FAST: Secure and High Performance Format-Preserving Encryption and Tokenization

F. Betül Durak[1], Henning Horst[2], Michael Horst[2], and Serge Vaudenay[3]

[1] Microsoft Research, Redmond, USA
[2] Comforte AG, Germany
[3] EPFL, Lausanne, Switzerland

Abstract. We propose a new construction for format-preserving encryption. Our design provides the flexibility for use in format-preserving encryption (FPE) and for static table-driven tokenization. Our algorithm is a substitution-permutation network based on random Sboxes. Using pseudorandom generators and pseudorandom functions, we prove a strong adaptive security based on the super-pseudorandom permutation assumption of our core design. We obtain empirical parameters to reach this assumption. We suggest parameters for quantum security.
Our design accommodates very small domains, with a radix a from 4 to the Unicode alphabet size and a block length ℓ starting 2. The number of Sbox evaluations per encryption is asymptotically $\ell^{\frac{3}{2}}$, which is also the number of bytes we need to generate using AES in CTR mode for each tweak setup. For instance, we tokenize 10 decimal digits using 29 (parallel) AES computations to be done only once, when the tweak changes.

1 Introduction

Symmetric encryption offers an efficient way to keep data private. However, it is typically only length-preserving in the sense that a plaintext and a ciphertext occupy *exactly* the same space and structure in memory. However, length preservation falls short when we consider non-binary plaintext data such as bank account numbers, driver license numbers, tax ID's and so forth. This limitation can be overcome by format-preserving protection mechanisms. *Format-Preserving Encryption* (FPE) was proposed to encrypt data while retaining the original format and field size of the input data. It can, for instance, encrypt a 16-digit credit card number (or part of it), as a series of digits, and produce a ciphertext which is still 16-digits long. One difficulty with FPE is that the message space can be very small for common fields used in personal data processing scenarios, for example, age, a postal code, names, bank sort codes, subsets of larger fields, and so on. In particular, adversaries may be able to go through the

message space exhaustively. Hence, FPE is strengthened with a *tweak*, following the tweakable encryption paradigm. Contrarily to a *nonce*, a tweak can be reused. Tweakable encryption was formalized by Liskov et al. [29].

Formally speaking, an FPE takes as input a key and a message, as well as parameters specifying the format and the tweak. In standard FPE, the format consists of a radix a and a length ℓ. The message domain has cardinality a^ℓ.

FPE first appeared as a *data-type preserving encryption* [13] and in the form of an *omnicipher* with the Hasty Pudding AES competitor [34]. The term FPE is due to Spies [36]. Rogaway presented a list of FPE schemes [32]. Some constructions are based on cycle walking [11]. The most popular FPEs are based on Feistel networks and have been standardized as FF1 and FF3 [1,3,7,8,12]. A weakness was found in FF1 and FF3 by Bellare et al. [6]. FF3 was broken and repaired by Durak and Vaudenay [18]. The attack was later improved by Hoang et al. [23,24]. There exist some dedicated constructions based on substitution-permutation network (SPN) such as TOY100 [21] and DEAN18 [5], but they are designed only for fixed blocks of decimal digits. Actually, one difficulty with SPN-based FPE is that the internal Sboxes must be adapted to the specific format of the input data. Another SPN-based construction allows more formats but suffers from lack of flexibility as well [15]. Another construction mixes the cycle walking techniques with SPN based on Sboxes working on a domain which is larger than the format [16]. However, this construction is not a pure SPN. It is rather based on one-time-pad with a keystream generated from an SPN. Hence, it needs a nonce and it has trivial chosen ciphertext decryption attacks.[4] We believe that substitution-permutation networks has been under-explored for FPE so far and will present a new FPE design based on an SPN.

Some cipher designs use pseudorandom Sboxes and use them in a pseudorandom sequence. Both random selections can be derived from the secret key. This approach was used in LUCIFER [20] (the preliminary version of DES), where Sboxes are invertible 4-bit to 4-bit functions selected from a pool of two using a binary key for each Sbox. It was used in KHUFU [31] as well, where Sboxes are pseudorandom 8-bit to 32-bit functions generated from the secret key. BLOWFISH [33] also used pseudorandom 8-bit to 32-bit functions. In our design, we use pseudorandom

[4] Note that we want exact format preservation hence no stretch in the ciphertext. Consequently, FPE cannot authenticate at the same time and authentication cannot be used to defeat chosen ciphertext attacks.

permutations over an alphabet set $\mathbf{Z}_a$, where a is the radix of the message to be encrypted. The key is used to select a sequence of Sboxes from a pool. The pool of Sboxes is generated by a secret too.

Tokenization is another concept for format-preserving data protection that introduces the notion of mapping cleartext values to substitute "token" values that retain format and structure of the original data, but no cleartext data, while logically isolating the process that performs the mapping. While encryption itself makes no assumption about key ownership, tokenization typically implies that tokenization secret(s) and mapping process are owned by a tokenization system, a strongly isolated single entity authenticating and auditing access to the token mapping process using the tokenization secret(s). ANSI X9.119-2 [2] defines three main approaches for tokenization: (1) On Demand Random Assignment (ODRA) which generates random tokens on demand and stores the association with the plaintext value in a dynamic mapping table which grows per new token generated. However, using the method, the large and constantly growing mapping table creates severe operational issues for environments requiring high performance and resilience. (2) Static table-driven tokenization (a.k.a. vault-less tokenization) generates tokens using a tokenization mapping process which operates using small pregenerated static random substitution tables used as the tokenization secret. (3) Encryption-based tokenization generates tokens using a suitable FPE or symmetric encryption algorithm where the key serves as tokenization secret. Our design can be used as base for static table-driven tokenization as well for encryption-based tokenization.

Our contribution. In this paper, we design and analyze a new format-preserving protection construction method. We call our method FAST as for *Format-preserving Addition Substitution Transformation.*[5] FAST can be used in FPE or tokenization mode. Tokenization mode differs from FPE mode by having two specific inputs instead of one secret key: pregenerated random Sboxes as stateless table secret, which can be common to several domains, and a key, which is be used for domain separation. The stateless table secret is much more used than a given domain-specific key. Therefore, we consider a security model where the stateless table secret is revealed to the adversary and the rest works like in FPE security. In FPE mode, the pseudorandom Sboxes are generated from the key.

We formally define a strong security model for FPE which is essentially a multi-target chosen format and chosen tweak super-PRP (pseudoran-

[5] There exists another FAST algorithm in the literature which is unrelated [14].

dom permutation) notion. Concretely, we consider adversaries who can choose all parameters, have many targets, choose the plaintexts and the ciphertexts. One challenge is that the encryption domain can be really small. Hence, the adversary could get the complete codebook for some tweaks and even look at their permutation parity. We model security by indistinguishability from an ideal FPE making only even permutations.[6]

We formally prove strong security based on a weak security assumption: that the core design is a super-PRP in a weak model where the adversary uses a single target, a single format, and a single tweak. More precisely, we reduce strong security to this weak security notion. We formally prove this reduction in a tight and quantifiable manner.

The single-instance security of the core design is heuristic. We find (what we believe to be) the best attacks on the core design. We optimize our parameters to reduce the number of Sbox applications down to $\ell^{\frac{3}{2}}$ (instead of ℓ^2). We set the number of rounds to twice the one we can break to have a good safety margin. We also consider quantum security.

We implement our algorithm and show good performance. Concretely, our design needs some AES applications to generate random bytes for each tweak. However, in contrast to the FF1 and FF3 algorithms which use AES as round functions in a Feistel network, our design allows for these generations to be parallelized and thus achieve much higher performance by design. Then, the core encryption needs no AES computation or expensive modular reductions. It is purely an SPN with Sboxes, additions, subtractions, and permutations.

Structure of this paper. We first detail the specifications of FAST in Section 2. In Section 3, we give implementation results. We formally define a security model and we prove strong adaptive security of FAST based on the hypothesis that our core scheme is a super pseudorandom permutation in Section 4. Rationales are given in Section 5. The full version of this article [17] further include the formal proofs and the best known attacks which motivated our parameter choices.

2 Algorithm Specification

In what follows, we use the following integer parameters:

[6] A permutation π is called even if the number of (x, y) pairs such that $x < y$ and $\pi(x) > \pi(y)$ is even.

s: bit-security target
L: bitlength of key K
L_1: bitlength of key K_{SEQ}
L_2: bitlength of key K_S ($L_1 = L_2$)
a: the alphabet size a.k.a. radix ($a = 10$ for decimal digits) ($a \geq 4$)
ℓ: word length of input/output ($\ell \geq 2$)
m: number of Sboxes in the pool ($m = 256$)
n: total number of layers ($r = n/\ell$ rounds of ℓ layers)
w: a branch distance ($0 \leq w \leq \ell - 2$)
w': a second branch distance ($1 \leq w' \leq \ell - w - 1$)

Without loss of generality, the alphabet is $\mathbf{Z}_a = \{0, 1, \ldots, a-1\}$ to which we give the group structure defined by modulo a addition. The "input" is an element of $\mathbf{Z}_a^\ell$. An Sbox is a permutation of $\mathbf{Z}_a$. A pool of Sboxes is a tuple $S = (S_0, \ldots, S_{m-1})$ of m Sboxes.

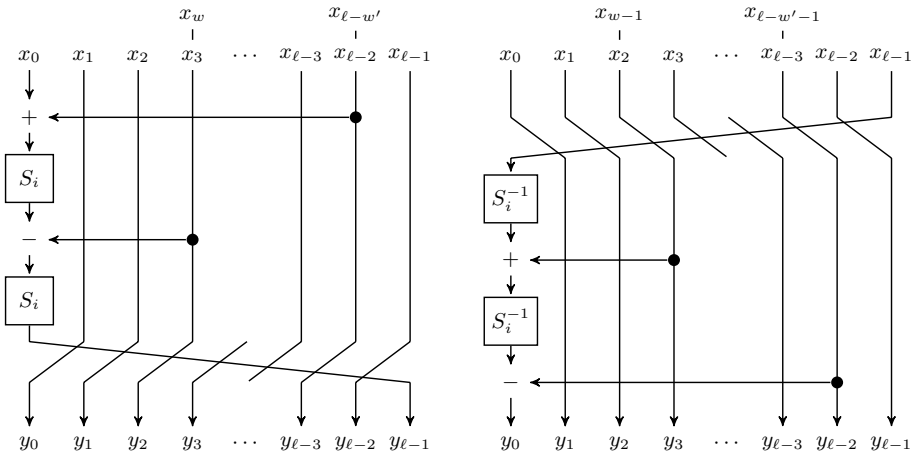


Fig. 1. One Layer with Circular Shift of Branches: Forward (on the left) and Backward (on the right) with $w = 3$, $w' = 2$

Layers of Encryption/Decryption. Given a pool S of m Sboxes and an index i in $\{0, \ldots, m-1\}$, we define the permutation $E_S[i]$ of $\mathbf{Z}_a^\ell$ as follows: for any $x = (x_0, \ldots, x_{\ell-1}) \in \mathbf{Z}_a^\ell$, we have

$$E_S[i](x) = \begin{cases} (x_1, \ldots, x_{\ell-1}, S_i(S_i(x_0 + x_{\ell-w'}))) & \text{if } w = 0 \\ (x_1, \ldots, x_{\ell-1}, S_i(S_i(x_0 + x_{\ell-w'}) - x_w)) & \text{if } w > 0 \end{cases}$$

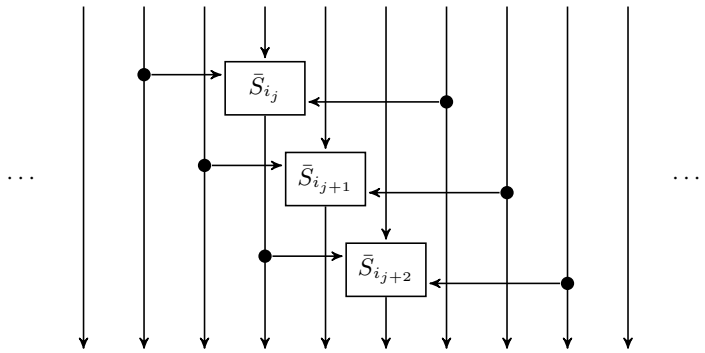


Fig. 2. Three Consecutive Layers with $w = 3$ and $w' = 2$ and without the Circular Shift of Branches (Each $\bar{S}$ represents a double-Sbox)

as depicted in Fig. 1. We call it a *forward layer*. Similarly, we define a *backward layer*

$$D_S[i](x) = \begin{cases} (S_i^{-1}(S_i^{-1}(x_{\ell-1})) - x_{\ell-w'-1}, x_0, \ldots, x_{\ell-2}) & \text{if } w = 0 \\ (S_i^{-1}(S_i^{-1}(x_{\ell-1}) + x_{w-1}) - x_{\ell-w'-1}, x_0, \ldots, x_{\ell-2}) & \text{if } w > 0 \end{cases}$$

If we represent layers without the circular shift of registers, the circuit of three consecutive layers looks like Fig. 2. Each layer involves one register which was modified w' layers before and one register which will be modified w layers later.

Given a sequence $i_0, \ldots, i_{n-1}$ of n indices, we define our m-layer core encryption/decryption functions

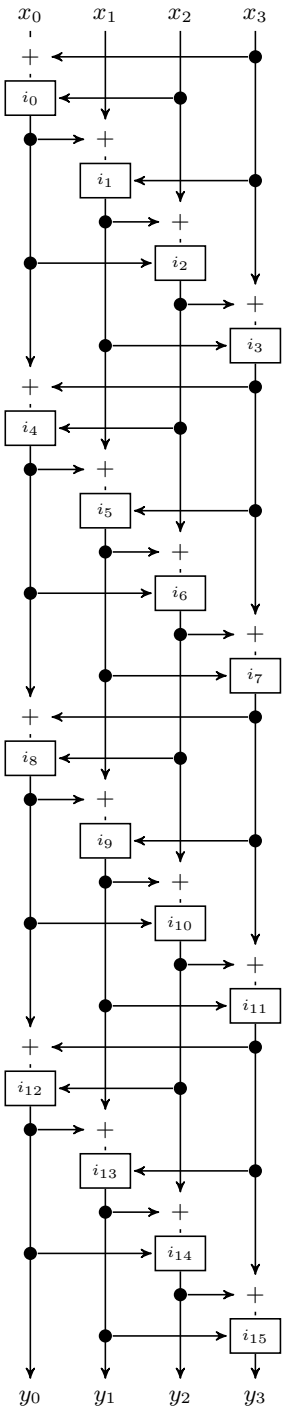
$$\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}] = E_S[i_{n-1}] \circ \cdots \circ E_S[i_0]$$
$$\mathsf{CDec}_S[i_0, \ldots, i_{n-1}] = D_S[i_0] \circ \cdots \circ D_S[i_{n-1}]$$

so that $(\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}])^{-1} = \mathsf{CDec}_S[i_0, \ldots, i_{n-1}]$. The n-layer encryption scheme is depicted in Fig. 3 with $\ell = 4$, $n = 16$, $w = 2$, and $w' = 1$. Since we require n to be multiple of ℓ, we consider n layers as being n/ℓ *rounds* of ℓ layers each.

Sbox Index Sequence Generation. Given an L_1-bit key K_{SEQ}, we generate a sequence $\mathsf{SEQ} = [i_0, i_1, i_2 \ldots, i_{n-1}]$ of n indices in $\{0, 1, \ldots, m-1\}$ using a pseudorandom generator:

$$\mathsf{SEQ} = \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$$

The choice of this function $\mathsf{PRNG}_{1,m,n}$ is open. Our preferred option is the use of AES in CTR mode as later explained.



$\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}](x_0, \ldots, x_{\ell-1})$
1: **for** $j = 0$ to $n-1$ **do**
2: $\quad z \leftarrow S_{i_j}(S_{i_j}(x_0 + x_{\ell-w'} \bmod a) - x_w \bmod a)$
3: $\quad$ **for** $k = 1$ to $\ell - 1$ **do**
4: $\quad\quad x_{k-1} \leftarrow x_k$
5: $\quad$ **end for**
6: $\quad x_{\ell-1} \leftarrow z$
7: **end for**
8: **return** $(x_0, \ldots, x_{\ell-1})$

$\mathsf{CDec}_S[i_0, \ldots, i_{n-1}](y_0, \ldots, y_{\ell-1})$
9: **for** $j = n-1$ down to 0 **do**
10: $\quad z \leftarrow y_{\ell-1}$
11: $\quad$ **for** $k = \ell - 1$ down to 1 **do**
12: $\quad\quad y_k \leftarrow y_{k-1}$
13: $\quad$ **end for**
14: $\quad y_0 \leftarrow S_{i_j}^{-1}(S_{i_j}^{-1}(z) + y_w \bmod a) - y_{\ell-w'} \bmod a$
15: **end for**
16: **return** $(y_0, \ldots, y_{\ell-1})$

$\mathsf{Enc}_K(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
17: $S \leftarrow \mathsf{Setup}_1(K, \mathsf{instance}_1)$
18: $\mathsf{SEQ} \leftarrow \mathsf{Setup}_2(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$
19: **return** $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$

$\mathsf{Dec}_K(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
20: $S \leftarrow \mathsf{Setup}_1(K, \mathsf{instance}_1)$
21: $\mathsf{SEQ} \leftarrow \mathsf{Setup}_2(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$
22: **return** $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$

$\mathsf{Setup}_1(K, \mathsf{instance}_1)$
23: $(a, m) \leftarrow \mathsf{instance}_1$
24: $K_S \leftarrow \mathsf{PRF}^{L_2}(K, \mathsf{instance}_1, \mathsf{cste}_2)$
25: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
26: **return** S

$\mathsf{Setup}_2(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$
27: $(a, m) \leftarrow \mathsf{instance}_1$
28: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
29: $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
30: $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
31: **return** SEQ

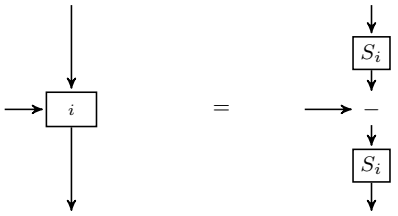


Fig. 3. Core Encryption $\mathsf{CEnc}[i_0, \ldots, i_{15}]$ with $\ell = 4$, $w = 2$, $w' = 1$, $n = 16$

Sbox Generation. Given an L_2-bit key K_S,

$$S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$$

The Sboxes can be any permutation of $\mathbf{Z}_a$. The choice of $\mathsf{PRNG}_{2,a,m}$ is open. We suggest one algorithm below.

Stateless Table-Driven Tokenization. The tokenization function uses a fixed pool S of Sboxes which plays the role of the stateless table secret. Given an L-bit key K (the key), a tweak tweak, a format specified by the parameters a and ℓ, the parameters (m, n, w, w'), the selected cipher suite $\mathsf{algo} = (\mathsf{PRNG}_1, \mathsf{PRF})$, and a plaintext $\mathsf{pt} \in \mathbf{Z}_a^\ell$, we define

$$\begin{aligned}
\mathsf{instance}_1 &= (a, m) \\
\mathsf{instance}_2 &= (\ell, n, w, w') \\
K_{\mathsf{SEQ}} &= \mathsf{PRF}^{L_1}(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak}) \\
\mathsf{ct} &= \mathsf{CEnc}_S[\mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})](\mathsf{pt}) \\
\mathsf{pt} &= \mathsf{CDec}_S[\mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})](\mathsf{ct})
\end{aligned}$$

where K_{SEQ} is an L_1-bit key which is used to generate SEQ. The value of cste_1 is a constant which encodes the label "$\mathsf{tokenization}$" and the size L_1. The function PRF^λ is a pseudorandom function with an L-bit key accepting an input of variable length and producing a λ-bit output.[7] With $\lambda = L_1$, we obtain a key K_{SEQ} for PRNG_1. The choice of PRF is open. Typically, we use $\mathsf{AES\text{-}CMAC}$.

Format-Preserving Encryption (FPE). The FPE uses a derived pool S of Sboxes. Given an L-bit key K, a tweak tweak, a format specified by the parameters a and ℓ, the parameters m and n, the selected cipher suite $\mathsf{algo} = (\mathsf{PRNG}_1, \mathsf{PRNG}_2, \mathsf{PRF})$, and an input $\mathsf{pt} \in \mathbf{Z}_a^\ell$, we define

$$\begin{aligned}
\mathsf{instance}_1 &= (a, m) \\
\mathsf{instance}_2 &= (\ell, n, w, w') \\
K_{\mathsf{SEQ}} &= \mathsf{PRF}^{L_1}(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_2, \mathsf{tweak}) \\
K_S &= \mathsf{PRF}^{L_2}(K, \mathsf{instance}_1, \mathsf{cste}_3) \\
S &= \mathsf{PRNG}_{2,a,m}(K_S) \\
\mathsf{ct} &= \mathsf{CEnc}_S[\mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})](\mathsf{pt}) \\
\mathsf{pt} &= \mathsf{CDec}_S[\mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})](\mathsf{ct})
\end{aligned}$$

[7] We will only use $\lambda = L_1 = L_2$. So, the superscript λ is only an informative notation.

where $\mathsf{instance}_1$, $\mathsf{instance}_2$, and PRF are like for tokenization, cste_2 is a constant which encodes the label "FPE SEQ" and the size L_1, and cste_3 is a constant which encodes the label "FPE Pool" and the size L_2. K_{SEQ} is an L_1-bit key which is used to generate SEQ. K_S is an L_2-bit key which is used to generate S. As m is the pool size and a is the Sbox size, changing m or a requires to recompute a new pool S, which is a tedious operation. However, changing ℓ should not require to recompute S. This is why we separate $\mathsf{instance}_1$ and $\mathsf{instance}_2$. In the pseudocode of Fig. 3, we separated the setup of S (in Setup_1 using $\mathsf{instance}_1$) and the setup of SEQ (in Setup_2 using $\mathsf{instance}_1$ and $\mathsf{instance}_2$).

Example. We consider $m = 256$ so that each Sbox index is a byte.

We define PRNG_1 from AES in CTR mode. We split $K_{\mathsf{SEQ}} = (K_1, \mathsf{IV}_1)$ with IV_1 of 128 bits with the last two bytes forced to 0.[8] Then,

$$\mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}}) =$$
$$\mathsf{trunc}_{8n}\left(\mathsf{AES}_{K_1}(\mathsf{IV}_1)\|\mathsf{AES}_{K_1}(\mathsf{IV}_1+1)\|\cdots\|\mathsf{AES}_{K_1}\left(\mathsf{IV}_1+\left\lceil\frac{n}{16}\right\rceil-1\right)\right)$$

where trunc_{8n} truncates to the first $8n$ bits and the addition is done modulo 2^{128}. The input of AES is an integer converted into a 128-bit string. We implicitly assume that the $8n$-bit result is converted into a sequence of bytes which define $(i_0, \ldots, i_{n-1})$.

The $\mathsf{PRNG}_{2,a,m}(K_S)$ generator first splits $K_S = (K_2, \mathsf{IV}_2)$ with IV_2 of 128 bits and generates a sequence

$$\mathsf{coins} = \mathsf{AES}_{K_2}(\mathsf{IV}_2)\|\mathsf{AES}_{K_2}(\mathsf{IV}_2+1)\|\cdots$$

of pseudorandom coins (this is AES in CTR mode) then applies a shuffling technique to generate each Sbox σ.[9] We can use the Fisher-Yates algorithm [26, p.145]:

1: initialize $\sigma(i) = i$ for $i = 0, \ldots, a-1$
2: **for** i from $a-1$ down to 1 **do**
3: pick $j \in \{0, \ldots, i\}$ at random using the random coins
4: exchange $\sigma[j]$ and $\sigma[i]$
5: **end for**

[8] Forcing the last two bytes of IV to 0 avoids that some slide properties in coins such as $\mathsf{IV}' = \mathsf{IV} + 1$ and $K' = K$ may occur (although the complexity to obtain such properties could be very high).

[9] Interestingly, if the algorithm generating coins is secure, there is no need to care about constant-time implementation for the Sbox generations as discussed in Section 5.

The way we generate j can follow many options [4,27,28]. To generate unbiased j from the random coins, we adapt a technique described by Lemire [27]. This approach uses L random input bits to generate a random value the range $[0, ..., i]$ (with $i < 2^L$) using only a multiplication with rejection probability $\frac{2^L \bmod (i+1)}{2^L}$. We choose L as a tradeoff between random bit consumption and rejection frequency: a value of $\lceil \log_2(i+1) \rceil + 4$ results in a maximum rejection probability of $1/2^4 = 0.0625$ and a much lower average rejection probability. We need on average a number of coins per Sbox equal to

$$\sum_{i=1}^{a-1} L \left(1 - \frac{2^L \bmod (i+1)}{2^L}\right)^{-1}$$

with $L = \lceil \log_2(i+1) \rceil + 4$. This generates unbiased Sboxes when the coins are random.

We define $\mathsf{PRF}^\lambda(K, \mathsf{list})$ as

$$\mathsf{trunc}_\lambda\left(\mathsf{AES\text{-}CMAC}_K(\mathsf{encode}(0, \mathsf{list})) \| \mathsf{AES\text{-}CMAC}_K(\mathsf{encode}(1, \mathsf{list})) \| \cdots\right)$$

which returns a λ-bit key. We only use $\lambda = L_1 = L_2$. In practice, we use $\lambda = 256$ or $\lambda = 384$ so that we only need 2 or 3 CMAC computations. Here, encode must be a non-ambiguous encoding of a list into a bitstring.

As an example, we can take $a = 10$ and $\ell = 10$ to encrypt a part of credit card numbers. For that, our recommended parameters below suggest to use $n = 390$ (i.e. 39 rounds of 10 layers), with $w = 3$ and $w' = 2$. We first need two AES computations to generate K_S (assuming encoding $(\mathsf{instance}_1, \mathsf{cste}_2)$ takes a single 128-bit block) and 129 (parallel) AES computations to generate the pool of Sboxes once for all. Thus, 131 AES for setting up the pool. Then, once for each tweak, we need 4 AES computations to generate K_{SEQ} then 25 AES computations to generate SEQ. Thus, 29 AES for each new tweak. This latter computation can be parallelized to have very fast processing. Finally, encryption/decryption requires no AES computation.

Security goal. Our construction is supposed to offer a pretty high security (e.g. 128-bit security) even though the input domain could be of very small size a^ℓ. Security holds even when the adversary can choose the parameters, the tweak, the plaintexts, and the ciphertexts. We also have security when the pool of Sboxes is known, which may happen for instance when the stateless table secret leaks in tokenization. Towards this goal, we will need PRNG_1 and PRNG_2 to be secure pseudorandom generators, PRF to be a

pseudorandom function, and we will reduce to the assumption that CEnc_S is a super-pseudorandom permutation (keyed by a random SEQ) when S is known but randomly set.

Recommended parameters. An instance defines a (a, m, ℓ, n, w, w') tuple and a set of algorithms. Admissible $(\mathsf{instance}_1, \mathsf{instance}_2)$ instances are defined by a set $\mathcal{F}$. We define what admissible tuples are here, depending on the targeted security s. We recommend $L = s$, $L_1 = L_2 = 2s$, $m = 256$, $a \geq 4$, $\ell \geq 2$, n multiple of ℓ, and w, w', and n tuned as $w \sim \sqrt{\ell}$, $w' \sim \sqrt{\ell}$, and $n \sim \ell^{\frac{3}{2}}$. More precisely,

$$
\begin{aligned}
w &= \min(\lfloor\sqrt{\ell}\rceil, \ell - 2) \\
w' &= \max(1, w - 1) \\
n &= \ell \times \left\lceil 2 \times \max\left(\frac{2s}{\ell \log_2 m}, \frac{s}{\sqrt{\ell}\ln(a-1)}, \frac{s}{\sqrt{\ell}\log_2(a-1)} + 2\sqrt{\ell}\right)\right\rceil
\end{aligned}
$$

(Note that the $\ell - 2$ in the min defining w is reached only for $\ell \in \{2, 3\}$. Similarly, the 1 in the max defining w' is reached only for $\ell \in \{2, 3\}$. For, $\ell > 3$, we have $w = \lfloor\sqrt{\ell}\rceil$ and $w' = w-1$.) These parameters were adjusted based on cryptanalysis and performance reasons. For $s = 128$, we write in Table 1 the number n/ℓ of "rounds" for a few sets of parameters.

For *quantum security*, we consider adversaries who can run quantum algorithms such as Grover [22] or Simon [35]. However, we do not assume quantum access to encryption/decryption oracles. To face quantum adversaries, we use the same formulas by replacing s by $2s$, except for $L_1 = L_2 = 3s$. We obtain that the number of rounds is doubled for the low ℓ values but remains unchanged for the large ones. This is because our security analysis suggests $n = \Omega(s\ell^{\frac{1}{2}} + \ell^{\frac{3}{2}})$. More details about figures are provided in the full version [17]. For quantum 128-bit security, there are a few changes in the underlying algorithms which should move to quantum 128-bit security. Namely, PRNG_1 and PRNG_2 are still $\mathsf{AES\text{-}CTR}$ but with a 256-bit key. As $\mathsf{AES\text{-}CMAC}$ does not offer 256-bit security, we need another algorithm or to twist CMAC with a 256-bit key.

Our design does not accommodate radix $a = 2$ or $a = 3$. We did not see as a disadvantage as radix $a = 4$ or 8 and $a = 9$ or 27 are possible if needed.

Rationales for w and w'. Our first design was using $w' = 1$ and $w = 0$ but had a powerful chosen ciphertext attack for up to ℓ^2 layers. Using $w > 0$ mitigated this attack and an optimal $w \sim \sqrt{\ell}$ was found. Then, we observed that decryption was faster than encryption because optimized

Table 1. Number $r = n/\ell$ of Rounds for 128-bit Security

ℓ:	2	3	4	5	6	7	8	9	10	12	16	32	50	64	100
$a = 4$	165	135	117	105	96	89	83	78	74	68	59	52	52	53	57
$a = 5$	131	107	93	83	76	70	66	62	59	54	48	46	47	48	53
$a = 6$	113	92	80	72	65	61	57	54	51	46	44	43	44	46	52
$a = 7$	102	83	72	64	59	55	51	48	46	43	41	41	43	45	50
$a = 8$	94	76	66	59	54	50	47	44	42	41	39	39	42	44	50
$a = 9$	88	72	62	56	51	47	44	42	40	39	38	38	41	43	49
$a = 10$	83	68	59	53	48	45	42	39	39	38	37	37	40	43	49
$a = 11$	79	65	56	50	46	43	40	38	38	37	36	37	40	42	48
$a = 12$	76	62	54	48	44	41	38	37	37	36	35	36	39	42	48
$a = 13$	73	60	52	47	43	39	37	36	36	35	34	36	39	41	48
$a = 14$	71	58	50	45	41	38	36	36	35	34	34	35	39	41	47
$a = 15$	69	57	49	44	40	37	36	35	34	34	33	35	38	41	47
$a = 16$	67	55	48	43	39	36	35	34	34	33	33	35	38	41	47
$a = 100$	40	33	28	27	26	26	25	25	25	26	26	30	34	37	44
$a = 128$	38	31	27	26	25	25	25	25	25	25	26	30	34	37	44
$a = 256$	33	27	25	24	23	23	23	23	23	24	25	29	33	37	44
$a = 1000$	32	22	21	21	21	21	21	21	21	22	23	28	32	36	43
$a = 1024$	32	22	21	21	21	21	21	21	21	22	23	28	32	36	43
$a = 10\,000$	32	22	18	18	18	18	19	19	19	20	21	27	32	35	42
$a = 65\,536$	32	22	17	17	17	17	17	18	18	19	21	26	31	35	42

compiled codes could process the computation of consecutive branches in parallel, but not for encryption. This motivated us to adopt a larger w', which is quite counter-intuitive. Indeed, it looks like slowing down the diffusion. However, our analysis did not show any need to increase the number of rounds. Using $w' \sim \sqrt{\ell}$ was good but we had to care for corner cases such as $\mathsf{gcd}(w, w', \ell) > 1$. Having $w' = w - 1$ ensures that $\mathsf{gcd}(w, w') = 1$.

3 Implementation Results

We implemented the algorithm in C++ on Intel Xeon 1.80GHz, using OpenSSL with AESNI support enabled.[10] It was compiled using g++ with flags

```
-O3 -Wall -Wextra -Wno-unused-const-variable -fPIC -DG_PLUS_PLUS
-falign-functions=32 -DHOT_ASSERTS=0
```

We took an open source implementation of FF1 and FF3[11] and optimized it for using AESNI and 128-bit arithmetic for modulo reduction where the input size allowed.[12]

Setting up a key and an instance requires generating the Sboxes. We need some AES computations to generate pseudorandom coins then apply the Fisher-Yates shuffling algorithm. Using our Sbox generation algorithm for $a = 10$, we need 61.8 random coins per Sbox on average. Hence, 124 AES parallel computations for $m = 256$. Note that our model allows S to be public, so even though side channel attacks might be considered against rejection sampling, this should be of no harm.

Setting up a tweak implies generating SEQ by some parallel AES computations. This can nicely exploit the AESNI pipeline architecture which is 3-4 times faster than the CBC mode of AES which is needed in FF1 and FF3. The SEQ sequence occupies a space of n bytes (with $m = 256$) in addition to the ma words of S, thus a total of 2-3KB for $a = 10$. For our implementation, we used a tweak of 8 bytes.

Finally, the core encryption needs no further AES computation. Due to $w' > 1$, consecutive encryption branches can be done in parallel, which speeds up the computation by the compiler. Similarly, $w > 1$ allows to run consecutive decryption branches in parallel.

[10] We tested other Intel CPUs and obtained comparable results.
[11] https://github.com/0NG/Format-Preserving-Encryption
[12] Our code is available on https://github.com/comForte/Format-Preserving-Encryption.

We report here some implementation results showing a big advantage for FAST over FF1 and FF3. We plot in Fig. 4 the time for an encryption/decryption cycle per $\mathbf{Z}_a$ symbol (so it is $\frac{1}{\ell}$ of the encryption time) for FAST, FF1, and FF3. For FAST, we plot both the time when we reset the tweak or we reuse it (hence with no AES computation). For FF1 or FF3, changing the tweak or reusing it has no visible difference on the curve so we did not plot it. Essentially, the figures for FAST are as follows for Setup_1, Setup_2, and Enc/Dec:

- AES key setup: 572 cycles. (With 128-bit key $K/K_S/K_{\mathsf{SEQ}}$ in either AES-CTR or AES-CMAC.)
- S generation (in PRNG_2 based on AES-CTR, after AES key setup): about 88 cycles for each Sbox to generate for $a = 10$. (for various a, this is: $10 : 88$, $16 : 137$, $32 : 282$, $64 : 617$, $128 : 1334$, $256 : 2831$). This includes the generation of the random coins and the shuffle.
- K_{SEQ} and K_S derivation (PRF based on two parallel AES-CMAC, after AES key setup): ~ 300 cycles per input block. With 16-byte instance encoding and 8-byte tweak, we need 7 AES computations in total (encryption of the zero-block, two CBC encryption of two blocks for K_{SEQ}, and two AES encryption of a single block for K_S); some of them could be done in parallel.
- SEQ derivation (in PRNG_1 based on AES-CTR, after AES key setup): 1.2 to 0.8 cycles for each of the n bytes.
- Core encryption/decryption: 5.7 cycles for each of the n layers, for $a = 10$.

For a best case comparison of most typical use cases, we optimized the open source FF1 implementation (which used big number modular arithmetic for all input lengths) to use the built-in (unsigned) `__int128` type provided by GCC as "native 128 bit integers" for this platform for $\ell \leq 32$. We did not change the implementation for $\ell > 32$ and acknowledge that it does not leverage full optimization potential. Therefore, we should take the $\ell > 32$ figures with a grain of salt, but we do have good performances compared to FF1/FF3 using built-in 128-bit integer arithmetic for $\ell \leq 32$ and nearly sustain this performance for longer inputs. (Note that FF3 limits to $\ell \leq 56$ and we abusively let an entry for $\ell = 64$.) We can safely say that FF1 performance per symbol is impacted negatively by the need for using a big integer library where modulo operands do not fit into 128 bits. The performance breakdown of FF1 for larger strings is also acknowledged by other researchers.[13]

[13] See `https://www.researchgate.net/publication/332088303_Evolution_of_Format_Preserving_Encryption_on_IoT_Devices_FF1` for instance.

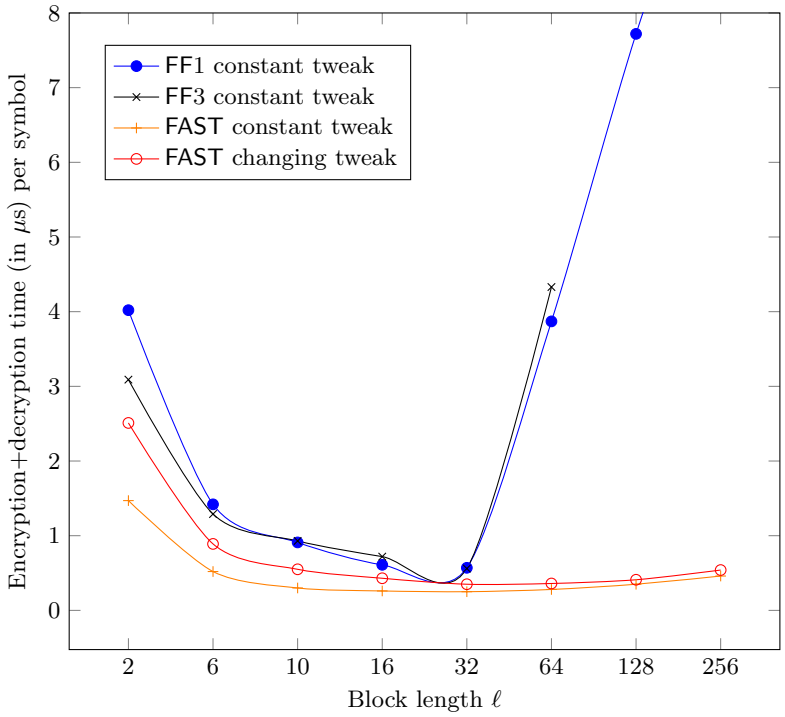


Fig. 4. Implementation Results for $a = 10$ (FF3 limits $\ell \leq 56$)

4 Security Proof

4.1 Security Definitions

We first define our strongest security notion by indistinguishability from an ideal FPE. It corresponds to the natural notion of tweakable super-pseudorandom permutation. I.e., the adversary can choose the tweak, a plaintext, a ciphertext. The adversary can further change the parameters (in the list $\mathcal{F}$ of allowed parameters). For instance, the adversary can change the format adaptively by keeping the same key.

First, we assume that the interface of the algorithm can be modeled by

$$F : K \mapsto \left(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak} \mapsto \text{even permutation of } \mathbf{Z}_a^\ell\right)$$

with $\mathsf{instance}_1 = (a, m)$ and $\mathsf{instance}_2 = (\ell, n, w, w')$ (indeed, we will see in Section 5 that our permutations of $\mathbf{Z}_a^\ell$ are always even). We assume a set $\mathcal{F}$ of admissible instances. Given a secret k, F_k maps $\mathsf{instance}_1$, $\mathsf{instance}_2$, and tweak to an even permutation $F_k(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ of $\mathbf{Z}_a^\ell$. Hence,

$$\begin{aligned}\mathsf{ct} &= F_k(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})\\ \mathsf{pt} &= \left(F_k(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})\right)^{-1}(\mathsf{ct})\end{aligned}$$

Based on this, we propose the security game in Fig. 5. In the Step 1 of the game, we implicitly mean that the game will define tables of $(\mathsf{pt}, \mathsf{ct})$ pairs for $(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ by lazy sampling, when needed in OTE and OTD. We obtain the following strong security notion.

The $\mathsf{INDstrong}$ security definition (as for "Strong INDistinguishability") gives oracle access (through OTE and OTD, as for "Oracle - Target Encryption/Decryption") to encryption/decryption with the ith target function with unknown key K_i. We limit the number of targets to a parameter τ as discussed below.

Definition 1. *We say that the algorithm is* $(T, \tau, q, \varepsilon)$-$\mathsf{INDstrong}$-*secure if for any* $\mathsf{INDstrong}$-*adversary* $\mathcal{A}$ *running the* $\mathsf{INDstrong}$ *game with* τ *targets, if the average complexity is limited by* T*, and if the average number of queries to the oracles is limited by* q *(we call this a* (T, q)*-limited adversary), the advantage is bounded by* ε*:*

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) = \Pr[\mathsf{INDstrong}_1 \to 1] - \Pr[\mathsf{INDstrong}_0 \to 1] \leq \varepsilon$$

The average complexity is measured when running the game on average over all random coins (from the game and the adversary). We favor average number of queries instead of sharp upper bounds on the number of queries of each type.

Game $\mathsf{INDstrong}_b$:
1: pick F following the interface at random:
for each K, $\mathsf{instance}_1$ (including a), $\mathsf{instance}_2$ (including ℓ), tweak, $F_K(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
2: pick $K[1], \ldots, K[\tau] \in \{0,1\}^L$ at random
3: run $\mathcal{A}^{\mathsf{OTE},\mathsf{OTD}} \to z$
4: **return** z

Oracle $\mathsf{OTE}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
5: **if** $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ **then return** $\perp$
6: **if** $i \notin \{1, \ldots, \tau\}$ **then return** $\perp$
7: **if** $b = 0$ **then**
8: **return** $F_{K_i}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
9: **else**
10: **return** $\mathsf{Enc}_{K_i}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
11: **end if**

Oracle $\mathsf{OTD}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
12: **if** $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ **then return** $\perp$
13: **if** $i \notin \{1, \ldots, \tau\}$ **then return** $\perp$
14: **if** $b = 0$ **then**
15: **return** $\left(F_{K_i}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})\right)^{-1}(\mathsf{ct})$
16: **else**
17: **return** $\mathsf{Dec}_{K_i}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
18: **end if**

Fig. 5. General Indistinguishability Game with Access to Ideal F

Our security model is quite powerful as it allows the adversary to attack one of several target keys and also to mount attacks in which he can choose the tweak, as well as the instance. Namely, the adversary can adaptively choose the format (a, ℓ) and parameters (m, n, w, w') (as long as they are in an admissible set) and reuse the same keys with multiple instances. The adversary can further choose the input to the encryption or the decryption algorithm.

Regarding **passive related-key attacks**, since our key K is only used in a PRF, the security of PRF against passive related-key attacks and the security of our design in a multi-target model imply the security against passive related-key attacks. By *passive* related-key attack, we mean those in which the adversary launches many targets $K[1], \ldots, K[\tau]$ and expect some to be related by random selection. We could also address *active* related-key attacks in which the adversary can force the creation of a target in a related manner to another, e.g. the creation of $K[1]$ and $K[2]$

such that $K[2] = K[1] + 1$. We could prove that if PRF resists to related-key attacks, then the FPE as well. Unfortunately, the PRF which we use is based on AES which does not resist to this type of attacks for keys larger than 128 bits [10]. However, we are not aware of any related-key attack on our FPE.

On limiting the number of targets. An adversary can always prepare a dictionary of u keys with the encryption of a fixed plaintext, make a chosen plaintext attack on τ targets, and spot if any target belongs to the dictionary [9]. This gives an easy distinguisher with advantage $u\tau/2^L$. The value of τ could in principle reach a value close to q while the value of u would be proportional to the time complexity T. This type of attack applies well to AES too. With $L = 128$, we can take $u = \tau = 2^{64}$ and have a good distinguisher. In practice, it makes sense to assume that the number of targets is limited to a small number τ. Since we do not want to enlarge the key length due to this attack but still offer security with large q, we chose to introduce a low τ parameter.

Meaning of a 128-bit security. We target a "128-bit security" (classical or quantum). However, the meaning of 128-bit security is often incorrectly understood as any attack would need at least $T = 2^{128}$ complexity to succeed. Attacks are however measured by several metrics such as T, q, ε, and τ. We could have attacks with a small T and a ridiculously low ε still 128-bit security.

It is hard to compare two attacks with different advantages. Sometimes, a (T, q, ε)-attack could be amplified into a $(kT, kq, k\varepsilon)$-attack (in which case we could say that the attack needs $k = \frac{1}{\varepsilon}$ to succeed, hence it has a normalized complexity of $\frac{T}{\varepsilon}$) but sometimes, the amplification works as a $(kT, kq, \sqrt{k}\varepsilon)$-attack (in which case we could say that the attack needs $k = \frac{1}{\varepsilon^2}$ to succeed, hence it has a normalized complexity of $\frac{T}{\varepsilon^2}$). It could also be the case that no amplification is possible. Hence, there is no general rule. We could try, as much as possible, to focus on adversaries achieving a constant advantage such as advantage $\frac{1}{2}$.

Another difficulty is the introduction of the multiple instances, target keys, and tweaks as mentioned above. Things become easier when the security notion implies a single key, a single target, and a single tweak.

Hence, by "128-bit security", we mean "like AES". We mean that in a same (q, ε, τ) configuration, we could have an attack against AES with a T which is lower or equal. If an adversary achieves a constant advantage with a single target, a single instance, and a single tweak, it must have a complexity comparable to an exhaustive search on a 128-bit key.

Indistinguishability to even permutations. Most of existing ciphers are even permutations. A random even permutation over a domain of size N is perfectly indistinguishable from a random permutation when the number of available input/output pairs does not reach $N-1$. Block ciphers are defined over domains with large N so this is not a problem. In FPE, N can be very small and the parity may leak. Hence, we preferred to make sure that our FPE are even permutations and to adopt as a security model the indistinguishability to a random even permutation. In Section 5, we prove that our FPE with the selected parameters is even.

Known pool security. Our construction is based on a pool S of Sboxes. We can enrich the security notion to capture attacks in which the adversary learns S: a "known S attack".[14] There are several motivations for considering known S attacks:

- In tokenization mode, S is fixed once for all and the secret is only determining SEQ. We could imagine that by time, the pool S eventually leaks.
- Some (side-channel) attacks may uncover some Sboxes.
- Security in this model with single target and single instance offers some nice security reductions from strong security.

Hence, it is relevant to wonder if some attacks could exploit having the pool S as prior knowledge and with a random S and SEQ instead of a pseudorandom one. We enrich our security game as in Fig. 6. This is the $\mathsf{INDKSsprp}$ game (as for "Known-S Super-Pseudorandom Permutation"), working with the OE and OD oracles (as for "Oracle - Encryption/Decryption") and OT_1 and OT_2 oracles to set up the target parameters.

Definition 2. *We say that the algorithm is* (T, q, ε)-$\mathsf{INDKSsprp}$-*secure if for any* (T, q)-*limited* $\mathsf{INDKSsprp}$-*adversary* $\mathcal{A}$ *running the* $\mathsf{INDKSsprp}$ *game, the advantage is bounded by* ε*:*

$$\mathsf{Adv}_{\mathsf{INDKSsprp}}(\mathcal{A}) = \Pr[\mathsf{INDKSsprp}_1 \to 1] - \Pr[\mathsf{INDKSsprp}_0 \to 1] \leq \varepsilon$$

We could also consider revealing K_S with the same arguments. We would then have to set S generated by PRNG_2 on a random K_S in the game. One advantage is that we would no longer need a specific security

[14] A "chosen S attack" leads to trivial attacks. For instance, the adversary could pick all Sboxes equal, or all Sboxes linear (over $\mathbf{Z}_a$). Therefore, we do not consider chosen-S models.

Game $\mathsf{INDKSsprp}_b$:
1: run $\mathcal{A}^{\mathsf{OE},\mathsf{OD},\mathsf{OT}_1,\mathsf{OT}_2} \to z$
2: **return** z

Oracle $\mathsf{OT}_1(\mathsf{instance}_1)$
3: **if** S defined **then return** $\perp$
4: $(a, m) \leftarrow \mathsf{instance}_1$
5: pick $S_0, \ldots, S_{m-1}$ random permutations of $\mathbf{Z}_a$
6: $S \leftarrow (S_0, \ldots, S_{m-1})$
7: **return** S

Oracle $\mathsf{OT}_2(\mathsf{instance}_2)$
8: **if** SEQ defined **then return** $\perp$
9: **if** S undefined **then return** $\perp$
10: **if** $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ **then return** $\perp$
11: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
12: pick a random even permutation F of $\mathbf{Z}_a^\ell$
13: pick random $\mathsf{SEQ} \in \{0, \ldots, m-1\}^n$
14: **return**

Oracle $\mathsf{OE}(\mathsf{pt})$
15: **if** S or SEQ undefined **then return** $\perp$
16: **if** $b = 0$ **then**
17: **return** $F(\mathsf{pt})$
18: **else**
19: **return** $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
20: **end if**

Oracle $\mathsf{OD}(\mathsf{ct})$
21: **if** S or SEQ undefined **then return** $\perp$
22: **if** $b = 0$ **then**
23: **return** $F^{-1}(\mathsf{ct})$
24: **else**
25: **return** $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
26: **end if**

Fig. 6. Single-Target/Instance/Tweak Known S Indistinguishability Game

for PRNG_2 (it would be integrated in $\mathsf{INDKSsprp}$ with the above modification) and we could "compress" the storage of S by K_S if needed.

We finally define the one-time PRNG security and PRF security of our algorithms by the games in Fig. 7 and Fig. 8.

Definition 3. *We say that the algorithm is* (T, ε)-$\mathsf{INDPRNG}_1$*-secure if for any* T*-limited* $\mathsf{INDPRNG}_1$*-adversary* $\mathcal{A}$ *running the* $\mathsf{INDPRNG}_1$ *game, the advantage is bounded by* ε*:*

$$\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{A}) = \Pr[\mathsf{INDPRNG}_1^1 \to 1] - \Pr[\mathsf{INDPRNG}_1^0 \to 1] \leq \varepsilon$$

We similarly define $\mathsf{INDPRNG}_2$*-security. We similarly say that* PRF *is* (T, q, ε)-PRF*-secure if for any* (T, q)*-limited* PRF*-adversary* $\mathcal{A}$ *running the* PRF *game, the advantage is bounded by* ε*.*

We can now formally reduce the strong security $\mathsf{INDstrong}$ to the weak security $\mathsf{INDKSsprp}$. The next step will be to heuristically assess the weak security of our construction.

Theorem 4. *There exists a (small) constant* c *such that for any* s*,* T*,* q*, and any* (T, τ, q)*-limited* $\mathsf{INDstrong}$*-adversary* $\mathcal{A}$*, there exist* $q' \leq q$*, a* $(T + qc, q')$*-limited* $\mathsf{INDKSsprp}$*-adversary* $\mathcal{A}'$*, a* $(T, q+1)$*-limited* PRF*-adversary* $\mathcal{B}$*, a* $(T + qc)$*-limited* $\mathsf{INDPRNG}_1$*-adversary* $\mathcal{C}$ *and a* $(T + qc)$*-*

Game $\mathsf{INDPRNG}_1^b$:
1: run $\mathcal{A}^{\mathsf{OG}} \to z$
2: **return** z

Oracle $\mathsf{OG}(m, n)$
3: **if** SEQ defined **then return** $\perp$
4: **if** $b = 0$ **then**
5: pick random $\mathsf{SEQ} \in \{0, \ldots, m-1\}^n$
6: **else**
7: pick $K_{\mathsf{SEQ}} \in \{0,1\}^{L_1}$ at random
8: $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
9: **end if**
10: **return** SEQ

Game $\mathsf{INDPRNG}_2^b$:
11: run $\mathcal{A}^{\mathsf{OG}} \to z$
12: **return** z

Oracle $\mathsf{OG}(a, m)$
13: **if** S defined **then return** $\perp$
14: **if** $b = 0$ **then**
15: pick $S_0, \ldots, S_{m-1}$, random $\mathbf{Z}_a$ permutations
16: $S \leftarrow (S_0, \ldots, S_{m-1})$
17: **else**
18: pick $K_S \in \{0,1\}^{L_2}$ at random
19: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
20: **end if**
21: **return** S

Fig. 7. Indistinguishability Game for PRNG

Game PRF^b:
1: pick a random function F with same input/output domain as PRF
2: pick K at random
3: run $\mathcal{A}^{\mathsf{OF}} \to z$
4: **return** z

Oracle $\mathsf{OF}(x)$
5: **if** $b = 0$ **then**
6: **return** $F(x)$
7: **else**
8: **return** $\mathsf{PRF}_K(x)$
9: **end if**

Fig. 8. Indistinguishability Game for PRF

limited $\mathsf{INDPRNG}_2$*-adversary* $\mathcal{D}$ *such that*

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq q \cdot \frac{\mathsf{Adv}_{\mathsf{INDKSsprp}}(\mathcal{A}')}{q'} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + q \cdot (\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}) + \mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D})) + \frac{\tau^2}{2} \cdot 2^{-L}$$

In general, τ is limited by what is allowed in the implementation. The proof is provided in the full version [17].

Given that $\mathsf{Adv}_{\mathsf{INDKSsprp}}(\mathcal{A}')$, $\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C})$, or $\mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D})$ can easily be as large as $T \cdot q' \cdot m^{-n}$, $T \cdot 2^{-L_1}$, or $T \cdot 2^{-L_2}$ respectively (by doing an exhaustive search on a list limited to T), it is crucial that $n \log_2 m$ and L_1 are both larger than $2s$. This will match the criteria (2) and (12) in Section 5. Similarly, $\mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B})$ can be as large as $T \cdot 2^{-L} + q^2 \cdot 2^{-\lambda}$, with λ being the output length of the PRF^λ. By plugging all these values we obtain the upper bound $q \cdot T \cdot \left(m^{-n} + 2^{-L_1} + 2^{-L_2}\right) + \tau \cdot (T \cdot 2^{-L} + q^2 \cdot 2^{-\lambda}) + \frac{\tau^2}{2} \cdot 2^{-L}$ for $\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A})$. Given that $L = s$ and $n \log_2 m \approx \lambda = L_1 = L_2 = 2s$, this is $3q \cdot T \cdot 2^{-2s} + \tau \cdot (T \cdot 2^{-s} + q^2 \cdot 2^{-2s}) + \frac{\tau^2}{2} \cdot 2^{-s}$. The first term is fine. The other terms account for a normal degradation of the security by a factor τ in a multi-target setting. For instance, with the extreme case $q \approx T$, we need $T \approx \frac{1}{\tau} 2^s$ to reach a constant advantage.

The proof of Th. 4 follows several reduction steps as follows.

- $\Gamma_b^0(\mathcal{A})$: We start with an $\mathsf{INDstrong}$ game.
- $\Gamma_j^1(\mathcal{A})$: Reduce to a game with independent F_i instead of $F_{K[i]}$. (Cost is $\frac{\tau^2}{2} 2^{-L}$ due to possible collisions on $K[i]$.)
- $\Gamma_b^2(\mathcal{A}_j)$: Reduce to single target K. (Cost is a factor τ using an adversary for each target.)
- $\Gamma_b^3(\mathcal{A}_j)$: Replace PRF by a random function. (Cost is $\mathsf{Adv}_{\mathsf{PRF}}$.)
- $\Gamma_b^4(\mathcal{A}_j)$: Reduce to known S. (No cost.)
- $\Gamma_{j'}^5(\mathcal{A}_j)$ — $\Gamma_b^6(\mathcal{A}_{j,j'})$: Reduce to single $\mathsf{instance}_1$ with hybrid games and single adversary, then several adversaries playing a unique single-instance game.
- $\Gamma_{j''}^7(\mathcal{A}_{j,j'})$ — $\Gamma_b^8(\mathcal{A}_{j,j',j''})$: Reduce to single $(\mathsf{instance}_2, \mathsf{tweak})$ with the same method.
- $\Gamma_b^9(\mathcal{A}_{j,j',j''})$: Replace PRNG_1 by a truly random function. (Cost is $\mathsf{Adv}_{\mathsf{INDPRNG}_1}$.)
- $\mathsf{INDKSsprp}_b(\mathcal{A}_{j,j',j''})$: Replace PRNG_2 by a truly random function. (Cost is $\mathsf{Adv}_{\mathsf{INDPRNG}_2}$.)
- $\mathsf{INDKSsprp}_b(\mathcal{A}')$: We obtain an $\mathsf{INDKSsprp}$ game.

We use a trick to cumulate all hybrids which results in only a factor q. Namely, we take the hybrid with maximal $\frac{\mathsf{Adv}}{q'}$ and we upper bound the advantage of hybrids by their number of queries times this ratio. Summing them all results in $q \cdot \frac{\mathsf{Adv}}{q'}$.

5 Rationales

On the choice of w'. The $\ell = 2$ is a special case where we shall use $w = 0$ and $w' = 1$. For $\ell > 2$, we use $w = \lfloor \sqrt{\ell} \rceil$ and we wonder how to select w'.

We clearly need $w' > 0$. Furthermore, it is required that $w < \ell - w'$ to avoid changes in branch orders. Without loss of generality, we assume $w' \leq w$ (with the exception of $\ell = 2$ for which $w = 0$). The previous design was using $w' = 1$. However, it may be nice for performances to have w' of same order of magnitude as w.

There is an easy attack when $d \geq 2$, with $d = \mathsf{gcd}(w, w', \ell)$: if two plaintexts pt and pt' have a difference of form $\mathsf{pt}' - \mathsf{pt} = (?0^{d-1})^{\ell/d}$, this property is preserved after d layers. Hence, we have a distinguisher with advantage close to 1 and any number of layers. To avoid this problem, we adopt $w' = \max(1, w - 1)$ which guaranties that $\mathsf{gcd}(w, w') = 1$ so $d = 1$ as well.

Parity of CEnc. We prove that the parity of encryption only depends on the parameters a, ℓ, and n. Hence, it does not leak.

Lemma 5. *For every y in a domain of size A, we consider a permutation P_y. The $(x, y) \mapsto (P_y(x), y)$ is a permutation with parity equal to the sum of the parities of every P_y.*

Proof. For y fixed, the permutation restricts to a permutation with same cycle structure as P_y. Hence, the permutation is a composition of permutations with same cycle structure as P_y, for every y. □

Lemma 6. *The $(x, y) \mapsto (x + y, y)$ permutation over $\mathbf{Z}_a^2$ is even when a is odd and has the parity of $\frac{a}{2}$ when a is even. The same holds for $(x, y) \mapsto (x - y, y)$.*

Proof. We let $P_y(x) = x + y \bmod a$ and we apply Lemma 5. P_y is the composition of $\frac{a}{k}$ cycles of length k, where k is the order of y in $\mathbf{Z}_a$. The parity of P_y is the parity of $\frac{a}{k}(k - 1)$.

For $y = 0$, P_y is even.

For a odd, we notice that for $y \neq 0$, P_y and P_{-y} have the same parity hence cancel each other. Hence, the permutation is even when a is odd.

For a even, the same observation holds for $y \in \{1, \ldots, \frac{a}{2} - 1\}$. Hence, the parity is the same as the parity of $P_{a/2}$. We have $k = 2$ for $P_{a/2}$, thus its parity is the one of $\frac{a}{2}$. □

Lemma 7. *The parity of* $\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}]$ *is as follows:*

- *for* $a \bmod 4 \neq 3$*, it is even;*
- *for* $a \bmod 4 = 3$*, it is the parity of* $n(\ell - 1)$*.*

(For this, we assume that $\ell = 2$ *implies* $w = 0$*.)*

Since n is a multiple of ℓ, $n(\ell-1)$ is always even. Hence, the permutation $\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}]$ is always even.

Proof. The encryption is the composition of permutations of n layers using $S_{i_0}, \ldots, S_{i_{n-1}}$. The layer using an Sbox σ is the composition of

P1 the permutation $(x_0, \ldots, x_{\ell-1}) \mapsto (x_0 + x_{\ell-1}, x_1, \ldots, x_{\ell-1})$,
P1′ the permutation $(x_0, \ldots, x_{\ell-1}) \mapsto (x_0 - x_w, x_1, \ldots, x_{\ell-1})$,
P2 the permutation $(x_0, \ldots, x_{\ell-1}) \mapsto (\sigma(x_0), x_1, \ldots, x_{\ell-1})$ (used twice),
P3 the permutation $(x_0, \ldots, x_{\ell-1}) \mapsto (x_1, \ldots, x_{\ell-1}, x_0)$.

By writing $P(x_0, x_{\ell-1}) = (x_0 + x_{\ell-1}, x_{\ell-1})$, for $\ell > 2$, P1 has the form of the permutation of Lemma 5 with y in a domain of size $A = a^{\ell-2}$ and all P_y set to P, so the parity of P1 is $a^{\ell-2}$ times the parity of P, which is the same as a times the parity of P. For $\ell = 2$, the permutation P1 is P so has the same parity. By using Lemma 6, we obtain the parity of P. Therefore, the parity of P1 is

- (a even and $\ell > 2$) even,
- (a even and $\ell = 2$) the parity of $\frac{a}{2}$,
- (a odd) even.

The same holds for P1′.

The second permutation P2 has the form of the permutation given in Lemma 5 with a fixed permutation $P = \sigma$ with y in a domain of size $A = a^{\ell-1}$. We obtain that the parity of P2 is the same as a times the parity of σ. Therefore, the parity of P2 is

- (a even and $\ell > 2$) even,
- (a even and $\ell = 2$) even,
- (a odd) the parity of σ.

However, P2 is used twice so its parity has no impact.

The third permutation P3 is the composition of $\ell - 1$ permutations of form $(x_0, \dots x_{\ell-1}) \mapsto (x_0, \dots, x_{i-1}, x_{i+1}, x_i, x_{i+2}, \dots, x_{\ell-1})$ exchanging the coordinates of index i and $i+1$. Those permutations can be written as in Lemma 5 for $\ell > 2$ with $P(x_i, x_{i+1}) = (x_{i+1}, x_i)$. This permutation has a fixed points and $\frac{a^2-a}{2}$ cycles of length two. Hence, it has the same parity as $\frac{a^2-a}{2}$. For $\ell > 2$, we deduce that the parity of P3 is the parity of $(\ell-1)a\frac{a^2-a}{2}$. For $\ell = 2$, the parity of P3 is the parity of $(\ell-1)\frac{a^2-a}{2}$. Therefore, the parity of P3 is

- (a even and $\ell > 2$) even,
- (a even and $\ell = 2$) the parity of $\frac{a^2-a}{2}$,
- (a odd) the parity of $(\ell-1)\frac{a^2-a}{2}$.

We sum the parities over the n layers and obtain the result. In the a even and $\ell > 2$ case, everything is even. In the a even and $\ell = 2$ case, we first observe that $w = 0$ so P1′ is unused. Then, we observe that the parity of $\frac{a}{2} + \frac{a^2-a}{2}$ is always even. In the a odd case, we observe that $\frac{a^2-a}{2}$ is even if and only if $a \bmod 4 = 1$. □

Slide attack on previous SEQ *scheme.* A previous version of our construction was using SEQ with a sequence derived from a periodic repetition (or modified repetition) of an AES-generated sequence. This made the entire encryption process being a self-iteration on a simpler function, which is subject to a devastating slide attack. In our construction, SEQ is a random sequence of indices and the probability that it becomes periodic is negligible. So, the slide attack is defeated.

Best known attacks. We investigated several attack methods which we list here together with the requirement that security implies. They are detailed in the full version [17].

- Linear collapse. With too small parameters, there are chances that the encryption becomes linear over $\mathbf{Z}_a$.

$$\min(m, n) > \frac{s}{\log_2(a-1)! - \log_2 \varphi(a)} \tag{1}$$

- Known Sbox pool dictionary attack. With not enough layers, a partial dictionary attack can work.

$$n > \frac{2s}{\log_2 m} \tag{2}$$

– Chosen ciphertext distinguisher with $w = 0$ and $w' = 1$.

$$n > \ell(\ell - 1) + \frac{s}{\log_2 \min(m, a\sqrt{a})} \qquad \text{if } w = 0 \text{ and } w' = 1 \tag{3}$$

– Chosen plaintext distinguisher with $w > 1$ and w and w' coprime.

$$n > (w' + 1)(\ell - w') + \frac{s - 1 + 2\log_2 a}{\log_2 \min(m, a!)} \qquad \text{if } w > 1 \tag{4}$$

– Chosen ciphertext distinguisher with $w > 1$ and w and w' coprime.

$$n > (w + 1)(\ell - w) + \frac{s - 1 + 2\log_2 a}{\log_2 \min(m, a!)} \qquad \text{if } w > 1 \tag{5}$$

– Chosen plaintext distinguisher with $w + w'$ factor of ℓ.

$$n > (w + w')\frac{s - 1 + \ell \log_2 a}{2\log_2(a - 1)} \tag{6}$$

$$n > \frac{s - 1 + \ell \log_2 a}{\log_2 \min(m, a!)} \tag{7}$$

– Chosen plaintext distinguisher with w' factor of $\ell + 1$.

$$n > w'\frac{s - 1 + (\ell + 2)\log_2 a}{\log_2 a} \tag{8}$$

$$n > \frac{s - 1 + (\ell + 2)\log_2 a}{\log_2 \min(m, a!)} \tag{9}$$

– Differential and linear attacks. Given a framework which captures truncated differentials, impossible differentials, regular differentials, and linear hulls over the algebraic structure of $\mathbf{Z}_a$, we can derive a lower bound for n to achieve security.

$$n > \frac{s\sqrt{\ell}}{\ln(a - 1)} \tag{10}$$

– Fixed point attacks. It may happen that all selected Sboxes have 0 as a fixed point, which would lead to a trivial attack.

$$n > \frac{s}{\log_2 a} \tag{11}$$

– Collision attacks. Trying many tweak until there is a collision on K_{SEQ} leads to a trivial distinguishing attack. Hence, the bitlength of K_{SEQ} must be at least $2s$.

$$L_1 \geq 2s \tag{12}$$

Choice of parameters. The generic attacks have clearly shown that w should be around $\sqrt{\ell}$ but lower bounded by $\ell - 2$. As for the selection of n, we looked at all obtained requirements. For each a and ℓ, we computed the suggested w and the maximal requirement. Table 2 represents the minimal value for n with a subscript set to the equation number of the critical requirement. We set $s = 128$. For instance, $a = 10$ and $\ell = 6$ (encryption of 6 decimal digits) has entry 143_{10} which means that we need $n \geq 143$ layers (24 rounds) which is critical for Eq (10), the differential and linear attacks. We suggest 48 rounds which gives a good security margin.

Table 2. Minimal Number n of Layers Following Criteria with Reference to the Critical One

ℓ:	2	3	4	5	6	7	8	9	10	12	16	32	50	64	100
$a = 4$	165_{10}	202_{10}	234_{10}	261_{10}	286_{10}	309_{10}	330_{10}	350_{10}	369_{10}	404_{10}	467_{10}	663_6	931_6	1207_6	1960_6
$a = 5$	131_{10}	160_{10}	185_{10}	207_{10}	227_{10}	245_{10}	262_{10}	277_{10}	292_{10}	320_{10}	370_{10}	554_6	791_6	1034_6	1707_6
$a = 6$	113_{10}	138_{10}	160_{10}	178_{10}	195_{10}	211_{10}	225_{10}	239_{10}	252_{10}	276_{10}	319_{10}	497_6	718_6	945_6	1578_6
$a = 7$	102_{10}	124_{10}	143_{10}	160_{10}	175_{10}	190_{10}	203_{10}	215_{10}	226_{10}	248_{10}	286_{10}	462_6	673_6	890_6	1499_6
$a = 8$	94_{10}	114_{10}	132_{10}	148_{10}	162_{10}	175_{10}	187_{10}	198_{10}	209_{10}	228_{10}	264_{10}	437_6	642_6	853_6	1445_6
$a = 9$	88_{10}	107_{10}	124_{10}	138_{10}	151_{10}	163_{10}	175_{10}	185_{10}	195_{10}	214_{10}	247_{10}	419_6	619_6	825_6	1406_6
$a = 10$	83_{10}	101_{10}	117_{10}	131_{10}	143_{10}	155_{10}	165_{10}	175_{10}	185_{10}	202_{10}	234_{10}	405_6	602_6	804_6	1377_6
$a = 11$	79_{10}	97_{10}	112_{10}	125_{10}	137_{10}	148_{10}	158_{10}	167_{10}	176_{10}	193_{10}	223_{10}	394_6	587_6	787_6	1353_6
$a = 12$	76_{10}	93_{10}	107_{10}	120_{10}	131_{10}	142_{10}	151_{10}	161_{10}	169_{10}	185_{10}	214_{10}	385_6	576_6	773_6	1334_6
$a = 13$	73_{10}	90_{10}	104_{10}	116_{10}	127_{10}	137_{10}	146_{10}	155_{10}	163_{10}	179_{10}	207_{10}	377_6	566_6	762_6	1318_6
$a = 14$	71_{10}	87_{10}	100_{10}	112_{10}	123_{10}	133_{10}	142_{10}	150_{10}	158_{10}	173_{10}	200_{10}	370_6	558_6	752_6	1304_6
$a = 15$	69_{10}	85_{10}	98_{10}	109_{10}	119_{10}	129_{10}	138_{10}	146_{10}	154_{10}	169_{10}	195_{10}	365_6	551_6	743_6	1292_6
$a = 16$	67_{10}	82_{10}	95_{10}	106_{10}	116_{10}	126_{10}	134_{10}	142_{10}	150_{10}	164_{10}	190_{10}	359_6	545_6	736_6	1282_6
$a = 100$	40_{10}	49_{10}	56_{10}	63_{10}	69_{10}	74_{10}	79_{10}	84_{10}	89_{10}	97_{10}	124_6	282_6	451_6	625_6	1135_6
$a = 128$	38_{10}	46_{10}	53_{10}	60_{10}	65_{10}	70_{10}	75_{10}	80_{10}	84_{10}	92_{10}	120_6	277_6	444_6	618_6	1125_6
$a = 256$	33_{10}	41_{10}	47_{10}	52_{10}	57_{10}	62_{10}	66_{10}	70_{10}	74_{10}	81_{10}	112_6	264_6	429_6	600_6	1102_6
$a = 1000$	32_2	33_{10}	38_{10}	42_{10}	46_{10}	50_6	53_{10}	56_{10}	59_{10}	65_{10}	101_6	247_6	408_6	576_6	1072_6
$a = 1024$	32_2	32_2	37_{10}	42_{10}	46_{10}	50_6	53_{10}	56_{10}	59_{10}	64_{10}	101_6	246_6	408_6	576_6	1071_6
$a = 10\,000$	32_2	32_2	32_2	32_2	35_{10}	42_6	44_6	47_6	49_6	56_5	90_6	229_6	388_6	552_6	1041_6
$a = 65\,536$	32_2	32_2	32_2	32_2	32_2	38_6	40_5	44_5	48_5	56_5	84_6	220_6	377_6	540_6	1026_6

As we can see, low ℓ values have Eq. (10) (differential cryptanalysis) as critical while high ℓ have one of the generic attacks as critical. Large ℓ values have Eq. (6) (differential chosen plaintext attack) as best attack. We can also see that Eq. (2) (Dictionary Attack) appears for low ℓ and large a. Actually, our lower bounds asymptotically give $n = \Omega(s\ell^{\frac{1}{2}} + \ell^{\frac{3}{2}})$ when s and ℓ grow to infinity. To select n, we doubled the requirement for a safety margin and we took the smallest acceptable multiple of ℓ by using Eq. (2), Eq. (10), and Eq. (6).

6 Conclusion

We constructed a flexible truly-SPN FPE. We proved that it is competitive in terms of both throughput and security, even when the encryption domain is very small. We encourage researchers to analyze the security.

References

1. Retail Financial Services - Requirements for Protection of Sensitive Payment Card Data - Part 1: Using Encryption Method. ANSI X9.119-1-2016. American National Standards Institute.
2. Retail Financial Services - Requirements for Protection of Sensitive Payment Card Data - Part 2: Implementing Post Authorization Tokenization Systems. ANSI X9.119-2-2017. American National Standards Institute.
3. Recommendation for Block Cipher Modes of Operation: Methods for Format Preserving Encryption. NIST Special Publication (SP) 800-38G. National Institute of Standards and Technology.
   `http://dx.doi.org/10.6028/NIST.SP.800-38G`
4. Thomas Baignères, Matthieu Finiasz. Dial C for Cipher. In *Selected Areas in Cryptography'06*, Montreal, Quebec, Canada, Lecture Notes in Computer Science 4356, pp. 76–95, Springer-Verlag, 2007.
5. Thomas Baignères, Jacques Stern, Serge Vaudenay. Linear Cryptanalysis of Non Binary Ciphers. In *Selected Areas in Cryptography'07*, Ottawa, Ontario, Canada, Lecture Notes in Computer Science 4876, pp. 184–211, Springer-Verlag, 2007.
6. Mihir Bellare, Viet Tung Hoang, Stefano Tessaro. Message-Recovery Attacks on Feistel-Based Format-Preserving Encryption. In *23rd ACM Conference on Computer and Communications Security*, Vienna, Austria, pp. 444–455, ACM Press, 2016.
7. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, Till Stegers. Format-Preserving Encryption. In *Selected Areas in Cryptography'09*, Calgary, Alberta, Canada, Lecture Notes in Computer Science 5867, pp. 295–312, Springer-Verlag, 2009.
8. Mihir Bellare, Phillip Rogaway, Terence Spies. The FFX Mode of Operation for Format-Preserving Encryption.
   `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf`
9. Eli Biham. How to Decrypt or even Substitute DES-Encrypted Messages in 2^{28} steps. *Information Processing Letters*, vol. 84, pp. 117–124, 2002.
10. Alex Biryukov, Dmitry Khovratovich. Related-key Cryptanalysis of the Full AES-192 and AES-256. In *Advances in Cryptology ASIACRYPT'09*, Tokyo, Japan, Lecture Notes in Computer Science 5912, pp. 1–18, Springer-Verlag, 2009.
11. John Black, Phillip Rogaway. Ciphers with Arbitrary Finite Domains. In *Topics in Cryptology (CT-RSA'02): The Cryptographers' Track at the RSA Conference 2002*, San Jose, California, U.S.A., Lecture Notes in Computer Science 2271, pp. 114–130, Springer-Verlag, 2002.
12. Eric Brier, Thomas Peyrin, Jacques Stern. BPS: A Format-Preserving Encryption Proposal.
    `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf`

13. Michael Brightwell, Harry E. Smith. Using Datatype-Preserving Encryption To Enhance Data Warehouse Security. 20th NISSC Proceedings pp. 141–149. 1997.
http://csrc.nist.gov/nissc/1997/proceedings/141.pdf
14. Debrup Chakraborty, Sebati Ghosh, Cuauhtemoc Mancillas López, Palash Sarkar. FAST: Disk Encryption and Beyond. Eprint 2017/849
https://eprint.iacr.org/2017/849
15. Donghoon Chang, Mohona Ghosh, Kishan Chand Gupta, Arpan Jati, Abhishek Kumar, Dukjae Moon, Indranil Ghosh Ray, Somitra Kumar Sanadhya. SPF: A New Family of Efficient Format-Preserving Encryption Algorithms. In *Information Security and Cryptology (INSCRYPT'16)*, Beijing, China, Lecture Notes in Computer Science 10143, pp. 64–83, Springer-Verlag, 2016.
16. Donghoon Chang, Mohona Ghosh, Arpan Jati, Abhishek Kumar, Somitra Kumar Sanadhya. eSPF: A Family of Format-Preserving Encryption Algorithms Using MDS Matrices. In *Security, Privacy, and Applied Cryptography Engineering (SPACE'17)*, Goa, India, Lecture Notes in Computer Science 10662, pp. 133–150, Springer-Verlag, 2017.
17. F. Betül Durak, Henning Horst, Michael Horst, Serge Vaudenay. FAST: Secure and High Performance Format-Preserving Encryption and Tokenization. Eprint 2021/1171
https://eprint.iacr.org/2021/1171
18. F. Betül Durak, Serge Vaudenay. Breaking the FF3 Format-Preserving Encryption Standard Over Small Domains. In *Advances in Cryptology CRYPTO'17*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 10401–10403, pp. 679–707 vol. 2, Springer-Verlag, 2017. Eprint 2017/521
https://eprint.iacr.org/2017/521
19. F. Betül Durak. Serge Vaudenay. Generic Round-Function-Recovery Attacks for Feistel Networks over Small Domains. In *Applied Cryptography and Network Security (ACNS'18)*, Leuven, Belgium, Lecture Notes in Computer Science 10892, pp. 440–458, Springer-Verlag, 2018. Eprint 2018/108
https://eprint.iacr.org/2018/108
20. Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, vol. 228 (5), pp. 15–23, 1973.
https://www.jstor.org/stable/10.2307/24923044
21. Louis Granboulan, Eric Levieil, Gilles Piret. Pseudorandom permutation families over Abelian groups. In *Fast Software Encryption'06*, Graz, Austria, Lecture Notes in Computer Science 4047, pp. 57–77, Springer-Verlag, 2006.
22. Lov Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, Philadelphia, Pennsylvania, U.S.A., pp. 212–219, ACM Press, 1996.
23. Viet Tung Hoang, David Miller, Ni Trieu. Attacks Only Get Better: How to Break FF3 on Large Domains. In *Advances in Cryptology EUROCRYPT'19*, Darmstadt, Germany, Lecture Notes in Computer Science 11476–11478, pp. 85–116 vol. 2, Springer-Verlag, 2019. Eprint 2019/244
https://eprint.iacr.org/2019/244
24. Viet Tung Hoang, Stefano Tessaro, Ni Trieu. The Curse of Small Domains: New Attacks on Format-Preserving Encryption. In *Advances in Cryptology CRYPTO'18*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 10991–10993, pp. 221–251 vol. 1, Springer-Verlag, 2018. Eprint 2018/556
https://eprint.iacr.org/2018/556

25. Lars R. Knudsen. Truncated and Higher Order Differentials. In *Fast Software Encryption'94*, Leuven, Belgium, Lecture Notes in Computer Science 1008, pp. 196–211, Springer-Verlag, 1995.
26. Donald E. Knuth. Seminumerical Algorithms — The Art of Computer Programming (3rd ed.). Addison-Wesley. 1998.
27. Daniel Lemire. Fast Random Generation in an Interval. arXiv:1805.10941v4. 2018. `https://arxiv.org/pdf/1805.10941.pdf`
28. Jérémie Lumbroso. Optimal Discrete Uniform Generation from Coin Flips, and Applications. arXiv:1304.1916v1. 2013. `https://arxiv.org/pdf/1304.1916.pdf`
29. Moses Liskov, Ronald L. Rivest, David Wagner. Tweakable Block Ciphers. *Journal of Cryptology*, vol. 24, pp. 588–613, 2011.
30. Michael Luby, Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing*, vol. 17, pp. 373–386, 1988.
31. Ralph C. Merkle. Fast Software Encryption Functions. In *Advances in Cryptology CRYPTO'90*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 537, pp. 477–501, Springer-Verlag, 1991.
32. Phillip Rogaway. A Synopsis of Format Preserving Encryption. 2010. `http://web.cs.ucdavis.edu/~rogaway/papers/synopsis.pdf`
33. Bruce Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In *Fast Software Encryption'93*, Cambridge, United Kingdom, Lecture Notes in Computer Science 809, pp. 191–204, Springer-Verlag, 1994.
34. Richard Schroeppel. The Hasty Pudding Cipher. AES submission. 1998.
35. Daniel R. Simon. On the Power of Quantum Computation. *SIAM Journal on Computing*, vol. 26 (5), pp. 1474–1483, 1997.
36. Terence Spies. Format Preserving Encryption. White paper. 2008. `https://www.voltage.com/wp-content/uploads/Voltage-Security-WhitePaper-Format-Preserving-Encryption.pdf`