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 the distinguishing attacks on Even-Mansour block ciphers in the related key model can easily be converted into extremely efficient key recovery attacks. This includes in particular all iterated Even-Mansour constructions 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 $\mathrm{E}((k_1, k_0), p)$ of the plaintext p under the (possibly equal) keys k_0 and k_1 as $\mathrm{P}(p \oplus k_0) \oplus k_1$, where 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, many later works investigated the security of variants of the Even-Mansour construction. A simple one is the iterated Even-Mansour scheme with independent keys and independent permutations, with its r-round version defined as $\mathrm{IEM}^r((k_{r-1}, k_{r-2}, \ldots, k_0), p) = \mathrm{P}_{r-1}(\mathrm{P}_{r-2}(\ldots \mathrm{P}_0(p \oplus k_0) \oplus k_1) \ldots) \oplus k_{r-1}$. 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 such a model is necessary. Yet until the recent works of Cogliati and Seurin & Farsim and Procter, no 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 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 [CS15,FP14]. While related-key analysis obviously gives much more power to the attacker than the single-key setting, it is a widely

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

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 contributions. We show that the distinguishing attacks on the Even-Mansour constructions 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 concerns both the trivial distinguishers on iterated Even-Mansour with independent keys and 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 Notations

We use $\bullet$ 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 $0^{n-i-1} \bullet 1 \bullet 0^{i-1}$. 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 $\oplus$ (XOR) and $+$ (modular addition).

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

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 difference 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 $\mathrm{E}(k,\cdot)$, $\mathrm{E}(\phi^{\oplus}(k),\cdot)$ and $\mathrm{E}(\phi^{+}(k),\cdot)$ for the block cipher E with κ-bit keys, we can easily learn the value of the bit $k[i]$ of k by querying $\mathrm{E}(k+\Delta_i, p)$ and $\mathrm{E}(k\oplus\Delta_i, p)$ and by comparing their values. For $i < \kappa - 1$, the plaintext p is encrypted under the same key iff $k[i] = 0$, 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.

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 difference 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 E using the two classes $\phi^{\oplus}(k)$ and $\phi^{+}(k)$

does not say much on 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 construction, we explicit the trivial related-key distinguishers mentioned in the introduction, that exists for r-round iterated Even-Mansour block ciphers with independent keys. These distinguishers using only key 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 $\mathrm{E}((k_{r-1}, k_{r-2}, \ldots, k_0), p)$ and $\mathrm{E}((k_{r-1}, k_{r-2}, \ldots, k_0 \oplus \delta), p \oplus \delta)$ are equal for any difference δ when $\mathrm{E} = \mathrm{IEM}^r$ and that this equality does not hold in general, thence allowing to distinguish IEM^r from an ideal cipher.

We now show how this distinguisher can be combined with the attack using two classes 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 $\mathrm{E}((k_1, k_0), p) = \mathrm{P}(p \oplus k_0) \oplus k_1$; one can learn the value of the bit $k_0[i]$ by querying $\mathrm{E}((k_1, k_0), p)$ and $\mathrm{E}((k_1, k_0 + \Delta_i), p \oplus \Delta_i)$ 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 construction 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 construction [DKS12]. An attack in that case works similarly by querying *e.g.* $\mathrm{E}((k_1, k_0), \Delta_i)$ and $\mathrm{E}((k_1, k_0 \oplus \Delta_i), 0^\kappa)$.

In both cases, the attacks use a single difference class for the related keys (either $\phi^{\oplus}$ or ϕ^{+}), and 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 constructions. 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 not independent and are derived from a master key by a linear key schedule [CS15]. It is then only fair to wonder if this distinguisher can also be extended to a key-recovery attack; we answer positively to this question and show the procedure in the case where all keys are equal:

1. Query $y_1 := \mathrm{E}(k + \Delta_1, x_1)$
2. Set x_2 to $x_1 \oplus \Delta_2 \oplus \Delta_1$ and query $y_2 := \mathrm{E}(k + \Delta_2, x_2)$
3. Set y_3 to $y_1 \oplus \Delta_1 \oplus \Delta_3$ and query $x_3 := \mathrm{E}^{-1}(k + \Delta_3, y_3)$
4. Set Δ_4 to $\Delta_3 \oplus \Delta_2 \oplus \Delta_1$
5. Set y_4 to $y_2 \oplus \Delta_2 \oplus \Delta_4$ and query $x_4 := \mathrm{E}^{-1}(k + \Delta_4, y_4)$

6. Test if $x_4 = x_3 \oplus \Delta_3 \oplus \Delta_4$

If the test is successful, it means that with overwhelming probability the key bits at the positions of the differences $\Delta_1, \Delta_2, \Delta_3$ are all zero. As soon as one has found two such bits (which happens after an expected four trials for random keys), the rest of the key bits can be tested one by one. The case where the keys are not equal but generated from a linear key schedule works exactly in the same way, and only the XOR differences need to be adapted in consequence.

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. In the case of PRØST-OTR, this is a block cipher defined as a one-round Even-Mansour construction using the PRØST permutation with identical keys; PRØST-COPA similarly defines a tweakable block cipher based on the same underlying Even-Mansour construction. Because our attack works regardless of the actual permutation, 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. We describe how the encryption of the first block of plaintext is performed, which is the only thing necessary for our attack, but also show the parallel Feistel approach of OTR in Figure 4.1.

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 $\mathscr{F}(k, n, m_1, m_2)$ of k, n, and the first two blocks of plaintext m_1 and m_2: let $\ell := \mathrm{E}(k, n \bullet 10^*)$ 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 $\mathrm{E}(k, \ell' \oplus m_1) \oplus m_2$. Let us now apply the attack from Section 3.

A first attack that does not quite work. 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 $c_1 = \mathscr{F}(k, n, m_1, m_2)$ and $\hat{c}_1 = \mathscr{F}(k + \Delta_i, n \oplus \Delta_i, m_1 \oplus \Delta_i \oplus \pi(\Delta_i), m_2)$. Indeed, if this bit is equal to 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 this key bit is 1, the latter equality does not hold with overwhelming probability.

Yet this does not work quite completely, because the nonce in PRØST-OTR is restricted to a size half of the one of the block cipher 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

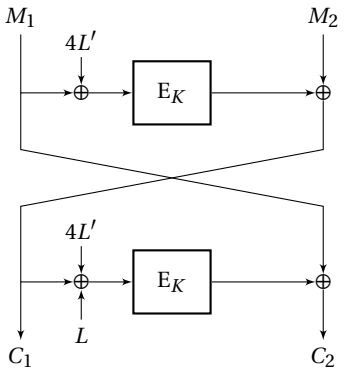

Fig. 4.1. The encryption of the first two blocks of message with OTR.

cannot introduce appropriate differences in the computation 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. 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 $2^{\frac{\kappa}{2}}$.

REMARK. This procedure does not either recover the most significant bit of the key, as an addition on this bit never generates a carry. This bit can of course easily be recovered at the cost of one additional query once all the others have been determined.

A recursive procedure to recover the rest of the key. Although we just saw that 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 generic procedure) after a single encryption by E. For the sake of simplicity, we assume for now that the two keys of E are independent, and we omit to write the second (outer) key. We note k^{MSB} the (known) most significant half of the (inner) key. Then one queries $\text{E}(k+\Delta_{\kappa/2-1}-k^{\text{MSB}}, p\oplus\Delta_{\kappa/2})$ and compares it to $\text{E}(k-\Delta_{\kappa/2-1}-k^{\text{MSB}}, p)$. We can see that in this case the two inputs to the permutation of E are equal iff $k[\kappa/2-1]=1$. Indeed, the carry in the addition $k+\Delta_{\kappa/2-1}$ propagates by exactly one position and is "cancelled" by the difference in p, and there is no carry propagation in $k-\Delta_{\kappa/2-1}$. If on the other hand $k[\kappa/2-1]=0$, the latter carry propagates all the way to the most significant bit, whereas only two differences are introduced in the first query. Now that the value of $k[\kappa/2-1]$ has been learned, one can iterate the process to recover the remaining bits. The only subtlety is that we want to ensure that if there is a carry propagation in $k+\Delta_{\kappa/2-1-i}$ (resp. $k-\Delta_{\kappa/2-1-i}$), it propagates up to $k_{\kappa/2-1+1}$, the position where we cancel it with an XOR

difference (resp. up to the most significant bit). This can be easily achieved by adding two terms to both keys: if we call γ_i the value of the key k only on positions $\kappa/2-1\ldots\kappa/2-i$, completed with zeros left and right, and $\tilde{\gamma}_i$ its binary complement on the same positions, we simply have to query $\mathrm{E}(k+\Delta_{\kappa/2-1}-k^{\mathrm{MSB}}+\tilde{\gamma}_i, p\oplus\Delta_{\kappa/2})$ and $\mathrm{E}(k-\Delta_{\kappa/2-1}-k^{\mathrm{MSB}}-\gamma_i, p)$. We finally remark that in the event of the bit $k[\kappa/2-1-i]$ being one, the (identical) results of the queries are equal to encrypting p under k with all its bits of position higher than $\kappa/2-1-i$ set to zero. This shows why even when the outer key is chosen equal to the inner key, we can deduce the difference (modular as well as XOR) between the two results, and thence are still able to recover the value of this particular key bit; this difference is in fact independent of i and 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 $\mathscr{F}(k+\Delta_i^+, n\oplus\Delta_{\kappa/2}, m_1\oplus\Delta_{\kappa/2}, m_2)$ and $\mathscr{F}(k+\Delta_i^-, n, m_1\oplus\pi(\Delta_{\kappa/2}), m_2)$, 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 could easily be implemented. We give an example of such a program for an 8-bit toy cipher in the Appendix A.

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

Input: Oracle access to $\mathscr{F}(k,\cdot,\cdot,\cdot)$ and $\mathscr{F}(\phi^+(k),\cdot,\cdot,\cdot)$ 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)
    $x := \mathscr{F}(k, n, m_1, m_2)$
    $y := \mathscr{F}(k+\Delta_i, n\oplus\Delta_i, m_1\oplus\Delta_i\oplus\pi(\Delta_i), m_2)$
    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)
    $x := \mathscr{F}(k+\Delta_i^+, n\oplus\Delta_{\kappa/2}, m_1\oplus\Delta_{\kappa/2}, m_2)$
    $y := \mathscr{F}(k+\Delta_i^-, n, m_1\oplus\pi(\Delta_{\kappa/2}), m_2)$
    if $x = y\oplus\Delta_{\kappa/2}$ **then**
        $k'[i] := 1$
    else
        $k'[i] := 0$
$k'' := k'$
$k''[\kappa-1] := 1$
return (k', k'')

REMARK. If the padding of the nonce were done on the most significant bits, no similar attack could recover these 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 differences introduced in the padding. An attack in that case would thus most likely be applicable for 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 class using an addition operation reading the bits in reverse. While admittedly unorthodox, this would not result in a stronger model than using ϕ^+.

Discussion. One should be aware that the designers of PRØST do not make any claim for the resistance of PRØST-OTR to related-key attack, and therefore we do not consider ours to be an attack on PRØST *per se*, and we do not pretend to break the security of this primitive.

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 may 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 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 constructions 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 an 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 uses an Even-Mansour construction similar to PRØST-OTR. However, in this mode, encryption and tag generation depend on the encryption of a fixed plaintext $\ell = \mathrm{E}(k,0)$ 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 construction is derived from the master key in a highly non-linear way. In fact, the construction used in Minalpher may actually benefit from the security proof of Cogliati and Seurin[1]. 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.

[1] Their proof requires two distinct non-linear *permutations*, whereas Minalpher uses the same non-linear *function* to derive both keys. We did not investigate the impact of these differences, but it is reasonable to expect a good security for this construction.

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 type. What we have shown is that one must be extremely careful when contemplating such a decision for Even-Mansour constructions, as in that case it may actually be equivalent to allowing key-recovery, the most powerful of all attacks.

References

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. Benoît Cogliati and Yannick Seurin. On the Provable Security of the Iterated Even-Mansour Cipher against Related-Key and Chosen-Key Attacks. *IACR Cryptology ePrint Archive*, 2015:69, 2015.

DEM15. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Related-Key Forgeries for Prøst-OTR. *IACR Cryptology ePrint Archive*, 2015:91, 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.

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.

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 an 8-bit permutation

We give the source of a C program that recovers an 8-bit key from a design similar to PRØST-OTR where the permutation has been replaced by the AES S-box, for compactness. For the sake of simplicity, we do not ensure that the nonce does not repeat in the queries. Amusingly, for this size of permutation, the attack is not better than the targeted security level of $2^{\frac{8}{2}} = 16$.

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

static uint8_t key = 0xD2;
static uint8_t nonce = 0;
static uint8_t aes_s[256] =
{
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B,
        0xFE, 0xD7, 0xAB, 0x76, 0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
        0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0, 0xB7, 0xFD, 0x93, 0x26,
        0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2,
        0xEB, 0x27, 0xB2, 0x75, 0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
        0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84, 0x53, 0xD1, 0x00, 0xED,
        0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F,
        0x50, 0x3C, 0x9F, 0xA8, 0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
        0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2, 0xCD, 0x0C, 0x13, 0xEC,
        0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
        0xDE, 0x5E, 0x0B, 0xDB, 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
        0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79, 0xE7, 0xC8, 0x37, 0x6D,
        0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F,
        0x4B, 0xBD, 0x8B, 0x8A, 0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
        0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E, 0xE1, 0xF8, 0x98, 0x11,
        0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F,
        0xB0, 0x54, 0xBB, 0x16
};

#define P(x) aes_s[(x)]
#define TIMES2(x) ((x & 0x80) ? ((x) << 1) ^ 0x1b : (x << 1))
#define TIMES4(x) TIMES2(TIMES2((x)))
#define REFRESH(x) ((x) + 1)
#define DELTA(x) (1 << (x))
#define MSB(x) ((x) & 0xF0)
#define LSB(x) ((x) & 0x0F)

uint8_t em8(uint8_t k, uint8_t p)
{
        return P(k ^ p) ^ k;
}

uint8_t potr_1(uint8_t k, uint8_t n, uint8_t m1, uint8_t m2)
{
        uint8_t l, c;

        l = TIMES4(em8(k, n));
        c = em8(k, l ^ m1) ^ m2;

        return c;
}
```

```
uint8_t recover_hi(void)
{
        uint8_t kk = 0;

        for (int i = 6; i >= 4; i--)
        {
                uint8_t m1, m2, c11, c12, n;

                m1 = 17; m2 = 19;
                n = (nonce << 4) ^ 0x8;
                c11 = potr_1(key, n, m1, m2);
                c12 = potr_1(key + DELTA(i), n ^ DELTA(i), m1 ^ DELTA(i) ^
                    TIMES4(DELTA(i)), m2);

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

                nonce = REFRESH(nonce);
        }

        return kk;
}

uint8_t recover_lo(uint8_t hi_key)
{
        uint8_t kk = hi_key;

        for (int i = 3; i >= 0; i--)
        {
                uint8_t m1, m2, c11, c12, n;
                uint8_t delta_p, delta_m;

                m1 = 17; m2 = 19;
                n = (nonce << 4) ^ 0x8;
                delta_p = DELTA(i) - MSB(kk) + (((LSB(~kk)) >> (i + 1)) << (i +
                    1));
                delta_m = DELTA(i) + MSB(kk) + LSB(kk);
                c11 = potr_1(key + delta_p, n ^ DELTA(4), m1 ^ DELTA(4), m2);
                c12 = potr_1(key - delta_m, n, m1 ^ TIMES4(DELTA(4)), m2);

                if (c11 == (c12 ^ DELTA(4)))
                        kk |= DELTA(i);

                nonce = REFRESH(nonce);
        }

        return kk;
}

int main()
{
        uint8_t keycand = recover_lo(recover_hi());
        printf("The key candidates are %02X and %02X\n", keycand, keycand ^ 0x80
            );
        return 0;
}
```