Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE

Itai Dinur

Département d'Informatique, École Normale Supérieure, Paris, France

Abstract. The FX-construction was proposed in 1996 by Kilian and Rogaway as a generalization of the DESX scheme. The construction increases the security of an n-bit core block cipher with a κ-bit key by using two additional n-bit masking keys. Recently, several concrete instances of the FX-construction were proposed, including PRINCE (proposed at Asiacrypt 2012) and PRIDE (proposed at CRYPTO 2014). These ciphers have $n = \kappa = 64$, and are proven to guarantee about $127 - d$ bits of security, assuming that their core ciphers are ideal, and the adversary can obtain at most 2^d data.

In this paper, we devise new cryptanalytic time-memory-data tradeoff attacks on FX-constructions, combining recent techniques by Fouque, Joux and Mavromati with time-memory-data tradeoffs for stream ciphers. While our attacks do not contradict the security proof of PRINCE and PRIDE, nor pose an immediate threat to their users, some specific choices of tradeoff parameters demonstrate that the security margin of the ciphers against practical attacks is smaller than expected. Finally, we propose very light changes to PRINCE and PRIDE. These changes ensure that the ciphers resist our attacks while maintaining their design goals, with the exception of the theoretical security proof (which is invalidated, as PRINCE and PRIDE are no longer FX-constructions). Consequently, we conclude that although the FX-construction provides a very simple way of increasing the security of a widely deployed cipher (such as DES at the time), using it for a new design is a less reasonable approach.

Keywords: Cryptanalysis, block cipher, time-memory-data tradeoff, FX-construction, DESX, PRINCE, PRIDE.

1 Introduction

The Advanced Encryption Standard (AES) is the most widely used block cipher today. It is believed to guarantee a large security margin again practical attacks, and can therefore be used to encrypt very sensitive data. The AES was preceded by the Data Encryption Standard (DES), whose 56-bit key made it vulnerable to straightforward exhaustive search. Consequently, in 1984, when DES was still widely used, Ron Rivest proposed a simple solution (known as DESX [20]) to address the concern regarding its small key size. The DESX construction simply XORs two independent 64-bit keys at the beginning and at the end of the core DES encryption process, such that the total key size becomes $56 + 64 + 64 = 174$ bits. This construction was generalized to the so-called *FX-construction* by Kilian and Rogaway in 1996 [17]. The FX-construction is built using an (arbitrary) n-bit block cipher F_K with a κ-bit key K and two additional n-bit whitening keys K_1, K_2, and defined as $FX_{K,K_1,K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$. Kilian and Rogaway proved that the cipher guarantees $\kappa + n - d - 1$ bits of security,[1] assuming that F is a perfect block cipher and the adversary can obtain $D = 2^d$ plaintext-ciphertext pairs. Furthermore, Kilian and Rogaway showed that the bound is tight by extending the attack of Daemen [11] (on the related Even-Mansour construction) to a simple attack on the FX-construction with complexity of about $2^{\kappa+n-d-1}$.

The analysis of Kilian and Rogaway implies that the security of the FX-construction depends on how much data the attacker can obtain, and thus the security is not completely determined by the computational power of the attacker. This is a unique situation, as for (almost) all block ciphers used in practice today that have no known weaknesses, obtaining additional data does not seem to give any significant advantage in key recovery attacks. Thus, the security level of $\kappa + n - d - 1$ does not allow to directly compare FX-constructions to classical ciphers, and does not give a clear indication on the effort required in order to break such a construction.

Until recently, the security guaranteed by FX-constructions was perhaps not very relevant, as (to the best of our knowledge) such constructions were not proposed for practical use (apart from DESX). This situation changed in 2012, when the FX-construction PRINCE was presented at Asiacrypt [7], and more recently, at CRYPTO 2014, a similar FX-construction (named PRIDE [1]) was proposed.[2] Both of these constructions have $n = \kappa = 64$, and thus they offer security of about $127 - d$ bits, assuming that their core ciphers are ideal.[3]

In order to encourage its adoption by the industry, the designers of PRINCE launched a competition (named the PRINCE Challenge [22]), calling for cryptanalysis of the cipher which would lead to better understanding of its security. The competition focuses on practical attacks on round-reduced variants of PRINCE, where a practical attack is defined to have data complexity of (up to) 2^{30} known plaintexts (or 2^{20} chosen plaintexts), time complexity of 2^{64} and memory complexity of 2^{45} bytes.

[1] A cipher guarantees b bits of security if the complexity of the most efficient attack on it is at least 2^b.

[2] PRINCE and PRIDE are FX-constructions of a particular type, where K_2 linearly depends on K_1. However, it is shown that the smaller key size does not reduce the security of the scheme against generic attacks.

[3] PRINCE guarantees slightly less than $127 - d$ bits of security, as its core cipher was designed to preserve a special property that ensures a small footprint.

Motivated by the PRINCE Challenge, in this paper, we investigate the security margin guaranteed by FX-constructions against practical attacks, with a focus on PRINCE and PRIDE (i.e, FX-constructions with $n = \kappa = 64$). We first analyze well-known generic attacks on FX-constructions [5, 12, 17], and conclude that these attacks do to threaten the security of PRINCE and PRIDE. Then, we devise new attacks with lower memory complexity, and claim that the security margin of FX-constructions with $n = \kappa = 64$ against these attacks is somewhat reduced (although the attacks remain impractical).

Despite the new attacks described above, our most interesting attacks are carried out in Hellman's time-memory tradeoff model [15]. In this model, the adversary spends a lot of resources on a one-time preprocessing phase that analyzes the scheme, and whose output is stored in (relatively small) memory. After this one-time preprocessing phase is completed, the scheme can be attacked much more efficiently, and this makes Hellman's model attractive in many cases.

The starting point of our attacks is a recent analysis of the FX-construction and related designs by Fouque et al. [14]. One of the attacks of [14] on PRINCE has data complexity of 2^{32} and a very efficient time complexity of 2^{32}. The main shortcomings of this attack are its huge memory complexity of about 2^{67} bytes, and its impractical preprocessing phase, which has time complexity of 2^{96}. The techniques we develop trade off these high memory and preprocessing complexities with time and data complexities, and allow to obtain more balanced and practical tradeoffs.

Some concrete parameters of our attacks on PRICE and PRIDE in Hellman's model are summarized in Table 1. Consider the online phase of Attack 1, which requires about 2^{32} data (32 gigabytes), takes 2^{64} time and requires 2^{51} bytes of memory. The parameters of this attack are thus not far from the parameters considered in the PRINCE challenge [22] as practical, and they are valid regardless of the cipher's internal number of rounds. Furthermore, we show in this paper that Attack 1 (as well as our other online attacks in Table 1) rarely accesses the memory (which can be stored on a hard disk), and can be efficiently realized using dedicated hardware with a budget of a medium-size enterprize. Therefore, we consider this attack to be semi-practical.

Attack 1 has two main shortcomings: the first shortcoming is that it requires the 2^{32} data in the form of adaptively chosen plaintexts, which is a rather strict requirement. However, the data collection may be practical in some restricted settings (e.g., if the attacker can obtain black-box access to the encryption device for a few minutes). The second limitation of the attack is more significant, as it requires a long and impractical (to the best of our knowledge) one-time precomputation phase of[4] complexity 2^{96}. The impracticality of the preprocessing phase should thus ensure that PRINCE and PRIDE remain out of the reach of this attack in the near future.

[4] Note that according to the bound of [17], any generic attack on FX-constructions with $n = \kappa = 64$ using 2^{32} data, must have time complexity of at least $2^{64+64-32-1} = 2^{95}$.

In order to reduce the preprocessing complexity, we consider Attack 2 which exploits a larger number of 2^{40} adaptively chosen plaintexts. This data can be collected (for example) if the attacker can obtain black-box access to the encryption device for a few hours, and can thus be considered practical in some (restricted) scenarios. The online attack runs in time 2^{56}, requires 2^{51} bytes of storage, and is therefore even more efficient than the online phase of Attack 1. More significantly, it requires a shorter precomputation phase of time complexity 2^{88}, which is still impractical, but only marginally.[5]

An interesting observation is that when we execute a $0 < p \leq 1$ fraction of the preprocessing phase of Attack 2, then the key recovery attack succeeds with probability p. If we consider $p = 2^{-8}$, the attack succeeds with a non-negligible probability of $p = 1/256$, requires only $2^{51-8} = 2^{43}$ bytes of memory (or 8 terabytes), and can be implemented today with a small academical budget (of about 10,000 dollars). Moreover, the preprocessing time complexity of the attack above becomes $2^{88-8} = 2^{80}$ (which is more practical than 2^{88}). This shows that it may be beneficial to start the preprocessing phase today, instead of waiting for the technology that would make it fully realizable in the future. We further note that the complexity parameters of Attack 2 for $p = 2^{-8}$ and $n = \kappa = 64$ are, in fact, equivalent to those for $p \approx 1$ and $n = 64, \kappa = 56$. Since DES has a 56-bit key, the full attack against DESX could potentially be carried out today by a resourceful adversary.

Table 1. Attacks on PRINCE and PRIDE

Attack ID	Reference	Data (Adaptively chosen plaintexts)	Preprocessing Time	Online Time	Memory (Bytes)
–	[14]	2^{32}	2^{96}	2^{32}	2^{67}
1	This paper	2^{32}	2^{96}	2^{64}	2^{51}
2	This paper	2^{40}	2^{88}	2^{56}	2^{51}
3	This paper	2^{40}	2^{88}	2^{64}	2^{47}
4	This paper	2^{48}	2^{80}	2^{64}	2^{51}

Although the FX-construction is a block cipher, our techniques are borrowed from cryptanalysis of stateful ciphers (i.e., stream ciphers). We first notice that the FX-construction can be viewed as a (standard) core block cipher with an additional secret state (namely, the input or output to the core block cipher), which is hidden by the masking keys using simple XOR operations. Our main methodological contribution it to use Hellman's time-memory tradeoff to in-

[5] We assume that some adversaries can spend a huge amount of resources on preprocessing (in contrast to online attacks). Therefore, we consider preprocessing time complexity of 2^{80} to be (marginally) practical, as demonstrated by the capacity of the Bitcoin network [6], and supported by the NIST recommendation to disallow 80-bit keys after 2014 [19].

vert a set of special states, similarly to the techniques that Biryukov, Shamir and Wagner applied to stream ciphers which have low sampling resistance [3, 4]. However, unlike the case of stream ciphers, in some cases we analyze (in particular for $d > n/2$), we have to request the data and optimize our algorithms in a non-trivial way in order to obtain efficient tradeoffs.

A unique feature of our time-memory-data tradeoff curve for $2^d \leq 2^{n/2}$, is that the effective hidden state (key) size of the FX-construction is reduced by a factor of $2^{1.5d}$ in the online phase of the attack. On the other hand, for stream ciphers, the effective hidden state size is only reduced by a factor of 2^d. The reason for this is that in most stream ciphers, the hidden state is permuted and its entropy is maintained when producing keystream. On the other hand, we exploit the basic technique of Fouque et al., which applies a non-bijective function to the hidden state of the FX-construction, reducing its entropy. In particular, for $\kappa = n = 64$ and $2^d = 2^{32}$, the effective key size is reduced to $64+64-1.5\cdot32 = 80$ bits, whereas one could have expected it to be $64+64-32 = 96$ bits. Exploiting memory to further reduce the effective key size, leads to online key recovery attacks on PRINCE and PRIDE with semi-practical complexities.

In the final part of this paper, we show that PRINCE and PRIDE can be easily tweaked to provide significantly improved security against generic attacks. These very modest tweaks allow the ciphers to maintain all of their design goals, with the exception of the security proof of [17] (which is invalidated, as the ciphers are no longer FX-constructions). This suggests that the DESX solution proposed by Ron Rivest in 1984 (in order to provide better security for the widely-deployed DES) may be less suitable for new ciphers, which can be easily designed to provide much higher security than DESX (or similar designs). Although the FX-construction provides a theoretical security proof, it may not be a sufficiently good reason to reduce the security margin of the cipher against practical attacks.

The paper is organized as follows. We begin by introducing our notation in Section 2, while Section 3 provides an overview of our attacks. Section 4 gives some necessary background, and our simple attacks (that do not use a preprocessing phase) are described in Section 5, while our advanced attacks are described in Section 6. Finally, we propose tweaks to PRINCE and PRIDE in Section 7 and conclude the paper in Section 8.

2 Notations and Conventions

The FX-construction [17] is built using an (arbitrary) n-bit block cipher F_K with a κ-bit key K and 2 additional n-bit whitening keys K_1, K_2, and defined as $FX_{K,K_1,K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$. We denote the plaintext by P, its ciphertext $FX_{K,K_1,K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$ by C, and the inner values $K_1 \oplus P$ and $F_K(K_1 \oplus P)$ by X and Y, respectively (see Figure 1).

In this paper, we are also interested in more specific instances of the FX-construction. In particular, this construction also inspired the design of the Even-Mansour scheme [13], in which the core cipher is, in fact, an unkeyed public

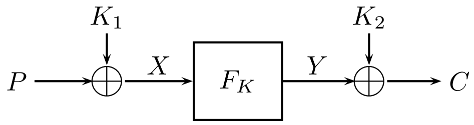

Fig. 1. The FX-Construction

permutation F (for which $\kappa = 0$). Furthermore, a major focus of this paper is placed on the recently proposed concrete FX-constructions PRINCE [7] and PRIDE [1]. These constructions use only n bits of whitening key material K_1, where K_2 is defined by $A(K_1)$, for an invertible affine function A. We refer to this simplified scheme as an *SFX-construction*.

As we deal with various tradeoffs between complexity parameters of attacks, we define some notation that is used in order to quantify these parameters. For an n-bit block cipher, we denote $N = 2^n$. When considering an attack, we denote by T its total time complexity, where a unit of time corresponds to an encryption of a single plaintext. We denote by $M = 2^m$ the memory complexity of the attack in terms of n-bit words, and by $D = 2^d$ its data complexity in terms of plaintext-ciphertext pairs. Finally, we denote by $\hat{T}$ the preprocessing time complexity of the attack (if the attack does not require preprocessing, then $\hat{T} = 0$).

The complexity of our attacks throughout this paper is formulated in terms of the parameters n, κ, d, m defined above. For the sake of convenience, we assume in several parts of this paper that $\kappa = n$, which is the case for PRINCE and PRIDE. We note that if $\kappa > n$, the attacks we describe have a small penalty of about $\lceil \kappa/n \rceil$ in the time and memory complexities.[6]

Since we estimate the practicality of some of our attacks, it is insufficient to merely compute their time, data, and memory complexities. Indeed, the practicality of an attack is largely influenced by more subtle properties such as the number of memory accesses during its execution (which determines whether the memory has to be stored in RAM, or can be stored on a cheaper hard disk), and whether the workload of the attack can be easily parallelized (i.e., divided across different CPUs). Another crucial element is the size of the implementation circuit, which determines whether the attack can be efficiently realized on cheap dedicated hardware.

3 Overview of Previous and New Attacks on FX-Constructions

The new and previously published tradeoffs for FX-constructions are summarized in Table 2. We now compare these attacks at a high level, and then em-

[6] The ratio between the key size and block size is typically small in modern block ciphers.

phasize their practicality for $n = \kappa = 64$, focusing on the concrete parameters for the attacks given in Table 1 that use preprocessing.[7]

Attacks without Preprocessing We first examine attacks with no preprocessing, for which an initial chosen plaintext attack was described in [17]. Then, a known plaintext attack with the same complexity was published in [5] for $D = 2^{n/2}$, and later generalized in [12] to work with any number of known plaintexts. All of these attacks require $M = D$ memory and seem impractical for $n = \kappa = 64$, as they either require impractical data and memory (e.g. for $D = M = 2^{64}$), or impractical time (e.g., for $D = M = 2^{32}$ then $T = 2^{96}$). Intermediate values such as $D = M = 2^{48}$ and $T = 2^{80}$ may seems more practical, but we note that the large memory of 2^{48} needs to be accessed a huge number of 2^{80} times (essentially, for each cipher evaluation). While this can be somewhat optimized by grouping together the memory accesses, these parameters still seem completely impractical.

The new attacks we describe in Section 5 show that in the adaptively chosen plaintext model, we can mount attacks with the same data and time complexities, and a reduced memory complexity. In particular, for $D = 2^{n/2}$, our attack requires negligible memory, while the attacks of [5, 12, 17] require $2^{n/2}$ memory. For $n = \kappa = 64$ and $D = 2^{48}$, the attack requires only $2^{2 \cdot 48 - 64} = 2^{32}$ words of memory, which is significantly better than the attack of [12]. In this case, the time complexity is 2^{80}, which can be considered as marginally practical, but is still a huge effort put into recovering a single key. Furthermore, as the data collection cannot be parallelized, obtaining 2^{48} data is generally not considered practical. Nevertheless, we still believe that the attack for $D = 2^{48}$ serves as an initial indication that the security margin of PRINCE and PRIDE is smaller than expected.

Attacks with Preprocessing We examine the attacks that require preprocessing assuming $2^d \leq 2^{n/2}$, for which the previously published attack was given in [14]. This attack has a very efficient online time complexity of 2^d, but requires $2^{\kappa+n-2d} \geq 2^{\kappa}$ words of memory. Our attack trades-off this memory at the expense of increasing the time complexity, and obtains $T = 2^{2(\kappa+n-m-1.5d)}$.

For $d > n/2$, assuming $2^m \leq 2^{\kappa+n-2d}$, we can further reduce the preprocessing complexity and obtain a more efficient online time complexity of $2^{\kappa+n/2-m-d/2}$. Concrete parameter sets for ciphers with $n = \kappa = 64$ are given in Table 1.

The Practicality of Our Attacks As we show in sections 6.1 and 6.2, the online attacks summarized in Table 1 rarely access the memory, which can be stored on a hard disk. Moreover, the attacks can be easy parallelized, and although they are conceptually non-trivial, their circuit sizes are almost as small

[7] We note that optimal choice of parameters and infrastructure to realize an attack depends of the setting, and there are many more options than listed in Table 1.

as the circuits of the attacked FX-constructions. Therefore, the attacks can be efficiently implemented on dedicated hardware.

As a consequence of the above, we estimate that the online phase of the attacks in Table 1 (which require at most 2^{64} cipher evaluations and 2^{51} bytes of storage) can be realized today by a medium-size enterprize with a budget of a few hundred thousand dollars. This rough estimation is based on the fact that up to about 2^{64} cipher evaluations can be performed in a few weeks on dedicated hardware with such a budget [10]. As for storage, a standard 1-terabyte hard disk costs less than 100 dollars (as of 2014). Therefore, 2^{51} bytes of storage (or about 2,000 terabytes) cost roughly 200,000 dollars. Of course, fully realizing such an attack has additional costs, but we do not expect them to increase the overall cost by a significant factor.

Table 2. Time-Memory-Data Tradeoffs for FX-Constructions

Reference	Data	Preprocessing Time	Online Time	Memory
[12]	$2^d \leq 2^n$ KP[†]	-	$2^{\kappa+n-d}$	2^d
Section 5.1	$2^d \leq 2^{n/2}$ ACP[††]	-	$2^{\kappa+n-d}$	negligible
Section 5.2	$2^d > 2^{n/2}$ ACP[††]	-	$2^{\kappa+n-d}$	2^{2d-n}
[14]	$2^d \leq 2^{n/2}$ ACP[††]	$2^{\kappa+n-d}$	2^d	$2^{\kappa+n-2d}$
Section 6.1	$2^d \leq 2^{n/2}$ ACP[††]	$2^{\kappa+n-d}$	$2^{2(\kappa+n-m-1.5d)}$	2^m
Section 6.2	$2^d > 2^{n/2}$ ACP[††]	$2^{\kappa+n-d}$	$2^{2(\kappa+n/2-m-d/2)}$	$2^m \leq 2^{\kappa+n-2d}$
Section 6.2	$2^d > 2^{n/2}$ ACP[††]	$2^{\kappa+n-d}$	$2^{\kappa+d-m}$	$2^m > 2^{\kappa+n-2d}$

[†] Known plaintexts

[††] Adaptively chosen plaintexts

4 Background

The new attacks described in this paper combine several previously published techniques, which are described in this section.

4.1 Hellman's Time-Memory Tradeoff [15] (with Preprocessing)

We summarize Hellman's classical time-memory tradeoff attack [15] on an n-bit block cipher E_K, assuming that $\kappa = n$. In the preprocessing phase, we fix a plaintext P, and define the function $h(\{0,1\}^n) \rightarrow \{0,1\}^n$ as $h(K) = E_K(P)$. The goal in this phase is to cover most (more than half) of the key space with chains defined by iterating the function h. For parameters M' and T', we choose M' arbitrary starting points for the chains, where each chain is of length T'.

We store in a table only the (startpoint, endpoint) pair[8] of each chain and sort the table according to the endpoint value. Such a table requires M' words of memory, and is referred to as a *Hellman table*.

After evaluating M' chains, and reaching the birthday bound (stopping rule) of $T' \cdot M'T' = N$, adding additional chains to the table is wasteful. Thus, we can cover $M'T' = N/T'$ points with M' words of memory. In order to cover the full key space, we use flavors of h, where flavor i is defined (for example) as $h^{[i]}(K) = h(K)+i$. Thus, we use T' flavors of h, and compute a Hellman table for each flavor, covering a total of about $N/T' \cdot T' = N = 2^{\kappa}$ keys as required. The T' tables are the output of the preprocessing phase, and they require $M = M'T'$ words of memory, and a total of $\hat{T} = N$ computation time.

During the online phase, we request the encryption of P under the unknown key K, $E_K(P)$. In order to recover K, we try to invert it using each of the Hellman tables by iteratively calculating $h^{[i]}$ starting form $E_K(P)$, and searching if the current value is an endpoint in the table. Once we reach an endpoint, we obtain its startpoint, and continue the evaluation, hoping to reach $(h^{[i]})^{-1}(E_k(P)+i)$ and to recover K. This process has a time complexity of about T' for each Hellman table. Thus, the total online time complexity is $T = T'^2$, and as $M = M'T'$ and $M'T'^2 = N$, we obtain a time-memory tradeoff of $TM^2 = N^2$, or $T = 2^{2(\kappa - m)}$.

Reducing memory access using distinguished points. A simple variant of Hellman's algorithm (attributed to Ron Rivest, and later analyzed in [8, 21]), stops each chain once it reaches a set of *distinguished points*, which are defined according to an easily verifiable condition on $h^{[i]}(K)$. For example, in case we require chains of length T', we define the set of distinguished points to contain the points whose $log(T')$ LSBs are zero. With this variant, the length of the chains is variable and is only defined on average, but this does not result in a significant penalty on the theoretical time complexity of the attack. On the other hand, the distinguished points method has a big advantage in practice, as we only need to access a large Hellman table once for (about) every T' evaluations of $h^{[i]}$ in the online phase of the attack. The small number of memory accesses allows the attacker to store the memory on hard disk, which is much cheaper than RAM.

Parallelization. Since each of the T' Hellman tables can be searched independently, the computation can be divided across (at most) T' CPUs, each requiring access to a Hellman table of size M'. Furthermore, each CPU is expected to access the memory only once during the computation of T' time (in order to find a startpoint that corresponds to an endpoint). Consequently, time-memory tradeoff algorithms can be implemented relatively cheaply on dedicated hardware [18, 21].

For example, in case $\kappa = 64$ and we have $M = 2^{48}$ words of memory, then the online algorithm requires $T = 2^{2(64-48)} = 2^{32}$ time. This computation can

[8] We note that we can save a large fraction of the memory required for startpoint storage by exploiting the freedom to choose them, as described in [2]. Thus, we assume that a chain requires a single word of storage.

be divided across $T' = N/M = 2^{16}$ processors, each performing 2^{16} operations, and requiring (a single) access to $M' = M/T' = 2^{32}$ memory (i.e., 32 gigabytes).

4.2 Parallel Collision Search [23]

The parallel collision search algorithm was published by van Oorschot and Wiener [23], reducing the memory required for finding collisions in an n-bit function F (compared to trivial algorithms). Given $M = 2^m$ words of memory, the algorithm builds a chain structure which is similar to a Hellman table. The chain structure contains 2^m chains, where each chain starts at an arbitrary point and is terminated at a distinguished point (stored in memory) such that its average length is $T' = 2^{(n-m)/2}$ (i.e., the distinguished point set is of size N/T'). As in the case of a Hellman table, since $T' \cdot T'M = N$, then every chain is expected to collide with about one other chain in the structure. Thus, the structure contains about M' collisions which can be recovered efficiently as shown in [23]. However, as our attacks only use a degenerated variant of this algorithm, this short description suffices in order to understand the rest of this paper.

4.3 A Basic Time-Memory-Data Tradeoff Attack on the FX-Construction [17] (without Preprocessing)

We describe the basic and well-known chosen plaintext attack of the FX-construction [17], which is based on the attack of Daemen on the Even-Manour scheme [11]. We also mention the known plaintext attack with the same complexity on the scheme (see [12]), but we do not describe it here, as it is less relevant for this paper. On the other hand, the ideas and notation introduced in this simple attack will be repeatedly used throughout the rest of this paper.

The attack on the FX-construction is based on encrypting plaintexts P_i, and independently evaluating the core function F_K with values of K and X_j, while looking for an (i, j) pair such that $P_i \oplus K_1 = X_j$. In order to detect such a collision efficiently, we cancel the effect of the masking keys by defining functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$ such that $P_i \oplus K_1 = X_j$ implies that $\phi_1(P_i) = \phi_2(K, X_j)$. These functions enable us to efficiently filter the (i, j) candidates.

For a general FX-construction, we pre-fix an arbitrary value $\Delta \neq 0$, set $P'_i \triangleq P_i \oplus \Delta$ and $X'_j \triangleq X_j \oplus \Delta$, and define:

$$\phi_1^{FX}(P_i) \triangleq C_i \oplus C'_i = FX_{K,K_1,K_2}(P_i) \oplus FX_{K,K_1,K_2}(P_i \oplus \Delta)$$

$$\phi_2^{FX}(K, X_j) \triangleq Y_j \oplus Y'_j = F_K(X_j) \oplus F_K(X_j \oplus \Delta).$$

Thus, $P_i \oplus K_1 = X_j$ implies that $\phi_1^{FX}(P_i) = FX_{K,K_1,K_2}(P_i) \oplus FX_{K,K_1,K_2}(P_i \oplus \Delta) = K_2 \oplus F_K(K_1 \oplus P_i) \oplus K_2 \oplus F_K(K_1 \oplus P_i \oplus \Delta) = F_K(X_j) \oplus F_K(X_j \oplus \Delta) = \phi_2^{FX}(K, X_j)$ as required. Note that each collision gives candidates for the full key $(K, K_1 = P_i \oplus X_j, K_2 = C_i \oplus Y_j)$, which can be easily tested using trial encryptions.

For an SFX-construction in which $K_2 = A(K_1)$ (such as PRINCE and PRIDE), the functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$ can be simplified to use single pairs of (P_i, C_i) and (X_j, Y_j), respectively. Formally, we define:

$$\phi_1^{SFX}(P_i) \triangleq A(P_i) \oplus C_i = A(P_i) \oplus FX_{K,K_1,K_2}(P_i)$$

$$\phi_2^{SFX}(K, X_j) \triangleq A(X_j) \oplus Y_j = A(X_j) \oplus F_K(X_j).$$

Thus $P_i \oplus K_1 = X_j$ implies that $\phi_1^{SFX}(P_i) = A(P_i) \oplus FX_{K,K_1,K_2}(P_i) = A(P_i) \oplus K_2 \oplus F_K(K_1 \oplus P_i) = A(P_i \oplus K_1) \oplus F_K(K_1 \oplus P_i) = A(X_j) \oplus F_K(X_j) = \phi_2^{SFX}(K, X_j)$ as required.

The details of the attack are given in Appendix A. It has a memory complexity of D and an expected time complexity of $max(2D, 2^{\kappa+n-d+1})$ on FX-constructions. For SFX-constructions, the data and time complexities of the attack are reduced by a factor of 2.

4.4 Time-Memory-Data Tradeoff attacks on Even-Mansour [14] (with Preprocessing)

We now summarize the time-memory-data tradeoff by Fouque et al. on the Even-Mansour scheme, which uses a one-time preprocessing phase. The main idea of the attack is to request plaintexts P_i, and evaluate the permutation F with values X_j such that a collision $P_i \oplus K_1 = X_j$ can be efficiently detected. In order to do so, we request the data P_i and evaluate values X_j in the form of chains, as in the parallel collision search algorithm [23].

Although we cannot immediately detect that $P_i \oplus K_1 = X_j$, the main observation of Fouque et al. is that we can independently add to P_i and X_j the same value $\phi_1(P_i) = \phi_2(X_j)$ (for the functions ϕ_1, ϕ_2 defined above for the FX and SFX constructions, where the key of the core cipher K is simply be ignored), which guarantees that in case $P_i \oplus K_1 = X_j$, then $P_{i+1} \oplus K_1 = X_{j+1}$.[9] Hence, the functions we iterate are defined as

$$P_{i+1} \triangleq \Phi_1(P_i) \triangleq P_i \oplus \phi_1(P_i)$$

$$X_{j+1} \triangleq \Phi_2(X_j) \triangleq X_j \oplus \phi_2(-, X_j).$$

We note that the downside of this approach is that the attack becomes an adaptively-chosen plaintext attack, as $P_{i+1} = P_i \oplus \phi_1(P_i)$ depends on C_i and cannot be computed in advance (both for general FX-constructions and SFX-constructions).

The details of the attack are described in Appendix B. The time and data complexities of the online phase of the attack are both about $T = D = 2^d$ for SFX-constructions and 2^{d+1} for FX-constructions. Its memory complexity is $M = 2^{n-2d}$, and it requires preprocessing time of $\hat{T} = 2^{n-d}$ for SFX-constructions and $2N/D = 2^{n-d+1}$ for FX-constructions.

[9] The paper of [14] refers to this situation as the chains becoming parallel.

4.5 Time-Memory-Data Tradeoff attacks on the FX-Construction [14] (with Preprocessing)

As described in [14], we can easily generalize the previous attack on Even-Mansour to FX-constructions in which the internal permutation is keyed. During preprocessing, we simply iterate over all the 2^{κ} keys of the internal permutation, and store in memory 2^{n-2d} distinguished points for each key. In total, we have $M = 2^{\kappa} \cdot 2^{n-2d} = 2^{\kappa+n-2d}$ and the preprocessing time is about $\hat{T} = 2^{\kappa+n-d}$ for SFX-constructions and $\hat{T} = 2^{\kappa+n-d+1}$ for FX-constructions. The online phase of the attack is essentially the same as in the previous attack, i.e., $T = D = 2^{d}$ for SFX-constructions and 2^{d+1} for FX-constructions.

5 New Time-Memory-Data Tradeoff Attacks on the FX-Construction without Preprocessing

In this section, we describe our new time-memory-data tradeoff attacks on the FX-construction, without using a preprocessing phase. The attacks are described in the most general form, i.e., they are applicable to general FX-constructions (exploiting the general definitions of ϕ_1, ϕ_2, given in Section 4.3). However, as this paper focuses on the concrete SFX-constructions PRINCE and PRIDE, we directly analyze only the variants of the attacks which are optimized for SFX-constructions (i.e, assuming $\phi_1 = \phi_1^{SFX}, \phi_2 = \phi_2^{SFX}$). In order to calculate the complexity parameters for general FX-constructions, we simply multiply the data and time complexities of the attack by a factor of 2, as in the attacks of sections 4.3, 4.4 and 4.5.

5.1 The Case of $D \leq 2^{n/2}$

The attack for the case of $D \leq 2^{n/2}$ can be considered as a straightforward extension of the attack of [14] on the FX-construction (described in Section 4.5). However, [14] focused on attacks with preprocessing on the FX-Construction, and attacks without preprocessing were not described.

We extend the iteration function Φ_2 (defined in the Even-Mansour attack of the FX-construction in Section 4.4) by adding the key of the core cipher as a parameter. The iteration functions are now defined[10] as

$$P_{i+1} \triangleq \Phi_1(P_i) \triangleq P_i \oplus \phi_1(P_i)$$

$$X_{j+1} \triangleq \Phi_2(K, X_j) \triangleq X_j \oplus \phi_2(K, X_j).$$

The attack is described below:

1. Build a chain of plaintexts, starting from an arbitrary plaintext, extended using the iteration function $P_{i+1} = \Phi_1(P_i)$, and terminated at a distinguished point $\hat{P}$ for which the $\log(D)$ LSBs of $\phi_1(\hat{P})$ are 0 (as in the attack of Appendix B). Store the endpoint $\hat{P}$, and its value $\phi_1(\hat{P})$.

[10] Where ϕ_1 and ϕ_2 are defined in Section 4.3.

2. For each possible value of K:
   (a) For N/D^2 different starting points X_0:
      i. Build a chain starting from X_0, defined according to $X_{j+1} = \Phi_2(K, X_j)$, and terminated at a distinguished point $\hat{X}$ for which the $\log(D)$ LSBs of $\phi_2(K, \hat{X})$ are 0. If $\phi_2(K, \hat{X}) = \phi_1(\hat{P})$, test the full key K, K_1, K_2 derived from $\hat{P}$ and $\hat{X}$ using a trial encryption.

For the correct value of K in Step 2, we expect a collision between a node in the online chain and the (expected number of) N/D nodes evaluated offline. As this collision causes the corresponding chains to merge, it will be detected at the next distinguished point, allowing to recover the key.

The expected data complexity of the attack is $D = 2^d$, the expected time complexity is $2^{\kappa+n-d}$, while the memory complexity is negligible. Consequently, we obtain (almost) the same data and time complexities as the attack described in Section 4.3, but (almost) completely nullify the memory complexity (assuming that we can obtain $D \leq 2^{n/2}$ adaptively chosen plaintexts).

5.2 The Case of $D > 2^{n/2}$

The attack for $D \leq 2^{n/2}$ needs to be adapted for the case of $D > 2^{n/2}$, as the online chain (built is Step 1 of the previous attack) is expected to cycle (i.e., collide with it self) after about $2^{n/2}$ evaluations, and thus cannot cover more than $2^{n/2}$ nodes. Thus, we build the structure of chains online, while evaluating offline only one chain per key.

We note that while this attack can also be viewed as an extension of the attacks of [14], it is less straightforward, as all the attacks of [14] use $D \leq 2^{n/2}$. The reason is that the attacks of [14] are based on the basic Even-Mansour attack (described is 4.4), for which $\kappa = 0$ and there is no gain in using $D > 2^{n/2}$ (as this increases the total time complexity of the attack). On the other hand, we observe that for FX-constructions, we can indeed benefit from $D > 2^{n/2}$.

1. For D^2/N different starting points P_0:
   (a) Build a chain of plaintexts, starting from P_0, extended using the formula $P_{i+1} = \Phi_1(P_i)$, and terminated at a distinguished point $\hat{P}$ for which the $\log(N/D)$ LSBs of $\phi_1(\hat{P})$ are 0. Store the endpoint $\hat{P}$ in a list L, sorted according to $\phi_1(\hat{P})$.
2. For each possible value of K:
   (a) Build a chain starting from an arbitrary value X_0, defined according to $X_{j+1} = \Phi_2(K, X_j)$, and terminated at a distinguished point $\hat{X}$ for which the $\log(N/D)$ LSBs of $\phi_2(K, \hat{X})$ are 0. Search for $\phi_2(K, \hat{X})$ in the list L and for each match with some $\phi_1(\hat{P})$, recover $\hat{P}$, obtain a suggestion for the full key K, K_1, K_2 and test it using a trial encryption.

The chain structure of plaintexts covers $D^2/N \cdot N/D = D$ nodes on average, and it is expected to collide with the chain of values (of expected length N/D) for the correct K. The data complexity of the attack is $D = 2^d$, its time complexity is $T = max(2^d, 2^{\kappa+n-d}) = 2^{\kappa+n-d}$ (assuming $\kappa \geq n$), while its memory complexity is $M = D^2/N = 2^{2d-n}$. Thus, we obtain (about) the same time complexity as the basic attack of Section 4.3, but gain a factor of $2^d/(2^{2d-n}) = 2^{n-d}$ in memory.

6 New Time-Memory-Data Tradeoff Attacks on the FX-Construction with Preprocessing

In this section, we describe our new time-memory-data tradeoff attacks on the FX-construction, taking advantage of a preprocessing phase. As in Section 5, we directly analyze only the attack variants which are optimized for SFX-constructions (although the attacks are described in the most general form). For general FX-constructions, we simply multiply the data and time complexities of the attacks by a factor of 2.

We note that it is possible to apply standard time-memory-data tradeoffs for stream ciphers [3, 4] to the FX-construction in the chosen plaintext model (and to some extent, also in the known plaintext model). However, the most interesting tradeoffs are obtained in the adaptively chosen plaintext model, in which we combine the attacks of the previous section with techniques borrowed from stream cipher cryptanalysis.

6.1 The Case of $D \leq 2^{n/2}$

In the setting of $D \leq 2^{n/2}$, the online time complexity of the previous attack of [14] (described in Section 4.5) is only $T = 2^d$. However, the main drawback of the previous attack is its high memory complexity of (at least) 2^κ. This memory complexity is complectly impractical for standard values of $\kappa \geq 64$. We now show how to reduce the memory complexity of the algorithm, trading it off with the online time complexity. Our main idea is to use Hellman's time-memory tradeoff algorithm in order to cover during the preprocessing phase of the attack, the $2^{\kappa+n-2d}$ pairs of (key,distinguished point).

The idea of using Hellman's time-memory tradeoff algorithm to cover special points was first published in cryptanalysis of a certain type of stream ciphers,[11] and we now show how to adapt it to the FX-construction. In order to cover the pairs of (key,distinguished point), we define a mapping between the $2^{\kappa+n-2d}$ pairs, denoted by $h_2(K, \hat{X})$. One problem that we need to overcome is that $\hat{X}$ is an n-bit word, and does not contain the sufficient $\kappa + n - 2d \geq n$ bits[12] in order to define this mapping. Furthermore, the mapping cannot directly depend on K, as in the online phase we search the Hellman tables without knowledge of the online key. Thus, in order to collect more data, we simply continue evaluating the chain by applying Φ_2 to $(K, \hat{X})$ sufficiently many times, until we collect the $\kappa + n - 2d$ bits required in order to define the *Hellman value* of $(K, \hat{X})$. This value will be used in order to determine the next (key,distinguished point) pair, i.e., the output of $h_2(K, \hat{X})$. The algorithm of the Hellman mapping $h_2(K, \hat{X})$ is given below, assuming that $\kappa = n$ for the sake of simplicity. We note that in cases where $\kappa > n$, we simply apply Φ_2 more times in order to collect more data (the case of $\kappa < n$ can be handled by truncation).

[11] Refer to tradeoffs for stream ciphers with low sampling resistance [3, 4].

[12] We consider $\kappa = n$ and $d \leq n/2$, and thus $\kappa + n - 2d \geq n$.

1. Compute the $2n$-bit Hellman value of $(K, \hat{X})$ by first computing the next 2 points in the chain $X' = \Phi_2(K, \hat{X})$ and $X'' = \Phi_2(K, X')$. The Hellman value of $(K, \hat{X})$ is defined as $(\phi_2(X'), \phi_2(X''))$.
2. Interpret the Hellman value as $(Z, K^{next}) = (\phi_2(X'), \phi_2(X''))$ (note that both Z and K^{next} are n-bit words). Compute a chain of (average) length $D = 2^d$, using the iteration function $\Phi_2(K^{next}, X)$, starting from $X = Z$, and terminating at a distinguished point $(K^{next}, \hat{Z}^{next})$ (i.e., such that the $\log(D)$ LSBs of $\phi_2(\hat{Z}^{next})$ are zero). Output $h_2(K, \hat{X}) = (K^{next}, \hat{Z}^{next})$.

Once the mapping $h_2(K, \hat{X})$ is well-defined, we can use Hellman's preprocessing algorithm to cover a space of $2^{\kappa+n-2d}$ points (pairs) $(K, \hat{X})$. As the average time complexity of one application of $h_2(K, \hat{X})$ is D, the total time complexity of the preprocessing phase is $\hat{T} = D \cdot 2^{\kappa+n}/D^2 = 2^{\kappa+n-d}$. Since h_2 is defined on (at most) $2n$ bits, we can store $M/2$ chains with M words of memory. However, in case $D \approx 2^{n/2}$ and $\kappa \leq n$, we essentially need to cover a space of at most 2^n (we cover one distinguished point per key on average), and thus we can store a larger number of M chains with M words of memory.

We now point out a few technical issues about the preprocessing algorithm: since we are using Hellman's algorithm to cover $2^{\kappa+n-2d}$ points, the (average) length of the Hellman chains is $2^{\kappa+n-2d-m}$ (determined according to the available memory $M = 2^m$). In order to terminate a Hellman chain (computed using h_2 on the space of $\kappa + n - 2d$ bits), we need to define a subset of "Hellman distinguished points", containing pairs of $(K, \hat{X})$. Such a subset (which determines when to terminate an iteration chain of h_2) can be defined (for example) according to the LSBs of the Hellman value $(\phi_2(X'), \phi_2(X''))$, computed in Step 1 of the algorithm above. The "Hellman distinguished points" should be contrasted with the distinguished points defined for the iteration on the n-bit space with a fixed key using Φ_2 (such distinguished points are defined according to the LSBs of $\phi_2(\hat{X})$). In order to avoid confusion, we refer to chains and distinguished points computed using h_2 as *Hellman chains* and *Hellman distinguished points*, whereas the ones computed using Φ_1 and Φ_2 are simply referred to as (standard) chains and distinguished points.[13] An additional technical issue is that in order to cover the full space of $2^{\kappa+n-2d}$ points, we need to define flavors of h_2 (namely, $h_2^{[i]}$), and this can be done (for example) by defining $h_2^{[i]} = h_2(K, \hat{X}) + (i, i)$.

The online algorithm is given below.

1. Compute a chain of (approximately) D points using the iteration function ϕ_1, starting from an arbitrary plaintext, and terminating at a distinguished point $\hat{P}$, where the $log(D)$ LSBs of $\phi_1(\hat{P})$ are 0.
2. Given $\hat{P}$, compute the corresponding Hellman value $(\phi_1(P'), \phi_1(P''))$ similarly to the preprocessing phase, by computing the next 2 points in the chain $P' = \Phi_1(\hat{P})$ and $P'' = \Phi_1(P')$.
3. Invert $(K, X) = (\phi_1(P'), \phi_1(P''))$ using the Hellman tables, obtain a suggestion for the full key (K, K_1, K_2) and test it.

[13] Our definitions are related to the definitions of *full name*,*output name*, and *short name* in the context of stream ciphers with low sampling resistance [3].

The data complexity of the attack is D, and according to Hellman's time-memory tradeoff curve, its average time complexity is $T' = (N'/M')^2$ evaluations of h_2, where $N' = 2^{\kappa+n-2d}$ is the size of the covered space and $M' = M/2 = 2^{m-1}$ is the number of Hellman chains stored in memory. Thus $T' = (2^{\kappa+n-2d}/2^{m-1})^2 = 2^{2(\kappa+n-m-2d+1)}$, and since each evaluation of h_2 requires $D = 2^d$ time, then $T = 2^{2(\kappa+n-m-1.5d+1)}$. When $D \approx 2^{n/2}$ and $\kappa \leq n$, we can use the M memory words more efficiently and obtain a (slightly) improved time complexity of $T = 2^{2(\kappa+n-m-1.5d)}$.

Implementation for $\boldsymbol{n = \kappa = 64}$ In the case of PRINCE and PRIDE, then $n = \kappa = 64$. We assume that we have $2^m = 2^{48}$ words of memory (2^{51} bytes) and we can obtain $2^d = 2^{32}$ adaptively chosen plaintexts. In total, the online time complexity of the algorithm is $2^{2(64+64-48-48)} = 2^{64}$, corresponding to Attack 1 in Table 1. Similarly to the example given at the end of Section 4.1, as the Hellman chains cover a space of size $2^\kappa = 2^{64}$ points, the 2^{64} computation can be divided across 2^{16} CPUs, each requiring (a single) access to a memory of 2^{32} words (and it can thus be stored on a hard disk). Each CPU invokes h_2 about 2^{16} times, where each invocation requires 2^{32} cipher evaluations. Thus, each CPU performs about 2^{48} cipher evaluations in total.

6.2 The Case of $D > 2^{n/2}$

As in the case considered in Section 5.2, we need to adapt the previous attack to efficiently exploit $D > 2^{n/2}$ data. Similarly to the case of $D \leq 2^{n/2}$, our techniques are related to those of [3, 4]. However, the method in which we request the data and optimize the parameters in this setting are different from the stream cipher setting (where the method in which the keystream is obtained seems to give the attacker no advantage).

We consider two different adaptation methods to the previous attack of $D \leq 2^{n/2}$. In the first method, we obtain the data using chains of (maximal) length $2^{n/2}$. In order to ensure that these chains do not merge, we define flavors of Φ_1 and Φ_2 (which should be contrasted with the Hellman flavors of h_2). Thus, we define $2^{d-n/2}$ flavors, and build Hellman tables for each one. During the online phase, we obtain one distinguished point per flavor and search it in the corresponding Hellman table. One can observe that in terms of the time-memory tradeoff, the flavors of Φ_1 and Φ_2 play a similar role to the flavors of h_2 in the attack with $d = n/2$. Consequently, we obtain the same time-memory tradeoff as for $d = n/2$, i.e., $T = 2^{2(\kappa+n/4-m)}$.[14] On the other hand, the preprocessing complexity is reduced to $\hat{T} = 2^{\kappa+n-d}$.

An Improved Tradeoff The attack above is a direct extension of the tradeoff obtained for $D = 2^{n/2}$, which covered offline, 2^κ distinguished points using h_2. We now show how to obtain an improved attack, using a simple and yet subtle

[14] We also need to add the restriction that $T \geq D$.

and non-trivial observation. We notice that for $D > 2^{n/2}$, we can use chains of length $N/D < 2^{n/2}$. These chains are shorter than the ones used for $D = 2^{n/2}$ data, and are of the same length as in the attack with only $2^{d'} = 2^{n-d}$ data. At first, using shorter chains seems inefficient, since the number of chains is increased and we need to search for more distinguished points online, increasing the time complexity. One the other hand, we show that we can use the memory more efficiently than in the previous attack, and this is crucial, as in the time-memory tradeoff formula TM^2, the memory is more dominant than the time.

In order to understand the gain in memory, we first compare the cases of $2^d > 2^{n/2}$ data and $2^{d'} = 2^{n-d}$ data, which use the same chain length. The difference is that in the case of $2^{d'} = 2^{n-d}$ data, we obtained only one distinguished point online, whereas now we obtain $2^{d-(n-d)} = 2^{2d-n}$ such distinguished points, and we need to cover only one of them offline in order to succeed. Thus, in the attack with $2^{d'} = 2^{n-d}$ data, we had to cover offline the large space of $2^{\kappa+n-2d'}$ distinguished points using Hellman tables, whereas now we need to cover only $2^{\kappa+n-2d'-(2n-d)} = 2^{\kappa+n-2(n-d)-(2n-d)} = 2^{\kappa}$ distinguished points. Namely, we need to cover about one distinguished point for every key K of the core cipher, as in the attack above. However, the crucial observation is that the space of distinguished points is of the same size as in the attack with $2^{d'} = 2^{n-d}$ data. This space is larger than the space for $D = 2^{n/2}$, implying that we can build larger Hellman tables and use the memory more efficiently compared to the (non optimal) attack above (which is a direct extension of the case of $D = 2^{n/2}$).

A simple way to compute the tradeoff is to reuse the formula $T = 2^{2(\kappa+n-m'-1.5d')}$, obtained for the attack with $d' \leq n/2$, and to plug in $d' = n - d$ and $m' = m - (2d - n)$ (the number of Hellman tables is reduced by a factor of $2d - n$, which is the number of distinguished points obtained online). However, this tradeoff has some limitations and requires additional analysis for some parameter values. In order to better understand the full attack, we redo the low-level computation.

For parameters T' and M', we build a Hellman table of distinguished points (using the function h_2) with the stopping rule of $T' \cdot T'M' = 2^{\kappa+n-2d'} = 2^{\kappa-n+2d}$ (after which the Hellman chains start colliding extensively).[15] We need to cover 2^{κ} distinguished points with H Hellman tables, namely $HT'M' = 2^{\kappa}$. Since each evaluation of h_2 requires 2^{n-d} time, the preprocessing time complexity is $\hat{T} = 2^{\kappa+n-d}$ (as in the non optimized attack above). The total memory complexity of the attack is $M = HM'$ and the total online time complexity is calculated as follows: searching a single Hellman table requires T' evaluations of h_2, i.e., a total of $T' \cdot 2^{n-d}$ time. For each of the 2^{2d-n} distinguished points, we need to search the H Hellman tables, and thus the total online time complexity is $T = T' \cdot 2^{n-d} \cdot H \cdot 2^{2d-n} = T'H \cdot 2^d$.

We calculate the tradeoff by evaluating $T \cdot M^2 = T'H \cdot 2^d \cdot (HM')^2 = 2^d \cdot H^3 \cdot T' \cdot (M')^2$. As $HT'M' = 2^{\kappa}$, we obtain $T \cdot M^2 = 2^{\kappa+d} \cdot H^2 \cdot M'$. Furthermore, since $HT'M' = 2^{\kappa}$ and $(T')^2 \cdot M' = 2^{\kappa-n+2d}$, then $H^2 \cdot M' = 2^{2\kappa-(\kappa-n+2d)} = 2^{\kappa+n-2d}$. Thus, $T \cdot M^2 = 2^{\kappa+d+\kappa+n-2d} = 2^{2\kappa+n-d}$, namely, we obtain the tradeoff $T =$

[15] Note that the stopping rule in the previous attack was $T' \cdot T'M' = 2^{\kappa} < 2^{\kappa-n+2d}$.

$2^{2(\kappa+n/2-m-d/2)}$. This tradeoff efficiently exploits more than $2^{n/2}$ data, unlike the previous tradeoff $T = 2^{2(\kappa+n/4-m)}$.

A condition that we have to impose on the tradeoff above is that the number of Hellman tables is at least 1, i.e., $H \geq 1$. In order to calculate H, we recall that $H^2 \cdot M' = 2^{\kappa+n-2d}$, and as $2^m = HM'$, we obtain $H = 2^{\kappa+n-2d-m}$. Since $H \geq 1$, the tradeoff above is valid only for $m \leq \kappa + n - 2d$.

When we want to utilize 2^m memory for $m > \kappa + n - 2d$, then we use only one Hellman table (i.e., $H = 1$), and we stop the Hellman chains before the stopping rule $(T')^2 \cdot M' < 2^{\kappa-n+2d}$. Namely, we have $M' = M = 2^m$ and $T'M' = 2^\kappa$, implying that $T' = 2^{\kappa-m}$ and $T = T' \cdot 2^d = 2^{\kappa+d-m}$. Note that $(T')^2 \cdot M' = 2^{2(\kappa-m)} \cdot 2^m = 2^{2\kappa-m} < 2^{2\kappa-(\kappa+n-2d)} = 2^{\kappa-n+2d}$, so indeed we do not violate the stopping rule.

Finally, we observe that a similar restriction on m also applies to the previous tradeoff $T = 2^{2(\kappa+n/4-m)}$, and it is possible to show that the tradeoff obtained here is always at least as efficient as the previous one.

Implementation for $n = \kappa = 64$ We assume that we have $2^m = 2^{48}$ words of memory and we can obtain $2^d = 2^{40}$ adaptively chosen plaintexts. In total, the online time complexity of the algorithm is $2^{2(64+32-48-20)} = 2^{56}$, corresponding to Attack 2 in Table 1. In this case $H = 2^{\kappa+n-2d-m} = 1$, i.e., we have a single Hellman table. As we search the table with $2^{2d-n} = 2^{16}$ distinguished points, the 2^{56} computations can be divided across (up to) 2^{16} CPUs, each performing $2^{56-16} = 2^{40}$ computations, and accessing the memory only once (and it can therefore be stored on a hard disk).

7 Tweaking PRINCE and PRIDE

In this section, we tweak the SFX-constructions PRINCE [7] and PRIDE [1], such that they resist the attacks described in this paper. As our attacks are generic, we need to make sure that the tweaked block ciphers are no longer FX-constructions. Hence, we modify the key schedule of the core ciphers, such that they depend of the masking key K_1. Additionally, we need to make sure that the tweaked ciphers maintain their original design goals. In particular, we need to assure that our tweak is sufficiently simple and does not result in a (significant) performance penalty.

We stress that we do not claim that the tweaked ciphers have ideal security of (about) 128 bits. If fact, recent cryptanalytic results on round-reduced variants of these ciphers [9, 24] suggest that this may not be the case. However, we still believe that our tweaks ensure a significantly larger security margin against practical attacks.

PRIDE PRIDE is a software-optimized cipher and its design goals do not place any specific restriction on its internal key schedule (except for efficiency in software). The key schedule of PRIDE XORs to the state 20 round-keys $K^{[i]}$ (for

$i \in \{0, 1, \ldots, 19\}$) such that $K^{[i]} = L^{[i]}(K)$ and the $L^{[i]}$ functions are simple linear mappings. There are several ways to incorporate K_1 into the key schedule such that efficiency is maintained. Here, we propose a key schedule which alternates K and K_1, i.e., we set $K^{[2i+1]} = L^{[2i+1]}(K)$ and $K^{[2i]} = L^{[2i]}(K_1)$, for $i \in \{0, 1, \ldots, 9\}$.

PRINCE Apart from standard efficiency considerations, a central design goal of the hardware-optimized cipher PRINCE was to realize decryption on top of encryption with a negligible cost. This goal was implemented by ensuring that PRINCE has a specific symmetric property around its middle round, called α-reflection.[16]

The α-reflection property places several restrictions on the internal components of the core cipher. However, since we are only interested in tweaking the key schedule of PRINCE, we are only concerned with the restriction on the round keys. This restriction simply forces the round keys to be symmetric around the middle rounds. More specifically, the core of PRINCE XORs 12 round keys $K^{[i]}$ (for $i \in \{0, 1, \ldots, 11\}$) to the state, and they are restricted by the equalities $K^{[i]} = K^{[11-i]}$. The current specification of PRINCE preserves symmetry and ensures an efficient implementation by using a very simple key schedule that sets all round keys to be equal to the K (i.e., $K^{[i]} = K$).

There are several ways to incorporate K_1 into the key schedule such that symmetry is preserved and efficiency is maintained. Here, similarly the case of PRIDE, we propose a key schedule which alternates K and K_1, i.e., we set $K^{[0]} = K^{[2]} = K^{[4]} = K$ and $K^{[1]} = K^{[3]} = K^{[5]} = K_1$, and for $i \in \{6, 7, \ldots, 11\}$ we simply set $K^{[i]} = K^{[11-i]}$ as required.

8 Conclusions

In this paper, we proposed new generic time-memory-data tradeoffs for FX-constructions, and optimized them for the recent proposals PRINCE and PRIDE. Some of our attacks are surprisingly efficient, and although they require adaptively chosen plaintexts and a currently impractical preprocessing phase, we believe that the mere existence of these attacks demonstrates the small security margin of PRINCE and PRIDE against practical attacks. Finally, we showed that FX-constructions can provide better security with very little overhead by incorporating the masking keys into the key schedule of the core cipher. This questions the use of the FX-construction in new designs.

Acknowledgements: The author would like to thank Orr Dunkelman and Adi Shamir for helpful discussions on this work.

[16] If we denote by F_K the encryption of r rounds in the middle of the cipher under the key K, then the α-reflection property (for a fixed value of α) states that for any input X, $F_K(X) = F^{-1}_{K \oplus \alpha}(X)$.

References

1. Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block Ciphers - Focus On The Linear Layer (feat. PRIDE): Full Version. *IACR Cryptology ePrint Archive*, 2014:453, 2014. Accepted to CRYPTO 2014. To appear in Lecture Notes in Computer Science.
2. Elad Barkan, Eli Biham, and Adi Shamir. Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2006.
3. Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.
4. Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.
5. Alex Biryukov and David Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.
6. Bitcoin network graphs. `http://bitcoin.sipa.be/`.
7. 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, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
8. Johan Borst, Bart Preneel, and Joos Vandewalle. On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation. In *Proceedings of 19th Symposium in Information Theory in the Benelux, WIC*, pages 111–118, 1998.
9. Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. Multiple Differential Cryptanalysis of Round-Reduced PRINCE (Full version). *IACR Cryptology ePrint Archive*, 2014:89, 2014. Presented at 2014. To appear in Lecture Notes in Computer Science.
10. COPACOBANA faqs. `http://www.copacobana.org/faq.html`.
11. Joan Daemen. Limitations of the Even-Mansour Construction. In Imai et al. [16], pages 495–498.
12. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2012.
13. Shimon Even and Yishay Mansour. A Construction of a Cioher From a Single Pseudorandom Permutation. In Imai et al. [16], pages 210–224.
14. Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to Discrete Logs, Even-Mansour and Prince. Cryptology ePrint Archive, Report 2013/761, 2013. `http://eprint.iacr.org/`.
15. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
16. 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*. Springer, 1993.

17. Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 252–267. Springer, 1996.
18. Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking. In Koen Bertels, João M. P. Cardoso, and Stamatis Vassiliadis, editors, *ARC*, volume 3985 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2006.
19. National Institute of Standards and Technology. Recommendation for Key Management – Part 1: General (revision 3). NIST Special Publication 800–57, 2012.
20. Ronald L. Rivest. DESX, 1984. Never Published.
21. François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 593–609. Springer, 2002.
22. The PRINCE Team. The PRINCE Challenge, 2014. `https://www.emsec.rub.de/research/research_startseite/prince-challenge/`.
23. Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.
24. Jingyuan Zhao, Xiaoyun Wang, Meiqin Wang, and Xiaoyang Dong. Differential Analysis on Block Cipher PRIDE. Cryptology ePrint Archive, Report 2014/525, 2014. `http://eprint.iacr.org/`.

A Details of the Basic Time-Memory-Data Attack on the FX-Construction [17] (without Preprocessing)

We give the details of the basic attack using the general functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$, defined in Section 4.3.

1. Obtain the encryptions of D arbitrary values P_i, denoted by C_i. For general FX-constructions, also obtain the D additional encryptions required to calculate $\phi_1^{FX}(P_i)$, by asking for the encryptions of $P'_i = P_i \oplus \Delta$. Store (P_i, C_i) in a list L, sorted according to $\phi_1(P_i)$.
2. For each possible value of K:
   (a) For N/D different values of X_j:
      i. Compute $\phi_2(K, X_j)$ by computing $Y_j = F_K(X_j)$ (for a general FX-construction, also compute $F_K(X_j \oplus \Delta)$).
      ii. Search $\phi_2(K, X_j)$ in L. For each match, retrieve (P_i, C_i), and test each of the key candidate triplet $(K, K_1 = P_i \oplus X_j, K_2 = C_i \oplus Y_j)$ using trial encryptions. If a trial encryption succeeds, return the corresponding full key (otherwise return to Step 2.(a)).

According the the birthday paradox, for the correct value of K in Step 2, we expect a pair (i, j) such that $P_i \oplus K_1 = X_j$. Therefore, we expect to obtain a match in L in Step 2.(a).ii between $\phi_1(P_i)$ and $\phi_2(K, X_j)$ and recover the correct K, K_1, K_2.

For general FX-constructions, the data complexity of the attack is $2D$ chosen plaintexts, and its memory complexity is $M = 2D$ n-bit words, required in order to store L. In order to compute the time complexity, we note that for an arbitrary value of the n bits of $\phi_2(K, X_j)$, we expect at most one match in L (which contains at most 2^n elements). Thus, the expected time complexity of Step 2.(a) is about 2, implying that the expected time complexity of the full attack is $max(2D, 2^{\kappa+n-d+1})$ (i.e., we can efficiently exploit $D \leq 2^{(\kappa+n)/2}$ data). For SFX-constructions, the data and time complexities of the attack are reduced by a factor of 2 (note that in this case, the attack requires only known plaintexts).

B Details of the Time-Memory-Data Tradeoff Attack on Even-Mansour [14] (with Preprocessing)

We assume that we can obtain the encryptions of about $D \leq 2^{n/2}$ adaptively-chosen plaintexts during the online phase. During the preprocessing phase, we use the preprocessing iteration function $\Phi_2(X)$ (defined in Section 4.4) in order to build a structure containing N/D^2 chains. Each chain is evaluated from an arbitrary starting point, and terminated at a distinguished point $\hat{X}$ for which the $\log(D)$ LSBs of $\phi_2(\hat{X})$ are zero. Thus, the average chain length is D, implying that the time complexity of preprocessing is $\hat{T} = N/D = 2^{n-d}$ for SFX-constructions and $2N/D = 2^{n-d+1}$ for FX-constructions. For each chain in the structure, we store in memory only the endpoint[17] $\hat{X}$, and sort the chains according to their values $\phi_2(\hat{X})$. Thus, the memory complexity is about $M = N/D^2 = 2^{n-2d}$.

During the online phase, we evaluate a single chain of (expected) length D, starting for an arbitrary plaintext. The chain is defined according to the online iteration function $\Phi_1(P)$, and is terminated at a distinguished point $\hat{P}$ for which the $\log(D)$ LSBs of $\phi_1(\hat{P})$ are zero. Once a distinguished point is reached, we search for it in the structure, and for each match, we obtain and test the key suggestions for K_1, K_2. Note that unlike Hellman's original attack, we directly recover the key from the distinguished points stored in the structure, without the need to further traverse the chains (and thus we do not need to store any information about their startpoints).

As the structure covers about 2^{n-d} values of X_j and the online chain contains 2^d values of P_i, we expect a collision $P_i \oplus K_1 = X_j$. The collision implies that $\phi_1(P_i) = \phi_2(X_j)$, which causes the two corresponding chains to merge and reach distinguished points with the same value. This distinguished point is recovered in the online phase and allows to recover the key K_1, K_2. Thus, the time and data complexities of the online phase of the attack are both about $T = D = 2^d$ for SFX-constructions and 2^{d+1} for FX-constructions.

[17] The structure is somewhat different from a Hellman table, for which we also store information about the startpoints of the chains.