Clearly, the same attack applies if k^2 is used instead of ℓ. For m parties, we need m session keys to recover the m unknown master keys. The necessary table size and number of queries grow accordingly.

4 Comparison to Hellman's time-memory trade-off attack

Hellman [8] described a generic cryptanalytic time-memory trade-off attack on block ciphers. For a block cipher with a key size of n bits, after a precomputation with time complexity of about 2^n, Hellman's method has an (online) time complexity of $T = 2^{2n/3}$ and memory requirements of $M = 2^{2n/3}$ to recover the key. In more detail, it allows a time/memory trade-off curve of $M \cdot \sqrt{T} = 2^n$. Since we are only interested in attacks with $T \leq 2^n$ (faster than brute force), M has to be at least $2^{n/2}$. We want to note that the attack described in this paper is on a much better time/memory trade-off curve, $M \cdot T = 2^n$, and in particular does not require a 2^n precomputation.

5 Fixing the scheme

The main problem of the construction is that the function g is easy to invert. This allows to extend the time-memory trade-off attacks for session key recovery to full master key recovery as demonstrated in Section 3. A simple, but unsatisfactory solution to prevent this kind of attack is to increase the master key, session key, and nonce sizes to twice the security level each. For example, if the desired security level is 128 bits, AES-256 is a natural choice for the block cipher E, with a performance overhead of about 40 % compared to AES-128. Additionally, the nonce transmission overhead becomes twice as large. This is clearly not compatible with resource-constrained application scenarios.

The alternative is to fix the construction by using a function g that is hard to invert, as for instance the one suggested in [4]. It should be hard to recover the master key k from the knowledge of one or a few session keys k^* and corresponding nonces r. However, this raises the question how such a cryptographically strong function can be constructed without in turn being very costly to protect against side-channel attacks. It is not sufficient to simply postprocess k^* with some preimage-resistant function that does not additionally depend on any secret information (i.e., parts of the key). Clearly, additional research is necessary to identify suitable constructions and desirable properties for g.

6 Application to other fresh re-keying schemes

The attacked schemes [12,13] are nonce-based and stateless. In short, this means that the communicating parties share a secret key and derive the session key by using the exchanged nonces. Besides this type of schemes, other schemes have been proposed, such as the re-keying scheme by Kocher [10]. This scheme works without nonces. To generate the session keys, the communicating parties traverse a tree-like structure. We call schemes like Kocher's [10] stateful schemes.

It is easy to see that similar time-memory trade-off attacks are also possible on stateful schemes. To mount such attacks and recover the master key, the used functions to generate the session keys have to be publicly known and must be easy to invert.

7 Conclusions

In this paper, we have analyzed fresh re-keying schemes from a generic point of view. We demonstrated how to recover one (of many) used session keys with a complexity of about $2 \cdot 2^{n/2}$ chosen-plaintext queries. Depending on the function g used for deriving the session key, knowledge of one or a few session keys may allow to even recover the master key. In case of the simple and multi-party re-keying schemes suggested by Medwed et al. [12,13], recovering the master key is easily possible since their function g is easy to invert. The effect of our attacks is that the complexity to recover the master key is reduced from the ideal 2^n to about $2 \cdot 2^{n/2}$.

A simple, but unsatisfactory solution to prevent this kind of attacks is to increase the master key, session key and nonce sizes to twice the security level each. More promising approaches focus on the properties of g, in particular the hardness to invert $g(\cdot, r)$ to deduce the key k, as in the scheme by Belaid et al. [4]. Our results show that designing secure, efficient re-keying functions remains a challenging task, and that frequent re-keying opens up problems of its own that are not yet fully understood.

Acknowledgments. This work has been supported in part by the Austrian Science Fund (project P26494-N15) and by the Austrian Government through the research program ICT of the Future under the project number 4593209 (project SCALAS).

References

1. Abdalla, M., Belaïd, S., Fouque, P.A.: Leakage-Resilient Symmetric Encryption via Re-keying. In: Bertoni, G., Coron, J.S. (eds.) CHES. LNCS, vol. 8086, pp. 471–488. Springer (2013)
2. Akkar, M.L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Çetin Kaya Koç, Naccache, D., Paar, C. (eds.) CHES. LNCS, vol. 2162, pp. 309–318. Springer (2001)
3. Ali, S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES: towards reaching its limits. J. Cryptographic Engineering 3(2), 73–97 (2013)
4. Belaïd, S., Santis, F.D., Heyszl, J., Mangard, S., Medwed, M., Schmidt, J., Standaert, F., Tillich, S.: Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis. J. Cryptographic Engineering 4(3), 157–171 (2014)
5. Bogdanov, A., Dobraunig, C., Eichlseder, M., Lauridsen, M., Mendel, F., Schläffer, M., Tischhauser, E.: Key Recovery Attacks on Recent Authenticated Ciphers. In: Aranha, D., Menezes, A. (eds.) LATINCRYPT. LNCS, Springer (2014), to appear
6. Brent, R.P.: An improved Monte Carlo factorization algorithm. BIT, Nord. Tidskr. Inf.-behandl. 20 pp. 176–184 (1980)
7. Grosso, V., Poussier, R., Standaert, F.X., Gaspar, L.: Combining Leakage-Resilient PRFs and Shuffling (Towards Bounded Security for Small Embedded Devices). IACR Cryptology ePrint Archive 2014, 411 (2014)
8. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26(4), 401–406 (1980)

9. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) CRYPTO. LNCS, vol. 1666, pp. 388–397. Springer (1999)
10. Kocher, P.: Leak-resistant cryptographic indexed key update (Mar 25 2003), `http://www.google.com/patents/US6539092`, US Patent 6,539,092
11. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
12. Medwed, M., Petit, C., Regazzoni, F., Renauld, M., Standaert, F.X.: Fresh Rekeying II: Securing Multiple Parties against Side-Channel and Fault Attacks. In: Prouff, E. (ed.) CARDIS. LNCS, vol. 7079, pp. 115–132. Springer (2011)
13. Medwed, M., Standaert, F.X., Großschädl, J., Regazzoni, F.: Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT. LNCS, vol. 6055, pp. 279–296. Springer (2010)
14. Mendel, F., Mennink, B., Rijmen, V., Tischhauser, E.: A Simple Key-Recovery Attack on McOE-X. In: Pieprzyk, J., Sadeghi, A.R., Manulis, M. (eds.) CANS. LNCS, vol. 7712, pp. 23–31. Springer (2012)
15. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Application to Hash Functions and Discrete Logarithms. In: ACM Conference on Computer and Communications Security. pp. 210–218 (1994)
16. Pietrzak, K.: A Leakage-Resilient Mode of Operation. In: Joux, A. (ed.) EUROCRYPT. LNCS, vol. 5479, pp. 462–482. Springer (2009)
17. Quisquater, J.J., Delescaille, J.P.: How Easy is Collision Search. New Results and Applications to DES. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 408–413. Springer (1989)
18. Tiri, K., Verbauwhede, I.: Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In: Walter, C.D., Çetin Kaya Koç, Paar, C. (eds.) CHES. LNCS, vol. 2779, pp. 125–136. Springer (2003)