From Related-Key Distinguishers to Related-Key-Recovery on Even-Mansour Constructions*

Pierre Karpman[1,2]**

[1] Inria, France
[2] Nanyang Technological University, Singapore

pierre.karpman@inria.fr

Abstract. We show that a distinguishing attack in the related key model on an Even-Mansour block cipher can readily be converted into an extremely efficient key recovery attack. Concerned ciphers include in particular all iterated Even-Mansour schemes with independent keys. We apply this observation to the CAESAR candidate PRØST-OTR and are able to recover the whole key with a number of requests linear in its size. This improves on recent forgery attacks in a similar setting.

Keywords: Even-Mansour, related-key attacks, PRØST-OTR.

1 Introduction

The Even-Mansour scheme is arguably the simplest way to construct a block cipher from publicly available components. It defines the encryption $\mathcal{E}((k_1, k_0), p)$ of the plaintext p under the (possibly equal) keys k_0 and k_1 as $\mathcal{P}(p \oplus k_0) \oplus k_1$, where $\mathcal{P}$ is a public permutation. Even and Mansour proved in 1991 that for a permutation of size n, the probability of recovering the keys is upper-bounded by $\mathcal{O}(DT \cdot 2^{-n})$ when the attacker considers the permutation as a black box, where D is the data complexity and T is the time complexity of the attack [EM91]. Although of considerable interest, this bound also shows at the same time that the construction is not ideal, as one gets security only up to $\mathcal{O}(2^{\frac{n}{2}})$ queries, which is less than the $\mathcal{O}(2^n)$ one would expect for an n-bit block cipher. For this reason, much later work investigated the security of variants of the Even-Mansour cipher. A simple one is the iterated Even-Mansour scheme with independent keys and independent permutations, with its r-round version defined as $\mathsf{IEM}^r((k_r, k_{r-1}, \ldots, k_0), p) := \mathcal{P}_{r-1}(\mathcal{P}_{r-2}(\ldots \mathcal{P}_0(p \oplus k_0) \oplus k_1) \ldots) \oplus k_r$. It has been established that this construction is secure up to $\mathcal{O}(2^{\frac{rn}{r+1}})$ queries [CS14]. On the other hand, in a related-key model, the same construction lends itself to trivial distinguishing attacks, and one must consider alternatives if security in this model is necessary. Yet until the recent work of Cogliati and Seurin [CS15] and Farshim and Procter [FP14], no variant of the Even-Mansour construction was known to be secure in the related-key model. This is not the case anymore and it has now been proven that one can reach a non-trivial level of related-key security for IEM^r starting from $r = 3$ when

* This paper appears in the proceedings of ISC 2015 under the title *From Related-Key Distinguishers to Related-Key Key-Recovery on Even-Mansour Ciphers*.

** Partially supported by the Direction Générale de l'Armement and by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

using keys linearly derived from a single master key (instead of using independent keys), or even when $r = 1$ when this derivation is non-linear and meets some conditions. While related-key analysis obviously gives much more power to the attacker than the single-key setting, it is a widely accepted model that may provide useful results on primitives studied in a general context, especially as related keys may naturally arise in some protocols.

Our contribution. We show that the distinguishing attacks on the Even-Mansour ciphers in a related-key model can be extended to much more powerful key-recovery attacks by considering modular additive differences instead of XOR differences. This applies both to the trivial distinguishers on iterated Even-Mansour with independent keys and to the more complex distinguisher of Cogliati and Seurin for 2-round Even-Mansour with a linear key-schedule. While these observations are somewhat elementary, they lead to a key-recovery attack on the authenticated-encryption scheme and CAESAR candidate PRØST-OTR in a related-key model. This improves on the recent work from FSE 2015 of Dobraunig, Eichlseder and Mendel who use similar methods but only produce forgeries [DEM15].

2 Notation

We use $||$ to denote string concatenation, α^i with i an integer to denote the string made of the concatenation of i copies of the character α, and α^* to denote any string of the set $\{\alpha^i, i \in \mathbf{N}\}$, α^0 denoting the empty string ε. For any string s, we use $s[i]$ to denote its i^{th} element (starting from zero).

We also use Δ_i^n to denote the string $0^{n-i-1}||1||0^{i-1}$. The superscript n will always be clear from the context and therefore omitted.

Finally, we identify strings of length n over the binary alphabet $\{0, 1\}$ with elements of the vector-space $\mathbf{F}_2^n$ and with the binary representation of elements of the group $\mathbf{Z}/2^n\mathbf{Z}$. The addition operation on these structures are respectively denoted by $\oplus$ (bitwise exclusive or (XOR)) and $+$ (modular addition).

3 Generic related-key key-recovery attacks on Even-Mansour ciphers

Since the work of Bellare and Kohno [BK03], it is well known that no block cipher can resist related-key attacks (RKA) when an attacker may request encryptions under related keys using two relation classes. A simple example showing why this cannot be the case is to consider the classes $\phi^{\oplus}(k)$ and $\phi^{+}(k)$ of keys related to k by the XOR and the modular addition of any constant chosen by the attacker respectively. If we have access to (related-key) encryption oracles $\mathcal{E}(k, \cdot)$, $\mathcal{E}(\phi^{\oplus}(k), \cdot)$ and $\mathcal{E}(\phi^{+}(k), \cdot)$ for the block cipher $\mathcal{E}$ with κ-bit keys, we can easily learn the value of the bit $k[i]$ of k by comparing the result of the queries $\mathcal{E}(k + \Delta_i, p)$ and $\mathcal{E}(k \oplus \Delta_i, p)$. For $i < \kappa - 1$, the plaintext p is encrypted under the same key if $k[i] = 0$, then resulting in the same ciphertext, and is encrypted under different keys if $k[i] = 1$, then resulting in different ciphertexts with an overwhelming probability. Doing this test for every bit of k thus allows to recover the whole key with a complexity linear in κ, except its most significant bit. Indeed, the carry of an addition on this bit never

propagates and thus there will never be a difference between the related keys. This key bit can of course easily be recovered at the cost of one additional query once all the others have been determined.

In the same paper, Bellare and Kohno also show that no such trivial generic attack exists when the attacker is restricted to using only one of the two classes $\phi^{\oplus}$ or ϕ^{+}, and they prove that an ideal cipher is in this case resistant to RKA. Taken together, these results mean in essence that a related-key attack on a block cipher $\mathcal{E}$ using both classes $\phi^{\oplus}(k)$ and $\phi^{+}(k)$ does not say much on $\mathcal{E}$, as nearly all ciphers fall to an attack in the same model. On the other hand, an attack using either of $\phi^{\oplus}$ or ϕ^{+} *is* meaningful, because an ideal cipher is secure in that case.

3.1 Key-recovery attacks on r-round IEM with independent keys

Going back to the Even-Mansour cipher, we explicit the trivial related-key distinguishers mentioned in the introduction. These distinguishers exist for r-round iterated Even-Mansour block ciphers with independent keys, for any value of r. As they only use keys related with, say, the $\phi^{\oplus}$ class, they are therefore meaningful when considering the related-key security of IEM.

From the very definition of IEM^r, it is obvious to see that the two values $\mathcal{E}((k_{r-1}, k_{r-2}, \ldots, k_0), p)$ and $\mathcal{E}((k_{r-1}, k_{r-2}, \ldots, k_0 \oplus \delta), p \oplus \delta)$ are equal for any difference δ when $\mathcal{E} = \mathsf{IEM}^r$ and that this equality does not hold in general, thence allowing to distinguish IEM^r from an ideal cipher.

We now show how these distinguishers can be combined with the two-class attack of Bellare and Kohno in order to extend it to a very efficient key-recovery attack. We give a description in the case of one-round Even-Mansour, but it can easily be extended to an arbitrary r. The attack is very simple and works as follows: consider again $\mathcal{E}((k_1, k_0), p) = \mathcal{P}(p \oplus k_0) \oplus k_1$; one can learn the value of the bit $k_0[i]$ by querying $\mathcal{E}((k_1, k_0), p)$ and $\mathcal{E}((k_1, k_0 + \Delta_i), p \oplus \Delta_i)$ and by comparing their values. These differ with overwhelming probability if $k_0[i] = 1$ and are equal otherwise.

A similar attack works on the variant of the (iterated) Even-Mansour cipher that uses modular addition instead of XOR for the combination of the key with the plaintext. This variant was first analyzed by Dunkelman, Keller and Shamir and offers the same security bounds as the original Even-Mansour cipher [DKS12]. An attack in that case works similarly by querying *e.g.* $\mathcal{E}((k_1, k_0), \Delta_i)$ and $\mathcal{E}((k_1, k_0 \oplus \Delta_i), 0^\kappa)$.

Both attacks use a single difference class for the related keys (either $\phi^{\oplus}$ or ϕ^{+}), and they are therefore meaningful as related-key attacks. They simply emulate the attack that uses both classes simultaneously by taking advantage of the fact that the usage of key material is very simple in Even-Mansour ciphers. Finally, we can see that in the particular case of a one-round construction, the attack still works if one chooses the keys k_1 and k_0 to be equal.

3.2 Extension to 2-round Even-Mansour with a linear key schedule

As has been shown by Cogliati and Seurin, it is also possible to very efficiently distinguish the 2-round Even-Mansour with related keys, even when the keys are equal or derived from a master key by a linear key schedule [CS15]. Similarly as for IEM, we can adapt the distinguisher and transform it into a key-recovery attack. The idea remains the same: one replaces the $\phi^{\oplus}$ class of the original distinguisher with ϕ^{+}, which makes its success conditioned on the value of a few key bits, hence allowing their recovery. We give the description of our modified distinguisher for $\mathcal{E}(k,p) := \mathcal{P}(\mathcal{P}(k \oplus p) \oplus k) \oplus k$:

1. Query $y_1 := \mathcal{E}(k + \Delta_1, x_1)$
2. Set x_2 to $x_1 \oplus \Delta_1 \oplus \Delta_2$ and query $y_2 := \mathcal{E}(k + \Delta_2, x_2)$
3. Set y_3 to $y_1 \oplus \Delta_1 \oplus \Delta_3$ and query $x_3 := \mathcal{E}^{-1}(k + \Delta_3, y_3)$
4. Set y_4 to $y_2 \oplus \Delta_1 \oplus \Delta_3$ and query $x_4 := \mathcal{E}^{-1}(k + (\Delta_1 \oplus \Delta_2 \oplus \Delta_3), y_4)$
5. Test if $x_4 = x_3 \oplus \Delta_1 \oplus \Delta_2$

If the test is successful, it means that with overwhelming probability the key bits at the positions of the differences Δ_1, Δ_2, Δ_3 are all zero, as in that case $k + \Delta_i = k \oplus \Delta_i$ and the distinguisher works "as intended", and as otherwise at least one uncontrolled difference goes through $\mathcal{P}$ or $\mathcal{P}^{-1}$. It is possible to restrict oneself to using differences in only two bits in the Δ_is, and as soon as two such zero bits have been found (which happens after an expected four trials for random keys), the rest of the key bits can be tested one by one.

We conclude this short section by showing why the test of line 5 is successful when $k + \Delta_i = k \oplus \Delta_i$, but refer to Cogliati and Seurin for a complete description of their distinguisher, including the general case of distinct permutations and keys linearly derived from a master key (this only requires slight modifications to our simplified formulation).

For the sake of clarity, we write $k \oplus \Delta_i$ for $k + \Delta_i$, as they are equal by hypothesis. By definition, $y_1 = \mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1$ and $y_3 = \mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1 \oplus \Delta_1 \oplus \Delta_3$ which simplifies to $\mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_3$. This yields the following expression for x_3:

$$\begin{aligned}
x_3 &= \mathcal{P}^{-1}(\mathcal{P}^{-1}(\mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_3 \oplus k \oplus \Delta_3) \oplus k \oplus \Delta_3) \oplus k \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}^{-1}(\mathcal{P}(\,\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1)) \oplus k \oplus \Delta_3) \oplus k \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_1 \oplus k \oplus \Delta_3) \oplus k \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus \Delta_1 \oplus \Delta_3) \oplus k \oplus \Delta_3
\end{aligned}$$

Similarly, $y_2 = \mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_2) \oplus k \oplus \Delta_2$ and $y_4 = \mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_2) \oplus k \oplus \Delta_2 \oplus \Delta_1 \oplus \Delta_3$, which yields the following expression for x_4:

$$\begin{aligned}
x_4 &= \mathcal{P}^{-1}(\mathcal{P}^{-1}(\mathcal{P}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_2) \oplus k \oplus \Delta_2 \oplus \Delta_1 \oplus \Delta_3 \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3) \\
&\quad \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3) \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}^{-1}(\mathcal{P}(\,\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_2)) \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3) \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus k \oplus \Delta_2 \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3) \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3 \\
&= \mathcal{P}^{-1}(\mathcal{P}(x_1 \oplus k \oplus \Delta_1) \oplus \Delta_1 \oplus \Delta_3) \oplus k \oplus \Delta_1 \oplus \Delta_2 \oplus \Delta_3
\end{aligned}$$

From the final expressions of x_3 and x_4, we see that their XOR difference is indeed $\Delta_1 \oplus \Delta_2$.

4 Application to PRØST-OTR

We apply the simple generic key-recovery attack to the CAESAR candidate PRØST-OTR, which is an authenticated-encryption scheme member of the PRØST family [KLL+14]. This family is based on the PRØST permutation and defines three schemes instantiating as many modes of operation, namely COPA, OTR and APE. Only the latter can be readily instantiated with a permutation, and both COPA and OTR rely on a keyed primitive. For that purpose they use a block cipher defined as a one-round Even-Mansour cipher with identical keys $\mathcal{E}(k, p) := \mathcal{P}(p \oplus k) \oplus k$ with the PRØST permutation as $\mathcal{P}$. We will denote this cipher as PRØST/SEM.

Although the attack of Section 3 could be readily applied to the bare PRØST/SEM, this cipher is only meant to be embedded into a specific instantiation of a mode such as OTR, and attacking it out of context may not be relevant to its intended use. Hence we must be able to mount an attack on PRØST-COPA or PRØST-OTR as a whole for it to be really significant, which is precisely what we describe now for PRØST-OTR.

Because our attack solely relies on the Even-Mansour structure of the cipher, we refer the interested reader to the submission document of PRØST for the definition of its permutation. The same goes for the OTR mode [Min14], as we only need to focus on a small part to describe the attack. Consequently, we just describe how the encryption of the first block of plaintext is performed in PRØST-OTR.

OTR is a nonce-based mode of operation. It takes as input a key k, a message m, (possibly empty) associated data a, and produces a ciphertext c corresponding to the encryption of the message with k, and a tag t authenticating m and a together with the key k. It is important for the security of the mode to ensure that one cannot encrypt twice using the same nonce. However, there are no specific restriction as to their value, and we consider throughout that one can freely choose them.

The encryption of the first block of ciphertext c_1 by PRØST-OTR is defined as a function $\mathcal{F}(k, n, m_1, m_2)$ of k, n, and the first two blocks of plaintext m_1 and m_2: let $\ell := \mathcal{E}(k, n||10^*)$ be the encryption of the padded nonce and $\ell' := \pi(\ell)$, with π a linear permutation (the multiplication by 4 in a finite field), then c_1 is simply equal to $\mathcal{E}(k, \ell' \oplus m_1) \oplus m_2$. We show this schematically along with the encryption of the second block in Figure 4.1. Let us now apply the attack from Section 3.

Step 1: Recovering the most significant half of the key. It is straightforward to see that one can recover the value of the bit $k[i]$ by performing only two queries with related keys and different nonces and messages. One just has to compare $c_1 = \mathcal{F}(k, n, m_1, m_2)$ and $\hat{c}_1 = \mathcal{F}(k + \Delta_i, n \oplus \Delta_i, m_1 \oplus \Delta_i \oplus \pi(\Delta_i), m_2)$. Indeed, if $k[i] = 0$, then the value $\hat{\ell}$ obtained in the computation of $\hat{c}_1$ is equal to $\ell \oplus \Delta_i$ and $\hat{\ell}' = \ell' \oplus \pi(\Delta_i)$, hence $\hat{c}_1 = c_1 \oplus \Delta_i$. If $k[i] = 1$, the latter equality does not hold with overwhelming probability.

Yet this does not allow to recover the whole key because the nonce in PRØST-OTR is restricted to a length half of the width of the block cipher $\mathcal{E}$ (or equivalently of the underlying PRØST permutation), *i.e.* $\frac{\kappa}{2}$. It is then possible to recover only half of the bits of k using this procedure, as one cannot introduce appropriate differences in the computation

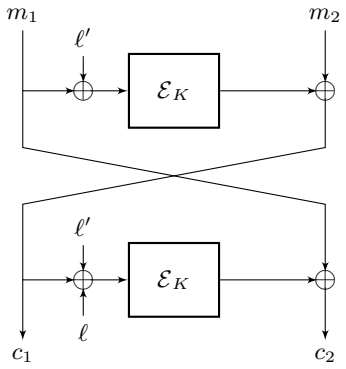

Fig. 4.1. The encryption of the first two blocks of message in PRØST-OTR.

of ℓ for the other half. The targeted security of the whole primitive being precisely $\frac{\kappa}{2}$ because of the generic single key attacks on Even-Mansour, one does not make a significant gain by recovering only half of the key. Even though, it should still be noted that this yields an attack with very little data requirements and with the same time complexity as the best point on the tradeoff curve of generic attacks, which in that case has a much higher data complexity of $2^{\frac{\kappa}{2}}$.

Step 2: Recovering the least significant half of the key. Even though the generic attack in its most simple form does not allow to recover the full key of PRØST-OTR, we can use the fact that the padding of the nonce is done on the least significant bits to our advantage, and by slightly adapting the procedure we can iteratively recover the value of the least significant half of the key with no more effort than for the most significant half.

Let us first show how we can recover the most significant bit of the least significant half of the key $k[\kappa/2-1]$ (*i.e.* the first bit for which we cannot use the procedure of Step 1) after a single encryption by $\mathcal{E}$.

We note k^{MSB} the (known) most significant half of the key k. To mount the attack, one queries $\mathcal{E}(k - k^{\mathsf{MSB}} + \Delta_{\kappa/2-1}, p \oplus \Delta_{\kappa/2})$ and $\mathcal{E}(k - k^{\mathsf{MSB}} - \Delta_{\kappa/2-1}, p)$. We can see that the inputs to $\mathcal{P}$ in these two cases are equal iff $k[\kappa/2-1] = 1$. Indeed, in that case, the carry in the addition $(k - k^{\mathsf{MSB}}) + \Delta_{\kappa/2-1}$ propagates by exactly one position and is "cancelled" by the difference in p, and there is no carry propagation in $(k - k^{\mathsf{MSB}}) - \Delta_{\kappa/2-1}$. The result of the two queries are therefore equal to $\mathcal{C} \oplus (k - k^{\mathsf{MSB}} + \Delta_{\kappa/2-1}) = \mathcal{C} \oplus (k \oplus k^{\mathsf{MSB}} \oplus \Delta_{\kappa/2-1} \oplus \Delta_{\kappa/2})$ and $\mathcal{C} \oplus (k - k^{\mathsf{MSB}} - \Delta_{\kappa/2-1}) = \mathcal{C} \oplus (k \oplus k^{\mathsf{MSB}} \oplus \Delta_{\kappa/2-1})$ with $\mathcal{C} = \mathcal{P}(p \oplus (k - k^{\mathsf{MSB}} - \Delta_{\kappa/2-1}))$. Consequently, the XOR difference between the two results is known and equal to $\Delta_{\kappa/2}$. If on the other hand $k[\kappa/2-1] = 0$, the carry in $(k - k^{\mathsf{MSB}}) - \Delta_{\kappa/2-1})$ propagates all the way to the most significant bit of k, whereas only two differences are introduced in the input to $\mathcal{P}$ in the first query. This allows to distinguish between the two cases and thus to recover the value of this key bit.

Once the value of $k[\kappa/2-1]$ has been learned, one can iterate the process to recover the remaining bits of k. The only subtlety is that we want to ensure that if there is a carry propagation in $(k-k^{\mathsf{MSB}})+\Delta_{\kappa/2-1-i}$ (resp. $(k-k^{\mathsf{MSB}})-\Delta_{\kappa/2-1-i}$), it should propagate up to $k_{\kappa/2}$, the position where we cancel it with an XOR difference (resp. up to the most significant bit); this can easily be achieved by adding two terms to both keys. Let us define γ_i as the value of the key k only on positions $\kappa/2-1\ldots\kappa/2-i$, completed with zeros left and right; that is $\gamma_i[j]=k[j]$ if $\kappa/2-1\geq j\geq\kappa/2-i$, and $\gamma_i[j]=0$ otherwise. Let us also define $\widetilde{\gamma_i}$ as the binary complement of γ_i on its non-zero support; that is $\widetilde{\gamma_i}[j]=\widetilde{k[j]}$ if $\kappa/2-1\geq j\geq\kappa/2-i$, and $\widetilde{\gamma_i}[j]=0$ otherwise. The modified queries then become $\mathcal{E}(k-k^{\mathsf{MSB}}+\Delta_{\kappa/2-1-i}+\widetilde{\gamma_i},p\oplus\Delta_{\kappa/2})$ and $\mathcal{E}(k-k^{\mathsf{MSB}}-\Delta_{\kappa/2-1-i}-\gamma_i,p)$, for which the propagation of the carries is ensured. Note that the difference between the results of these two queries when $k[\kappa/2-1-i]=1$ is independent of i and always equal to $\Delta_{\kappa/2}$.

We conclude by showing how to apply this procedure to Prøst-OTR. For the sake of readability, let us denote by Δ_i^+ and Δ_i^- the complete modular differences used to recover one less significant bit $k[i]$. We then simply perform the two queries $\mathcal{F}(k+\Delta_i^+,n\oplus\Delta_{\kappa/2},m_1\oplus\Delta_{\kappa/2},m_2)$ and $\mathcal{F}(k+\Delta_i^-,n,m_1\oplus\pi(\Delta_{\kappa/2}),m_2)$, which differ by $\Delta_{\kappa/2}$ iff k_i is one, with overwhelming probability.

All in all, one can retrieve the whole key of size κ using only 2κ related-key, related-nonce encryption requests, ignoring everything in the output (including the tag) apart from the value of the first block of ciphertext. We give the entire procedure to do so in Algorithm 4.1. Note that it makes use of a procedure Refresh which picks fresh values for two message words and (most importantly) for the nonce. Because the attack is entirely practical, it can easily be tested. We give an implementation of the attack for a 64-bit toy cipher in the Appendix A[1].

Remark. If the padding of the nonce in Prøst-OTR were done on the most significant bits, no attack similar to Step 2 could recover the corresponding key bits: the modular addition is a triangular function (meaning that the result of $a+b$ on a bit i only depends on the value of bits of position less than i in a and b), and therefore no XOR in the nonce in the less significant bits could control modular differences introduced in the padding in the more significant bits. An attack in that case would thus most likely be applicable to general ciphers when using only the ϕ^+ class, and it is proven that no such attack is efficient. However, one could always imagine using a related-key class using an addition operation reading the bits in reverse. While admittedly unorthodox, this would not result in a stronger model than using ϕ^+, strictly speaking.

Discussion. In a recent independent work, Dobraunig, Eichlseder and Mendel use similar methods to produce forgeries for Prøst-OTR by considering related keys with XOR differences [DEM15]. On the one hand, one could argue that the class $\phi^{\oplus}$ is more natural than ϕ^+ and more likely to arise in actual protocols, which would make their attack more applicable than ours. On the other hand, an ideal cipher is expected to give a similar security against RKA using either class, which means that our model is not theoretically

[1] The code is also available at: https://github.com/P1K/EMRKA.

Algorithm 4.1: Related-key key recovery for PRØST-OTR

Input: Oracle access to $\mathcal{F}(k,\cdot,\cdot,\cdot)$ and $\mathcal{F}(\phi^+(k),\cdot,\cdot,\cdot)$ for a fixed (unknown) key k of even length κ
Output: Two candidates for the key k

$k' := 0^\kappa$
for $i := \kappa - 2$ *to* $\kappa/2$ **do**
  REFRESH(n, m_1, m_2)
  $x := \mathcal{F}(k, n, m_1, m_2)$
  $y := \mathcal{F}(k + \Delta_i, n \oplus \Delta_i, m_1 \oplus \Delta_i \oplus \pi(\Delta_i), m_2)$
  if $x = y \oplus \Delta_i$ **then**
    $k'[i] := 0$
  else
    $k'[i] := 1$
for $i := \kappa/2 - 1$ *to* 0 **do**
  REFRESH(n, m_1, m_2)
  $x := \mathcal{F}(k + \Delta_i^+, n \oplus \Delta_{\kappa/2}, m_1 \oplus \Delta_{\kappa/2}, m_2)$
  $y := \mathcal{F}(k + \Delta_i^-, n, m_1 \oplus \pi(\Delta_{\kappa/2}), m_2)$
  if $x = y \oplus \Delta_{\kappa/2}$ **then**
    $k'[i] := 1$
  else
    $k'[i] := 0$
$k'' := k'$
$k''[\kappa - 1] := 1$
return (k', k'')

stronger than the one of Dobraunig *et al.*, while resulting in a much more powerful key recovery attack.

5 Conclusion

We made a simple observation that allows to convert related-key distinguishing attacks on some Even-Mansour ciphers into much more powerful key-recovery attacks, and we used this observation to derive an extremely efficient key-recovery attack on the PRØST-OTR CAESAR candidate.

Primitives based on the Even-Mansour construction are quite common, and it is natural to wonder if we could mount similar attacks on other ciphers. A natural first target would be PRØST-COPA which is also based on the PRØST/SEM cipher. However, in this mode, encryption and tag generation depend on the encryption of a fixed plaintext $\ell = \mathcal{E}(k, 0)$ which is different for different keys with overwhelming probability and makes our attack fail. The forgery attacks of Dobraunig *et al.* seem to fail in that case for the same reason. Keeping with CAESAR candidates, another good target would be Minalpher [STA+14], which also uses a one-round Even-Mansour block cipher as one of its components. The attack also fails in this case, though, because the masking key used in the Even-Mansour cipher is derived from the master key in a highly non-linear way. In fact, Mennink recently proved that both ciphers are resistant to related-key attacks [Men15]. Finally, leaving aside

authentication and going back to traditional block ciphers, we could consider designs such as LED [GPPR11]. The attack also fails in that case, however, because the cipher uses an iterated construction with at least 8 rounds and only one (or two) keys.

This lack of other results is not very surprising, as we only improve existing distinguishing attacks, and this improvement cannot be used without a distinguisher as its basis. Therefore, any primitive for which resistance to related-key attacks is important should already be resistant to the distinguishing attacks and thus to ours. Yet it would be reasonable and perfectly valid to allow the presence of a simple related-key distinguisher when designing a primitive, as this a very weak type of attack (in fact, this is for instance the approach taken by PRINCE, among others [BCG+12]). What we have shown is that one must be extremely careful when contemplating such a decision for Even-Mansour ciphers, as in that case it may actually be equivalent to allowing key-recovery, the most powerful of all attacks.

Acknowledgements. I am grateful to Jérémy Jean, Brice Minaud and the anonymous reviewers for their comments on this work.

References

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology — ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

BK03. Mihir Bellare and Tadayoshi Kohno. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2003.

CS14. Shan Chen and John P. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In Nguyen and Oswald [NO14], pages 327–350.

CS15. Benoit Cogliati and Yannick Seurin. On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology — EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 584–613. Springer, 2015.

DEM15. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Related-Key Forgeries for Prøst-OTR. *IACR Cryptology ePrint Archive*, 2015:91, 2015. *To appear in the proceedings of FSE 2015.*

DKS12. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology — EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2012.

EM91. Shimon Even and Yishay Mansour. A Construction of a Cioher From a Single Pseudorandom Permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances*

in Cryptology — ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1991.

FP14. Pooya Farshim and Gordon Procter. The Related-Key Security of Iterated Even–Mansour Ciphers. *IACR Cryptology ePrint Archive*, 2014:953, 2014. *To appear in the proceedings of FSE 2015.*

GPPR11. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems — CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

KLL+14. Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst. CAESAR Proposal, 2014. http://proest.compute.dtu.dk.

Men15. Bart Mennink. XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees. *IACR Cryptology ePrint Archive*, 2015:476, 2015.

Min14. Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In Nguyen and Oswald [NO14], pages 275–292.

NO14. Phong Q. Nguyen and Elisabeth Oswald, editors. *Advances in Cryptology — EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*. Springer, 2014.

STA+14. Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher. CAESAR Proposal, 2014. http://competitions.cr.yp.to/round1/minalpherv1.pdf.

A Example program for a 64-bit permutation

We give the source of a C program that recovers a 64-bit key from a design similar to PRØST-OTR where the permutation has been replaced by a small ARX, for compactness. For the sake of simplicity, we do not ensure that the nonce does not repeat in the queries. This code is also available at: https://github.com/P1K/EMRKA.

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define ROL32(x,r) (((x) << (r)) ^ ((x) >> (32 - (r))))
#define MIX(hi,lo,r) { (hi) += (lo); (lo) = ROL32((lo),(r)) ; (lo) ^= (hi); }

#define TIMES2(x) ((x & 0x8000000000000000ULL) ? ((x) << 1ULL) ^
    0x000000000000001BULL : (x << 1ULL))
#define TIMES4(x) TIMES2(TIMES2((x)))

#define DELTA(x) (1ULL << (x))
#define MSB(x) ((x) & 0xFFFFFFFF00000000ULL)
#define LSB(x) ((x) & 0x00000000FFFFFFFFULL)

/* Replace arc4random() by your favourite PRNG */

/* 64-bit permutation using Skein's MIX */
uint64_t p64(uint64_t x)
{
        uint32_t hi = x >> 32;
        uint32_t lo = LSB(x);
        unsigned rcon[8] = {1, 29, 4, 8, 17, 12, 3, 14};

        for (int i = 0; i < 32; i++)
        {
           MIX(hi, lo, rcon[i % 8]);
           lo += i;
        }

        return ((((uint64_t)hi) << 32) ^ lo);
}

uint64_t em64(uint64_t k, uint64_t p)
{
        return p64(k ^ p) ^ k;
}

uint64_t potr_1(uint64_t k, uint64_t n, uint64_t m1, uint64_t m2)
{
        uint64_t l, c;

        l = TIMES4(em64(k, n));
        c = em64(k, l ^ m1) ^ m2;

        return c;
}

uint64_t recover_hi(uint64_t secret_key)
{
        uint64_t kk = 0;

        for (int i = 62; i >= 32; i--)
```

```
        {
                uint64_t m1, m2, c11, c12, n;

                m1 = (((uint64_t)arc4random()) << 32) ^ arc4random();
                m2 = (((uint64_t)arc4random()) << 32) ^ arc4random();
                n  = (((uint64_t)arc4random()) << 32) ^ 0x80000000ULL;
                c11 = potr_1(secret_key, n, m1, m2);
                c12 = potr_1(secret_key + DELTA(i), n ^ DELTA(i), m1 ^ DELTA(i)
                    ^ TIMES4(DELTA(i)), m2);

                if (c11 != (c12 ^ DELTA(i)))
                        kk |= DELTA(i);
        }

        return kk;
}

uint64_t recover_lo(uint64_t secret_key, uint64_t hi_key)
{
        uint64_t kk = hi_key;

        for (int i = 31; i >= 0; i--)
        {
                uint64_t m1, m2, c11, c12, n;
                uint64_t delta_p, delta_m;

                m1 = (((uint64_t)arc4random()) << 32) ^ arc4random();
                m2 = (((uint64_t)arc4random()) << 32) ^ arc4random();
                n  = (((uint64_t)arc4random()) << 32) ^ 0x80000000ULL;

                delta_p = DELTA(i) - MSB(kk) + (((LSB(~kk)) >> (i + 1)) << (i +
                    1));
                delta_m = DELTA(i) + MSB(kk) + LSB(kk);
                c11 = potr_1(secret_key + delta_p, n ^ DELTA(32), m1 ^ DELTA(32)
                    , m2);
                c12 = potr_1(secret_key - delta_m, n, m1 ^ TIMES4(DELTA(32)), m2
                    );

                if (c11 == (c12 ^ DELTA(32)))
                        kk |= DELTA(i);
        }

        return kk;
}

int main()
{
        uint64_t secret_key = (((uint64_t)arc4random()) << 32) ^ arc4random();
        uint64_t kk1 = recover_lo(secret_key, recover_hi(secret_key));
        uint64_t kk2 = kk1 ^ 0x8000000000000000ULL;

        printf("The real key is %016llx, the key candidates are %016llx, %016llx
                 ", secret_key, kk1, kk2);
        if ((kk1 == secret_key) || (kk2 == secret_key))
                printf("SUCCESS!\n");
        else
                printf("FAILURE!\n");

        return 0;
}
```