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,

[⋆] This is the full version of a paper presented at the ASIACRYPT 2021 conference [17].

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

[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.

functions generated from the secret key. BLOWFISH [33] also used pseudorandom 8-bit to 32-bit functions. In our design, we use pseudorandom 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 pre-generated 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: pre-generated 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.

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

We formally define a strong security model for FPE which is essentially a multi-target chosen format and chosen tweak super-PRP (pseudorandom 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. We let as additional material the formal proofs (Appendix A) and the best known attacks (Appendix B) 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.

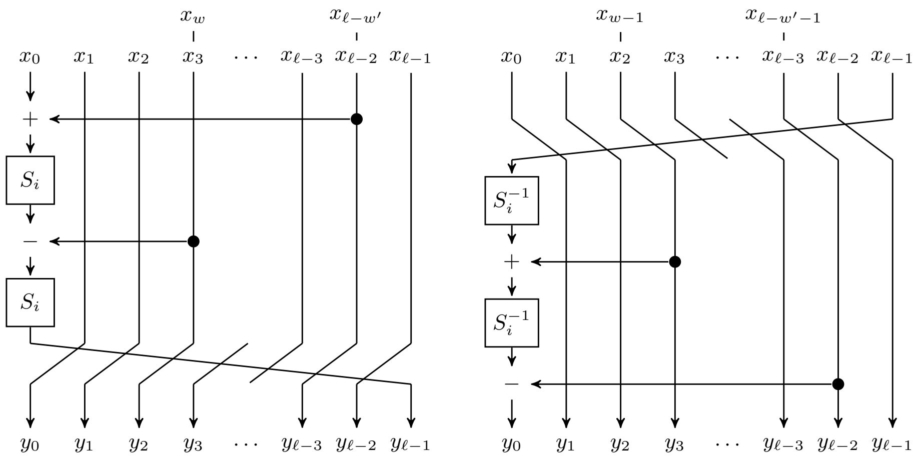

x_w
$x_{\ell-w'}$
x_0 x_1 x_2 x_3 $\cdots$ $x_{\ell-3}$ $x_{\ell-2}$ $x_{\ell-1}$
$+$
S_i
$-$
S_i
y_0 y_1 y_2 y_3 $\cdots$ $y_{\ell-3}$ $y_{\ell-2}$ $y_{\ell-1}$
x_{w-1}
$x_{\ell-w'-1}$
x_0 x_1 x_2 x_3 $\cdots$ $x_{\ell-3}$ $x_{\ell-2}$ $x_{\ell-1}$
S_i^{-1}
$+$
S_i^{-1}
$-$
y_0 y_1 y_2 y_3 $\cdots$ $y_{\ell-3}$ $y_{\ell-2}$ $y_{\ell-1}$

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

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

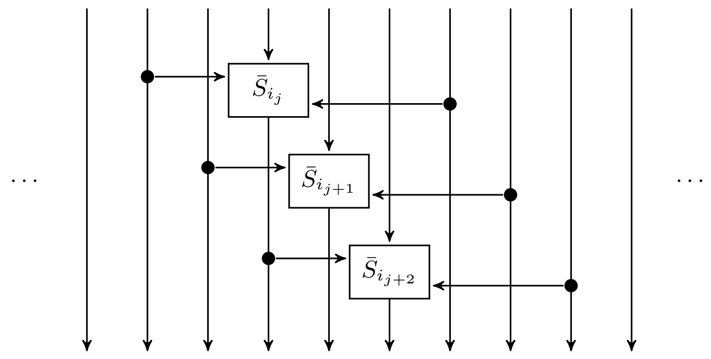


Fig. 2. Three Consecutive Layers with $w = 3$ and $w' = 2$ and without the Circular Shift of Branches (Each $\bar{S}$ represents a double-Sbox)

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

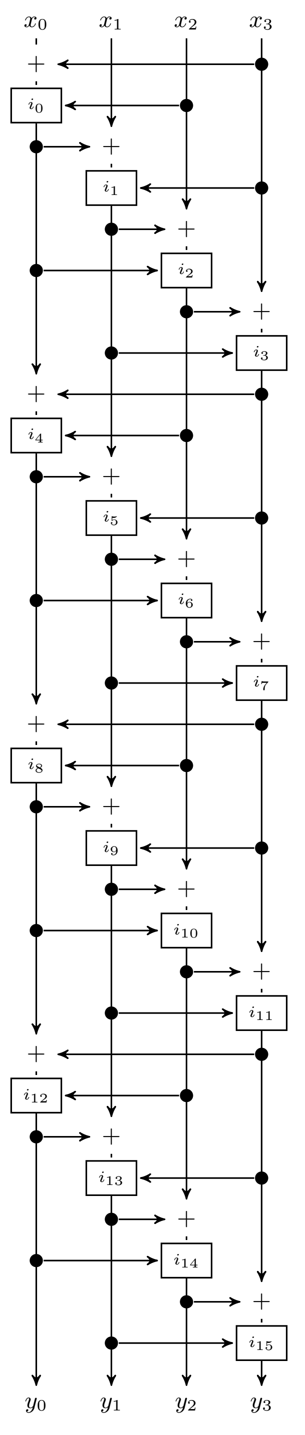
$$\mathsf{CEnc}_S[i_0, \ldots, i_{n-1}] = E_S[i_{n-1}] \circ \cdots \circ E_S[i_0]$$
$$\mathsf{CDec}_S[i_0, \ldots, i_{n-1}] = D_S[i_0] \circ \cdots \circ D_S[i_{n-1}]$$

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:     $z \leftarrow S_{i_j}(S_{i_j}(x_0 + x_{\ell-w'} \bmod a) - x_w \bmod a)$
 3:     for $k = 1$ to $\ell - 1$ do
 4:         $x_{k-1} \leftarrow x_k$
 5:     end for
 6:     $x_{\ell-1} \leftarrow z$
 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:     $z \leftarrow y_{\ell-1}$
11:     for $k = \ell - 1$ down to 1 do
12:         $y_k \leftarrow y_{k-1}$
13:     end for
14:     $y_0 \leftarrow S_{i_j}^{-1}(S_{i_j}^{-1}(z) + y_w \bmod a) - y_{\ell-w'} \bmod a$
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 $\mathsf{SEQ}$
```

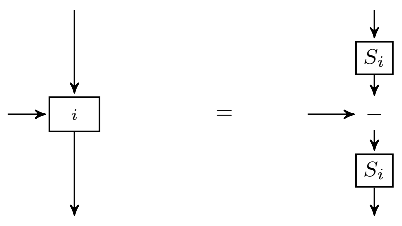


Fig. 3. Core Encryption $\mathsf{CEnc}[i_0, \ldots, i_{15}]$ with $\ell = 4$, $w = 2$, $w' = 1$, $n = 16$

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 "$\mathsf{FPE\ SEQ}$" and the size L_1, and cste_3 is a constant which encodes the label "$\mathsf{FPE\ Pool}$" 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 Appendix C. 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.

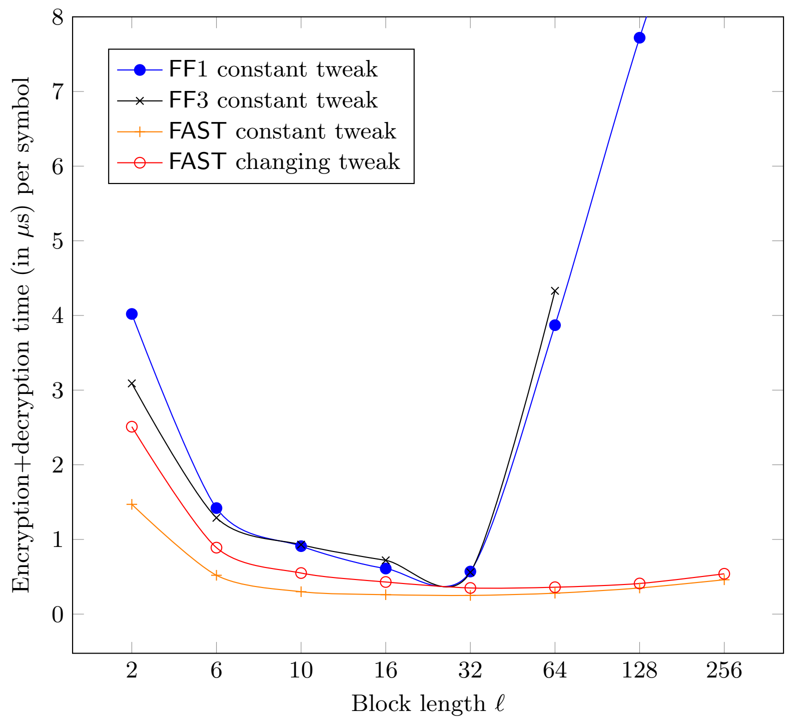


Fig. 4. Implementation Results for $a = 10$ (FF3 limits $\ell \leq 56$)

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,

$$\mathsf{ct} = F_k(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$$
$$\mathsf{pt} = (F_k(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$$

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 $\ell$), $\mathsf{tweak}$,
   $F_K(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ 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 $\mathsf{SEQ}$ 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 $\mathsf{SEQ}$ 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 $\mathsf{SEQ}$ 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 $\mathsf{SEQ}$ 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 $\mathsf{SEQ}$
```

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 $\mathsf{PRF}$
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 Appendix A.

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 $a \bmod 4 = 1$. $\square$

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 Appendix B.

– 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. To appear in ASIACRYPT'21.
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`

A Proof of Th. 4

The $\mathsf{INDstrong}_b$ game with our algorithm and adversary $\mathcal{A}$ defines the following $\Gamma_b^0(\mathcal{A})$ game.

```
Game $\Gamma_b^0(\mathcal{A})$:
 1: pick $F$ following the interface at random
    for each $K$, $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F_K(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ 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:     $(a, m) \leftarrow \mathsf{instance}_1$
11:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
12:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
13:     $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
14:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
15:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
16:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
17: end if

Oracle $\mathsf{OTD}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
18: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
19: if $i \notin \{1, \ldots, \tau\}$ then return $\perp$
20: if $b = 0$ then
21:     return $\left(F_{K[i]}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})\right)^{-1}(\mathsf{ct})$
22: else
23:     $(a, m) \leftarrow \mathsf{instance}_1$
24:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
25:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
26:     $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
27:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
28:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
29:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
30: end if
```

We want to bound the advantage between $b = 1$ and $b = 0$. We have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) = \mathsf{Adv}_{\Gamma^0}(\mathcal{A})$$

$\Gamma^0_b(\mathcal{A})$ to $\Gamma^1_j(\mathcal{A})$: reduction to independent random functions. We define a new game $\Gamma^1_j(\mathcal{A})$.

```
Game $\Gamma^1_j(\mathcal{A})$:
 1: pick $F_1, \ldots, F_\tau$ following the interface at random
    for each $i$, $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ 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 $i > j$ then
 8:     return $F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
 9: else
10:     $(a, m) \leftarrow \mathsf{instance}_1$
11:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
12:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
13:     $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
14:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
15:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
16:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
17: end if

Oracle $\mathsf{OTD}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
18: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
19: if $i \notin \{1, \ldots, \tau\}$ then return $\perp$
20: if $i > j$ then
21:     return $(F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
22: else
23:     $(a, m) \leftarrow \mathsf{instance}_1$
24:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
25:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
26:     $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
27:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
28:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
29:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
30: end if
```

We first introduce minor changes. Instead of an index $b \in \{0,1\}$, we use $j \in \{0, \ldots, \tau\}$ and treat target i depending on $i \leq j$ or not. Clearly, $\Pr[\Gamma^0_1(\mathcal{A}) \to 1] = \Pr[\Gamma^1_\tau(\mathcal{A}) \to 1]$.

The major change is that instead of selecting a random function F mapping a key to a random encryption function, we select τ independent random encryption functions for each target. The distribution of the obtained target functions does not change, except if a collision occurs in the random selection of the $K[i]$. Hence, we have

$$\Pr[\Gamma^1_0(\mathcal{A}) \to 1] - \Pr[\Gamma^0_0(\mathcal{A}) \to 1] \leq \frac{\tau^2}{2} 2^{-L}$$

By triangular inequality, the advantage between Γ^1_0 and Γ^1_τ is bounded by the sum of the advantages between Γ^1_{j-1} and Γ^1_j for $j = 1, \ldots, q$ and the overhead of collision probability.

$$\mathsf{Adv}_{\Gamma^0}(\mathcal{A}) \leq \frac{\tau^2}{2}2^{-L} + \sum_{j=1}^{\tau}\big(\Pr[\Gamma_j^1(\mathcal{A}) \to 1] - \Pr[\Gamma_{j-1}^1(\mathcal{A}) \to 1]\big)$$

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2}2^{-L} + \sum_{j=1}^{\tau}\big(\Pr[\Gamma_j^1(\mathcal{A}) \to 1] - \Pr[\Gamma_{j-1}^1(\mathcal{A}) \to 1]\big)$$

$\Gamma^1_j(\mathcal{A})$ to $\Gamma^2_b(\mathcal{A}_j)$: reduction to single-target. We can now move things in between the game and the adversary and obtain $\Gamma^2_b(\mathcal{A}_j)$.

Game $\Gamma^2_b(\mathcal{A}_j)$:

```
 1: pick $F'$ following the interface at random
    for each $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
 2: pick $K \in \{0,1\}^L$ at random
 3: run $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD}} \to z$
 4: return $z$
```

Oracle $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$

```
 5: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 6: if $b = 0$ then
 7:     return $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
 8: else
 9:     $(a, m) \leftarrow \mathsf{instance}_1$
10:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
11:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
12:     $K_S \leftarrow \mathsf{PRF}^{L_2}(K, \mathsf{instance}_1, \mathsf{cste}_2)$
13:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
14:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
15:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
16: end if
```

Oracle $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$

```
17: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
18: if $b = 0$ then
19:     return $(F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
20: else
21:     $(a, m) \leftarrow \mathsf{instance}_1$
22:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
23:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
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:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
27:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
28: end if
```

Algorithm $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD}}$:

```
 1: pick $F_1, \ldots, F_{j-1}, F_{j+1}, \ldots, F_\tau$ following the interface at random
    for each $i$, $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
    (sample the $F_i$ by lazy sampling)
 2: pick $K[1], \ldots, K[i-1], K[i+1], \ldots, K[\tau] \in \{0,1\}^L$ at random
 3: run $\mathcal{A}^{\mathsf{OTE},\mathsf{OTD}} \to z$
 4: return $z$
```

Subroutine $\mathsf{OTE}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$

```
 5: if i=j then
 6:     return $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
 7: end if
 8: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 9: if $i \notin \{1, \ldots, \tau\}$ then return $\perp$
10: if $i > j$ then
11:     return $F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
12: else
13:     $(a, m) \leftarrow \mathsf{instance}_1$
```

```
    $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
    $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
    $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
    $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
    $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
    return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
end if

Subroutine $\mathsf{OTD}(i, \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
if i=j then
    return $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
end if
if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
if $i \notin \{1, \ldots, \tau\}$ then return $\perp$
if $i > j$ then
    return $(F_i(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
else
    $(a, m) \leftarrow \mathsf{instance}_1$
    $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
    $K_{\mathsf{SEQ}} \leftarrow \mathsf{PRF}^{L_1}(K[i], \mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
    $K_S \leftarrow \mathsf{PRF}^{L_2}(K[i], \mathsf{instance}_1, \mathsf{cste}_2)$
    $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
    $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
    return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
end if
```

Let the F_i and $K[i]$ as selected by $\mathcal{A}_j$ for $i \neq j$ and F' and K as selected by Γ_b^2. We define $F_j = F'$ and $K[j] = K$. We easily check that the OTE and OTD subroutines in $\mathcal{A}_j$ playing Γ_b^2 return the same output as the OTE and OTD oracles in $\Gamma_{j+b}^1(\mathcal{A})$. Hence,

$$\Pr[\Gamma_j^1(\mathcal{A}) \to 1] - \Pr[\Gamma_{j-1}^1(\mathcal{A}) \to 1] = \mathsf{Adv}_{\Gamma^2}(\mathcal{A}_j)$$

We obtain an adversary $\mathcal{A}_j$ playing a game Γ^2 with a single target K.

The complexity of $\mathcal{A}_j$ is the complexity of $\mathcal{A}$ with an additional overhead bounded by $c\tau$, where c bounds the complexity of the lazy sampling and the complexity of the encryption/decryption algorithms. Using the same random coins, the sum of the number of queries made by $\mathcal{A}_j$ for $j = 1, \ldots, \tau$ is still bounded by q on average.

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \sum_{j=1}^{\tau} \mathsf{Adv}_{\Gamma^2}(\mathcal{A}_j)$$

$\Gamma_b^2(\mathcal{A}_j)$ to $\Gamma_b^3(\mathcal{A}_j)$: reduction with no PRF*.* We make $\Gamma_b^3(\mathcal{A}_j)$, a variant of $\Gamma_b^2(\mathcal{A}_j)$ which no longer use PRF but rather a random function. We define $\mathcal{B}_j$ playing the PRF security game.

Game $\Gamma_b^3(\mathcal{A}_j)$:

```
 1: pick $F'$ following the interface at random
    for each $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
 2: pick a random function $\mathsf{RF}$ with same input/output domain as $\mathsf{PRF}$
 3: run $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD}} \to z$
 4: return $z$
```

Oracle $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$

```
 5: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 6: if $b = 0$ then
 7:     return $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
 8: else
 9:     $(a, m) \leftarrow \mathsf{instance}_1$
10:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
11:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
12:     $K_S \leftarrow \mathsf{RF}^{L_2}(\mathsf{instance}_1, \mathsf{cste}_2)$
13:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
14:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
15:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
16: end if
```

Oracle $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$

```
17: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
18: if $b = 0$ then
19:     return $(F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
20: else
21:     $(a, m) \leftarrow \mathsf{instance}_1$
22:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
23:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
24:     $K_S \leftarrow \mathsf{RF}^{L_2}(\mathsf{instance}_1, \mathsf{cste}_2)$
25:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
26:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
27:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
28: end if
```

Game $\mathsf{PRF}^b(\mathcal{B}_j)$:

```
 1: pick a random function $\mathsf{RF}$ with same input/output domain as $\mathsf{PRF}$
 2: pick $K$ at random
 3: run $\mathcal{B}_j^{\mathsf{OF}} \to z$
 4: return $z$
```

Oracle $\mathsf{OF}(x)$

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

Algorithm $\mathcal{B}_j^{\mathsf{OF}}$:

```
10: run $\mathcal{A}_j^{\mathsf{OSE}',\mathsf{OSD}'} \to z$
11: return $z$
```

Subroutine $\mathsf{OSE}'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$

```
12: **if** $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ **then return** $\perp$
13: $(a, m) \leftarrow \mathsf{instance}_1$
14: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
15: $K_{\mathsf{SEQ}} \leftarrow \mathsf{OF}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
16: $K_S \leftarrow \mathsf{OF}(\mathsf{instance}_1, \mathsf{cste}_2)$
17: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
18: $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
19: **return** $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$

Subroutine $\mathsf{OSD}'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
20: **if** $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ **then return** $\perp$
21: $(a, m) \leftarrow \mathsf{instance}_1$
22: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
23: $K_{\mathsf{SEQ}} \leftarrow \mathsf{OF}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
24: $K_S \leftarrow \mathsf{OF}(\mathsf{instance}_1, \mathsf{cste}_2)$
25: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
26: $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
27: **return** $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
```

Clearly, this modification has no impact for $b = 0$ in $\Gamma_b^2(\mathcal{A}_j)$ and $\Gamma_b^3(\mathcal{A}_j)$ as neither PRF nor RF is used: $\Gamma_0^2(\mathcal{A}_j)$ and $\Gamma_0^3(\mathcal{A}_j)$ are identical. The game $\Gamma_1^2(\mathcal{A}_j)$ is identical to the game $\mathsf{PRF}^1(\mathcal{B}_j)$. The game $\Gamma_1^3(\mathcal{A}_j)$ is identical to the game $\mathsf{PRF}^0(\mathcal{B}_j)$. Hence, we obtain

$$\mathsf{Adv}_{\Gamma^2}(\mathcal{A}_j) \leq \mathsf{Adv}_{\Gamma^3}(\mathcal{A}_j) + \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}_j)$$

The complexity of $\mathcal{B}_j$ is the complexity of $\mathcal{A}_j$ with an additional overhead bounded by cq for some small constant c. The number of queries is still bounded by q. In the theorem, we take $\mathcal{B}$ as the $\mathcal{B}_j$ with largest advantage $\mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}_j)$.

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \sum_{j=1}^{\tau} \mathsf{Adv}_{\Gamma^3}(\mathcal{A}_j)$$

$\Gamma_b^3(\mathcal{A}_j)$ to $\Gamma_b^4(\mathcal{A}_j)$: reduction to known-$S$ attacks. We define the following game:

```
Game $\Gamma_b^4(\mathcal{A}_j)$:
 1: pick $F'$ following the interface at random
    for each $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
 2: pick a random function $\mathsf{RF}$ with same input/output domain as $\mathsf{PRF}$
 3: run $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD},\mathsf{OT}} \to z$
 4: return $z$

Oracle $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
 5: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 6: if $b = 0$ then
 7:     return $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
 8: else
 9:     $(a, m) \leftarrow \mathsf{instance}_1$
10:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
11:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
12:     $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
13:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
14:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
15: end if

Oracle $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
16: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
17: if $b = 0$ then
18:     return $(F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
19: else
20:     $(a, m) \leftarrow \mathsf{instance}_1$
21:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
22:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
23:     $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
24:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
25:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
26: end if

Oracle $\mathsf{OT}(\mathsf{instance}_1)$
27: $(a, m) \leftarrow \mathsf{instance}_1$
28: $K_S \leftarrow \mathsf{RF}^{L_2}(\mathsf{instance}_1, \mathsf{cste}_2)$
29: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
30: return $S$
```

Here, we only introduced a new orale OT (that $\mathcal{A}_j$ does not use) which reveals S. Since this oracle has no use here, the advantage is unchanged:

$$\mathsf{Adv}_{\Gamma^3}(\mathcal{A}_j) = \mathsf{Adv}_{\Gamma^4}(\mathcal{A}_j)$$

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \sum_{j=1}^{\tau} \mathsf{Adv}_{\Gamma^4}(\mathcal{A}_j)$$

$\Gamma^4_b(\mathcal{A}_j)$ to $\Gamma^5_{j'}(\mathcal{A}_j)$: reduction to hybrids for single-$\mathsf{instance}_1$. We define the following game:

```
Game $\Gamma^5_{j'}(\mathcal{A}_j)$:
 1: pick $F'$ following the interface at random
    for each $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
 2: pick a random function RF with same input/output domain as PRF
 3: initialize an array $T$ to empty
 4: run $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD},\mathsf{OT}} \to z$
 5: return $z$
```

```
Oracle $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
 6: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 7: if $T(\mathsf{instance}_1)$ undefined then
 8:     $T(\mathsf{instance}_1) \leftarrow |T| + 1$
 9: end if
10: if $T(\mathsf{instance}_1) > j'$ then
11:     return $F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
12: else
13:     $(a, m) \leftarrow \mathsf{instance}_1$
14:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
15:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
16:     $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
17:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
18:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
19: end if
```

```
Oracle $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
20: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
21: if $T(\mathsf{instance}_1)$ undefined then
22:     $T(\mathsf{instance}_1) \leftarrow |T| + 1$
23: end if
24: if $T(\mathsf{instance}_1) > j'$ then
25:     return $(F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
26: else
27:     $(a, m) \leftarrow \mathsf{instance}_1$
28:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
29:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
30:     $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
31:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
32:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
33: end if
```

```
Oracle $\mathsf{OT}(\mathsf{instance}_1)$
34: $(a, m) \leftarrow \mathsf{instance}_1$
35: $K_S \leftarrow \mathsf{RF}^{L_2}(\mathsf{instance}_1, \mathsf{cste}_2)$
36: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
37: return $S$
```

We now keep track of which $\mathsf{instance}_1$ value has been used and number those instances in an array $T(\mathsf{instance}_1)$. We treat instances like for $b = 0$ if their number are larger than j' and like $b = 1$ otherwise. Clearly, $\Pr[\Gamma^4_0(\mathcal{A}_j) \to 1] = \Pr[\Gamma^5_0(\mathcal{A}_j) \to 1]$ and $\Pr[\Gamma^4_1(\mathcal{A}_j) \to 1] = \Pr[\Gamma^5_{\tau_j}(\mathcal{A}_j) \to$

1]. Hence,

$$\mathsf{Adv}_{\mathit{\Gamma}^4}(\mathcal{A}_j) = \sum_{j'=1}^{\tau_j} \Pr[\mathit{\Gamma}^5_{j'}(\mathcal{A}_j) \to 1] - \Pr[\mathit{\Gamma}^5_{j'-1}(\mathcal{A}_j) \to 1]$$

So far, we have

$$\begin{aligned}\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq & \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \\ & \sum_{j=1}^{\tau} \sum_{j'=1}^{\tau_j} \left(\Pr[\mathit{\Gamma}^5_{j'}(\mathcal{A}_j) \to 1] - \Pr[\mathit{\Gamma}^5_{j'-1}(\mathcal{A}_j) \to 1] \right)\end{aligned}$$

$\Gamma^5_{j'}(\mathcal{A}_j)$ to $\Gamma^6_b(\mathcal{A}_{j,j'})$: reduction to single-$\mathsf{instance}_1$. We define the following game in which OT' is used only once:

```
Game $\Gamma^6_b(\mathcal{A}_{j,j'})$:
 1: pick a random function $\mathsf{RF}_2$ with same input/output domain as $\mathsf{PRF}$
 2: run $\mathcal{A}_{j,j'}^{\mathsf{OSE}',\mathsf{OSD}',\mathsf{OT}'} \to z$
 3: return $z$

Oracle $\mathsf{OSE}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
 4: if $S$ undefined then return $\bot$
 5: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\bot$
 6: if $b = 0$ then
 7:     return $F_2(\mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
 8: else
 9:     $(a, m) \leftarrow \mathsf{instance}_1$
10:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
11:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2^{L_1}(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
12:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
13:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
14: end if

Oracle $\mathsf{OSD}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
15: if $S$ undefined then return $\bot$
16: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\bot$
17: if $b = 0$ then
18:     return $(F(\mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
19: else
20:     $(a, m) \leftarrow \mathsf{instance}_1$
21:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
22:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2^{L_1}(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
23:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
24:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
25: end if

Oracle $\mathsf{OT}'(\mathsf{instance}_1)$
26: if $S$ defined then return $\bot$
27: $(a, m) \leftarrow \mathsf{instance}_1$
28: pick $F_2$ following the interface at random for each $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$, $F_2(\mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
29: pick $K_S \in \{0,1\}^{L_2}$ at random
30: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
31: return $S$

Algorithm $\mathcal{A}_{j,j'}^{\mathsf{OSE}',\mathsf{OSD}'\mathsf{OT}'}$:
32: pick $F_1$ following the interface at random for each $\mathsf{instance}_1$ (including $a$), $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$, $F_1(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
    (sample the $F_1$ by lazy sampling)
33: pick $K \in \{0,1\}^L$ at random
34: pick a random function $\mathsf{RF}_1$ with same input/output domain as $\mathsf{PRF}$
    (sample the $\mathsf{RF}_1$ by lazy sampling)
35: initialize an array $T$ to empty
36: run $\mathcal{A}_j^{\mathsf{OSE},\mathsf{OSD},\mathsf{OT}} \to z$
37: return $z$

Sub. $\mathsf{OSE}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
38: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\bot$
39: $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
40: if $T(\mathsf{instance}_1) > j'$ then
41:     return $F_1(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
42: else if $T(\mathsf{instance}_1) = j'$ then
43:     return $\mathsf{OSE}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
44: else
45:     $(a, m) \leftarrow \mathsf{instance}_1$
46:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
47:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_1^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
48:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
49:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
50: end if

Sub. $\mathsf{OSD}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
51: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\bot$
52: $S \leftarrow \mathsf{OT}(\mathsf{instance}_1)$
53: if $T(\mathsf{instance}_1) > j'$ then
54:     return $(F_1(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
55: else if $T(\mathsf{instance}_1) = j'$ then
56:     return $\mathsf{OSD}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
57: else
58:     $(a, m) \leftarrow \mathsf{instance}_1$
59:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
60:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_1^{L_1}(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
61:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
62:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
63: end if

Subroutine $\mathsf{OT}(\mathsf{instance}_1)$
64: if $T(\mathsf{instance}_1)$ undefined then
65:     $T(\mathsf{instance}_1) \leftarrow |T| + 1$
```

```
66:     if $|T| = j'$ then
67:         $S' \leftarrow \mathsf{OT}'(\mathsf{instance}_1)$
68:     end if
69: end if
70: if $T(\mathsf{instance}_1) = j'$ then return $S'$
71: $K_S \leftarrow \mathsf{RF}_1^{L_2}(\mathsf{instance}_1, \mathsf{cste}_2)$
72: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
73: return $S$
```

$\Gamma^5_{j'+b}(\mathcal{A}_j)$ and $\Gamma^6_b(\mathcal{A}_{j,j'})$ are supposed to be the same. To see this, we define RF and F' based on RF_1, RF_2, F_1, F_2, and the value u for $\mathsf{instance}_1$ and v for K_S which are defined by the one-time oracle OT' in Γ^6_b. Note that this u is characterized by $T(u) = j'$. We define

$$\mathsf{RF}(\mathsf{instance}_1, \mathsf{str}) = \begin{cases} \mathsf{RF}_1(\mathsf{instance}_1, \mathsf{str}) & \text{if } \mathsf{instance}_1 \neq u \\ \mathsf{RF}_2(\mathsf{instance}_2, \mathsf{str}) & \text{if } \mathsf{instance}_1 = u \text{ and } \mathsf{str} \neq \mathsf{cste}_2 \\ v & \text{if } \mathsf{instance}_1 = u \text{ and } \mathsf{str} = \mathsf{cste}_2 \end{cases}$$

We obtain RF with same distribution as in Γ^5. Similarly, we define

$$F'(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}) = \begin{cases} F_1(\mathsf{instance}_1, \mathsf{instance}_2, \mathsf{tweak}) & \text{if } \mathsf{instance}_1 \neq u \\ F_2(\mathsf{instance}_2, \mathsf{tweak}) & \text{if } \mathsf{instance}_1 = u \end{cases}$$

We obtain F' with same distribution as in Γ^5.

With this definition, we can see that the subroutines OT, OSE, and OSD in $\mathcal{A}_{j,j'}$ playing Γ^6_b are returning the same thing as the oracles OT, OSE, and OSD in Γ^5. Hence, we have

$$\Pr[\Gamma^5_{j'}(\mathcal{A}_j) \to 1] - \Pr[\Gamma^5_{j'-1}(\mathcal{A}_j) \to 1] = \mathsf{Adv}_{\Gamma^6}(\mathcal{A}_{j,j'})$$

The complexity of $\mathcal{A}_{j,j'}$ is still comparable to the complexity of $\mathcal{A}$ (with a small overhead). Furthermore, with the same coins, the sum of the number of queries made by $\mathcal{A}_{j,j'}$ for $j' = 1, \ldots, \tau_j$ is the same as the one by $\mathcal{A}_j$.

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \sum_{j=1}^{\tau} \sum_{j'=1}^{\tau_j} \mathsf{Adv}_{\Gamma^6}(\mathcal{A}_{j,j'})$$

$\Gamma^6_b(\mathcal{A}_{j,j'})$ to $\Gamma^7_{j''}(\mathcal{A}_{j,j'})$: reduction to hybrids for single-instance single-tweak. We define the following game:

```
Game $\Gamma^7_{j''}(\mathcal{A}_{j,j'})$:
 1: pick a random function $\mathsf{RF}_2$ with same input/output domain as $\mathsf{PRF}$
 2: initialize an array $T$ to empty
 3: run $\mathcal{A}_{j,j'}^{\mathsf{OSE}',\mathsf{OSD}',\mathsf{OT}'} \to z$
 4: return $z$

Oracle $\mathsf{OSE}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
 5: if $S$ undefined then return $\perp$
 6: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
 7: if $T(\mathsf{instance}_2, \mathsf{tweak})$ undefined then
 8:     $T(\mathsf{instance}_2, \mathsf{tweak}) \leftarrow |T| + 1$
 9: end if
10: if $T(\mathsf{instance}_2, \mathsf{tweak}) > j''$ then
11:     return $F_2(\mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
12: else
13:     $(a, m) \leftarrow \mathsf{instance}_1$
14:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
15:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
16:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
17:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
18: end if

Oracle $\mathsf{OSD}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
19: if $S$ undefined then return $\perp$
20: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
21: if $T(\mathsf{instance}_2, \mathsf{tweak})$ undefined then
22:     $T(\mathsf{instance}_2, \mathsf{tweak}) \leftarrow |T| + 1$
23: end if
24: if $T(\mathsf{instance}_2, \mathsf{tweak}) > j''$ then
25:     return $(F(\mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
26: else
27:     $(a, m) \leftarrow \mathsf{instance}_1$
28:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
29:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
30:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
31:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
32: end if

Oracle $\mathsf{OT}'(\mathsf{instance}_1)$
33: if $S$ defined then return $\perp$
34: $(a, m) \leftarrow \mathsf{instance}_1$
35: pick $F_2$ following the interface at random for each $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$, $F_2(\mathsf{instance}_2, \mathsf{tweak})$ is an even permutation of $\mathbf{Z}_a^\ell$
36: pick $K_S \in \{0,1\}^{L_2}$ at random
37: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
38: return $S$
```

Again, we obtain

$$\mathsf{Adv}_{\Gamma^6}(\mathcal{A}_{j,j'}) = \sum_{j''=1}^{\tau_{j,j'}} \left(\Pr[\Gamma^6_{j''}(\mathcal{A}_{j,j'}) \to 1] - \Pr[\Gamma^6_{j''-1}(\mathcal{A}_{j,j'}) \to 1]\right)$$

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \sum_{j=1}^{\tau} \sum_{j'=1}^{\tau_j} \sum_{j''=1}^{\tau_{j,j'}} \left(\Pr[\Gamma^6_{j''}(\mathcal{A}_{j,j'}) \to 1] - \Pr[\Gamma^6_{j''-1}(\mathcal{A}_{j,j'}) \to 1]\right)$$

$\Gamma^7_{j''}(\mathcal{A}_{j,j'})$ to $\Gamma^8_b(\mathcal{A}_{j,j',j''})$: reduction to single-instance single-tweak. We define the following game Γ^8_b and adversary $\mathcal{A}_{j,j',j''}$:

```
Game $\Gamma^8_b(\mathcal{A}_{j,j',j''})$:
 1: run $\mathcal{A}_{j,j',j''}^{\mathsf{OE},\mathsf{OD},\mathsf{OT}_1,\mathsf{OT}_2} \to z$
 2: return $z$

Oracle $\mathsf{OE}(\mathsf{pt})$
 3: if $S$ or $\mathsf{SEQ}$ undefined then return $\perp$
 4: if $b = 0$ then
 5:     return $F(\mathsf{pt})$
 6: else
 7:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
 8: end if

Oracle $\mathsf{OD}(\mathsf{ct})$
 9: if $S$ or $\mathsf{SEQ}$ undefined then return $\perp$
10: if $b = 0$ then
11:     return $F^{-1}(\mathsf{ct})$
12: else
13:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
14: end if

Oracle $\mathsf{OT}_1(\mathsf{instance}_1)$
15: if $S$ defined then return $\perp$
16: $(a, m) \leftarrow \mathsf{instance}_1$
17: pick $K_S \in \{0,1\}^{L_2}$ at random
18: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
19: return $S$

Oracle $\mathsf{OT}_2(\mathsf{instance}_2)$
20: if $\mathsf{SEQ}$ defined then return $\perp$
21: if $S$ undefined then return $\perp$
22: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return
    $\perp$
23: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
24: pick a random even permutation $F$ of $\mathbf{Z}_a^\ell$
25: pick $K_{\mathsf{SEQ}} \in \{0,1\}^{L_1}$ at random
26: $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
27: return

Algorithm $\mathcal{A}_{j,j',j''}^{\mathsf{OE},\mathsf{OD},\mathsf{OT}_1,\mathsf{OT}_2}$:
28: pick a random function $\mathsf{RF}_2$ with same
    input/output domain as $\mathsf{PRF}$
    (sample the $\mathsf{RF}_2$ by lazy sampling)
29: initialize an array $T$ to empty
30: $\mathcal{A}_{j,j'}^{\mathsf{OSE}',\mathsf{OSD}',\mathsf{OT}'} \to z$
31: return $z$

Subroutine $\mathsf{OSE}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{pt})$
32: if $S$ undefined then return $\perp$
33: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return
    $\perp$
34: if $T(\mathsf{instance}_2, \mathsf{tweak})$ undefined then
35:     $T(\mathsf{instance}_2, \mathsf{tweak}) \leftarrow |T| + 1$
36:     if $|T| = j''$ then $\mathsf{OT}_2(\mathsf{instance}_2)$
37: end if
38: if $T(\mathsf{instance}_2, \mathsf{tweak}) > j''$ then
39:     return $F_2(\mathsf{instance}_2, \mathsf{tweak})(\mathsf{pt})$
40: else if $T(\mathsf{instance}_2, \mathsf{tweak}) = j''$ then
41:     return $\mathsf{OE}(\mathsf{pt})$
42: else
43:     $(a, m) \leftarrow \mathsf{instance}_1$
44:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
45:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
46:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
47:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
48: end if

Subroutine $\mathsf{OSD}'(\mathsf{instance}_2, \mathsf{tweak}, \mathsf{ct})$
49: if $S$ undefined then return $\perp$
50: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return
    $\perp$
51: if $T(\mathsf{instance}_2, \mathsf{tweak})$ undefined then
52:     $T(\mathsf{instance}_2, \mathsf{tweak}) \leftarrow |T| + 1$
53:     if $|T| = j''$ then $\mathsf{OT}_2(\mathsf{instance}_2)$
54: end if
55: if $T(\mathsf{instance}_2, \mathsf{tweak}) > j''$ then
56:     return $(F(\mathsf{instance}_2, \mathsf{tweak}))^{-1}(\mathsf{ct})$
57: else if $T(\mathsf{instance}_2, \mathsf{tweak}) = j''$ then
58:     return $\mathsf{OD}(\mathsf{ct})$
59: else
60:     $(a, m) \leftarrow \mathsf{instance}_1$
61:     $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
62:     $K_{\mathsf{SEQ}} \leftarrow \mathsf{RF}_2(\mathsf{instance}_2, \mathsf{cste}_1, \mathsf{tweak})$
63:     $\mathsf{SEQ} \leftarrow \mathsf{PRNG}_{1,m,n}(K_{\mathsf{SEQ}})$
64:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
65: end if

Oracle $\mathsf{OT}'(\mathsf{instance}_1)$
66: if $S$ defined then return $\perp$
67: $(a, m) \leftarrow \mathsf{instance}_1$
68: pick $F_2$ following the interface at random
    for each $\mathsf{instance}_2$ (including $\ell$), $\mathsf{tweak}$,
    $F_2(\mathsf{instance}_2, \mathsf{tweak})$ is an even permuta-
    tion of $\mathbf{Z}_a^\ell$
    (sample the $F_2$ by lazy sampling)
69: $S \leftarrow \mathsf{OT}_1(\mathsf{instance}_1)$
70: return $S$
```

Again, we have

$$\Pr[\Gamma^6_{j''}(\mathcal{A}_{j,j'}) \to 1] - \Pr[\Gamma^6_{j''-1}(\mathcal{A}_{j,j'}) \to 1] = \mathsf{Adv}_{\Gamma^8}(\mathcal{A}_{j,j',j''})$$

The complexity of $\mathcal{A}_{j,j',j''}$ is still comparable to the complexity of $\mathcal{A}$ (with a small overhead). Furthermore, with the same coins, the sum of the number of queries made by $\mathcal{A}_{j,j',j''}$ for $j'' = 1, \ldots, \tau_{j,j'}$ is the same as the one by $\mathcal{A}_{j,j'}$. If $q_{j,j',j''}$ is the average number of queries of $\mathcal{A}_{j,j',j''}$, we thus have $\sum_{j,j',j''} q_{j,j',j''} \leq q$.

So far, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + \sum_{j=1}^{\tau} \sum_{j'=1}^{\tau_j} \sum_{j''=1}^{\tau_{j,j'}} \mathsf{Adv}_{\Gamma^8}(\mathcal{A}_{j,j',j''})$$

$\Gamma_b^8(\mathcal{A}_{j,j',j''})$ to $\Gamma_b^9(\mathcal{A}_{j,j',j''})$: reduction to no PRNG_1. We define the following game Γ_b^9 and the adversary $\mathcal{C}_{j,j',j''}$ playing the $\mathsf{INDPRNG}_1^b$ game:

```
Game $\Gamma_b^9(\mathcal{A}_{j,j',j''})$:
1: run $\mathcal{A}_{j,j',j''}^{\mathsf{OE},\mathsf{OD},\mathsf{OT}_1,\mathsf{OT}_2} \to z$
2: return $z$

Oracle OE(pt)
3: if $S$ or SEQ undefined then return $\perp$
4: if $b = 0$ then
5:     return $F(\mathsf{pt})$
6: else
7:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
8: end if

Oracle OD(ct)
9: if $S$ or SEQ undefined then return $\perp$
10: if $b = 0$ then
11:     return $F^{-1}(\mathsf{ct})$
12: else
13:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
14: end if

Oracle $\mathsf{OT}_1(\mathsf{instance}_1)$
15: if $S$ defined then return $\perp$
16: $(a, m) \leftarrow \mathsf{instance}_1$
17: pick $K_S \in \{0,1\}^{L_2}$ at random
18: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
19: return $S$

Oracle $\mathsf{OT}_2(\mathsf{instance}_2)$
20: if SEQ defined then return $\perp$
21: if $S$ undefined then return $\perp$
22: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return
    $\perp$
23: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
24: pick a random even permutation $F$ of $\mathbf{Z}_a^\ell$
25: pick random $\mathsf{SEQ} \in \{0, \dots, m-1\}^n$
26: return
```

```
Game $\mathsf{INDPRNG}_1^b(\mathcal{C}_{j,j',j''})$:
1: run $\mathcal{C}_{j,j',j''}^{\mathsf{OG}} \to z$
2: return $z$

Oracle $\mathsf{OG}(m, n)$
3: if SEQ defined then return $\perp$
4: if $b = 0$ then
5:     pick random $\mathsf{SEQ} \in \{0, \dots, m-1\}^n$
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

Algorithm $\mathcal{C}_{j,j',j''}^{\mathsf{OG}}$:
11: run $\mathcal{A}_{j,j',j''}^{\mathsf{OE}',\mathsf{OD}',\mathsf{OT}_1',\mathsf{OT}_2'} \to z$
12: return $z$

Subroutine OE'(pt)
13: if $S$ or SEQ undefined then return $\perp$
14: return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$

Subroutine OD'(ct)
15: if $S$ or SEQ undefined then return $\perp$
16: return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$

Subroutine $\mathsf{OT}_1'(\mathsf{instance}_1)$
17: if $S$ defined then return $\perp$
18: $(a, m) \leftarrow \mathsf{instance}_1$
19: pick $K_S \in \{0,1\}^{L_2}$ at random
20: $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
21: return $S$

Subroutine $\mathsf{OT}_2'(\mathsf{instance}_2)$
22: if SEQ defined then return $\perp$
23: if $S$ undefined then return $\perp$
24: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return
    $\perp$
25: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
26: $\mathsf{SEQ} \leftarrow \mathsf{OG}(m, n)$
27: return
```

We have that $\Gamma_0^8(\mathcal{A}_{j,j',j''})$ and $\Gamma_0^9(\mathcal{A}_{j,j',j''})$ produce the same output. The same goes for $\Gamma_1^8(\mathcal{A}_{j,j',j''})$ and $\mathsf{INDPRNG}_1^1(\mathcal{C}_{j,j',j''})$, and also for $\Gamma_1^9(\mathcal{A}_{j,j',j''})$ and $\mathsf{INDPRNG}_1^0(\mathcal{C}_{j,j',j''})$. Hence,

$$\mathsf{Adv}_{\Gamma^8}(\mathcal{A}_{j,j',j''}) = \mathsf{Adv}_{\Gamma^9}(\mathcal{A}_{j,j',j''}) + \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}_{j,j',j''})$$

If $q_{j,j',j''} = 0$, then $\mathcal{C}_{j,j',j''}$ makes no OG query so the adversary obtains no information and we have $\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}_{j,j',j''}) = 0$. Among all (j, j', j'') such that $q_{j,j',j''} > 0$, we take $\mathcal{C}$ as $\mathcal{C}_{j,j',j''}$ such that $\frac{\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}_{j,j',j''})}{q_{j,j',j''}}$

is the largest. Hence, for all (j, j', j''), we have $\mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}_{j,j',j''}) \leq q_{j,j',j''} \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C})$. By summing over all (j, j', j''), we obtain

$$\sum_{j,j',j''} \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}_{j,j',j''}) \leq q \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C})$$

So far, we have

$$\begin{aligned}\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq & \frac{\tau^2}{2} 2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + q \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}) + \\ & \sum_{j=1}^{\tau} \sum_{j'=1}^{\tau_j} \sum_{j''=1}^{\tau_{j,j'}} \mathsf{Adv}_{\Gamma^9}(\mathcal{A}_{j,j',j''})\end{aligned}$$

$\mathit{\Gamma}^9_b(\mathcal{A}_{j,j',j''})$ to $\mathsf{INDKSsprp}_b(\mathcal{A}')$: reduction to no PRNG_2. We define the following adversary $\mathcal{D}_{j,j',j''}$ playing the $\mathsf{INDPRNG}^b_2$ game:

```
Game INDKSsprp_b($\mathcal{A}_{j,j',j''}$):
 1: run $\mathcal{A}^{\mathsf{OE},\mathsf{OD},\mathsf{OT}_1,\mathsf{OT}_2}_{j,j',j''} \to z$
 2: return $z$

Oracle OE(pt)
 3: if $S$ or SEQ undefined then return $\perp$
 4: if $b = 0$ then
 5:     return $F(\mathsf{pt})$
 6: else
 7:     return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$
 8: end if

Oracle OD(ct)
 9: if $S$ or SEQ undefined then return $\perp$
10: if $b = 0$ then
11:     return $F^{-1}(\mathsf{ct})$
12: else
13:     return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$
14: end if

Oracle $\mathsf{OT}_1(\mathsf{instance}_1)$
15: if $S$ defined then return $\perp$
16: $(a, m) \leftarrow \mathsf{instance}_1$
17: pick $S_0, \ldots, S_{m-1}$, random $\mathbf{Z}_a$ permutations
18: $S \leftarrow (S_0, \ldots, S_{m-1})$
19: return $S$

Oracle $\mathsf{OT}_2(\mathsf{instance}_2)$
20: if SEQ defined then return $\perp$
21: if $S$ undefined then return $\perp$
22: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
23: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
24: pick a random even permutation $F$ of $\mathbf{Z}^\ell_a$
25: pick random $\mathsf{SEQ} \in \{0, \ldots, m-1\}^n$
26: return
```

```
Game $\mathsf{INDPRNG}^b_2(\mathcal{D}_{j,j',j''})$:
 1: run $\mathcal{D}^{\mathsf{OG}}_{j,j',j''} \to z$
 2: return $z$

Oracle OG$(a, m)$
 3: if $S$ defined then return $\perp$
 4: if $b = 0$ then
 5:     pick $S_0, \ldots, S_{m-1}$, random $\mathbf{Z}_a$ permutations
 6:     $S \leftarrow (S_0, \ldots, S_{m-1})$
 7: else
 8:     pick $K_S \in \{0,1\}^{L_2}$ at random
 9:     $S \leftarrow \mathsf{PRNG}_{2,a,m}(K_S)$
10: end if
11: return $S$

Algorithm $\mathcal{D}_{j,j',j''}$:
12: run $\mathcal{A}^{\mathsf{OE}',\mathsf{OD}',\mathsf{OT}'_1,\mathsf{OT}'_2}_{j,j',j''} \to z$
13: return $z$

Subroutine OE'(pt)
14: if $S$ or SEQ undefined then return $\perp$
15: return $\mathsf{CEnc}_S[\mathsf{SEQ}](\mathsf{pt})$

Subroutine OD'(ct)
16: if $S$ or SEQ undefined then return $\perp$
17: return $\mathsf{CDec}_S[\mathsf{SEQ}](\mathsf{ct})$

Subroutine $\mathsf{OT}'_1(\mathsf{instance}_1)$
18: if $S$ defined then return $\perp$
19: $(a, m) \leftarrow \mathsf{instance}_1$
20: $S \leftarrow \mathsf{OG}(a, m)$
21: return $S$

Subroutine $\mathsf{OT}'_2(\mathsf{instance}_2)$
22: if SEQ defined then return $\perp$
23: if $S$ undefined then return $\perp$
24: if $(\mathsf{instance}_1, \mathsf{instance}_2) \notin \mathcal{F}$ then return $\perp$
25: $(\ell, n, w, w') \leftarrow \mathsf{instance}_2$
26: pick a random even permutation $F$ of $\mathbf{Z}^\ell_a$
27: pick random $\mathsf{SEQ} \in \{0, \ldots, m-1\}^n$
28: return
```

We have that $\mathit{\Gamma}^9_0(\mathcal{A}_{j,j',j''})$ and $\mathsf{INDKSsprp}_0(\mathcal{A}_{j,j',j''})$ produce the same output. The same goes for $\mathit{\Gamma}^9_1(\mathcal{A}_{j,j',j''})$ and $\mathsf{INDPRNG}^1_2(\mathcal{D}_{j,j',j''})$, and for $\mathsf{INDKSsprp}_1(\mathcal{A}_{j,j',j''})$ and $\mathsf{INDPRNG}^0_2(\mathcal{D}_{j,j',j''})$. Hence,

$$\mathsf{Adv}_{\mathit{\Gamma}^9}(\mathcal{A}_{j,j',j''}) = \mathsf{Adv}_{\mathsf{INDKSsprp}}(\mathcal{A}_{j,j',j''}) + \mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D}_{j,j',j''})$$

We use the very same technique as before to upper bound the sum $\sum_{j,j',j''} \mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D}_{j,j',j''})$ by $q \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D})$ for some adversary $\mathcal{D}$ of similar complexity.

Therefore, we have

$$\mathsf{Adv}_{\mathsf{INDstrong}}(\mathcal{A}) \leq \frac{\tau^2}{2}2^{-L} + \tau \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}) + q \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_1}(\mathcal{C}) + \\ q \cdot \mathsf{Adv}_{\mathsf{INDPRNG}_2}(\mathcal{D}) + \sum_{j=1}^{\tau}\sum_{j'=1}^{\tau_j}\sum_{j''=1}^{\tau_{j,j'}} \mathsf{Adv}_{\mathsf{INDKSsprp}}(\mathcal{A}_{j,j',j''})$$

We let $\mathcal{A}'$ be the adversary $\mathcal{A}_{j,j',j''}$ making $\mathsf{Adv}_{\Gamma^8}(\mathcal{A}_{j,j',j''})/q_{j,j',j''}$ maximal among all those for which $q_{j,j',j''} > 0$.[15] We let q' be the average number of queries of $\mathcal{A}'$.

For any j, j', j'', we have

$$\mathsf{Adv}_{\Gamma^8}(\mathcal{A}_{j,j',j''}) \leq q_{j,j',j''} \cdot \frac{\mathsf{Adv}_{\Gamma^8}(\mathcal{A}')}{q'}$$

Since $\sum_{j,j',j''} q_{j,j',j''} = q$, we obtain

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

B Best Known **INDKSsprp** Attacks on **CEnc**

B.1 Linear Collapse

We have $a!$ possible Sboxes over $\mathbf{Z}_a$, but $a\varphi(a)$ are affine over $\mathbf{Z}_a$, where φ is the Euler totient function. Hence, a random Sbox is affine with probability $p_a = \frac{\varphi(a)}{(a-1)!}$. An adversary who pushes his luck expecting that the entire encryption is affine will thus succeed with probability at least $p_a^{\min(m,n)}$, the probability that all Sboxes, either from the pool, or the used ones, are all affine. To obtain an s-bit security, we thus need $\min(m,n) > \frac{s}{\log_2(1/p_a)}$. That is

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

In particular, we must have $a \geq 4$ (for $a = 2$ or $a = 3$, all functions are affine).

Note that we consider linearity in the sense of $\mathbf{Z}_a$ because our construction uses addition in $\mathbf{Z}_a$. For $a = 4$, Sboxes can be considered as linear in the sense of $\mathbf{Z}_2^2$ but mixing them with the modulo-4 addition makes the cipher non-linear.

[15] We note that when $q_{j,j',j''} = 0$, then $\mathcal{A}_{j,j',j''} = 0$ and we can just ignore those (j, j', j'').

B.2 Known Sbox Pool Dictionary Attack

Let u be a parameter. The adversary can prepare a dictionary of all SEQ with indices in $\{0,\ldots,u-1\}$ with the key set to the corresponding encryption of a few constant messages. With probability $\left(\frac{u}{m}\right)^n$, the target SEQ is in this dictionary and the adversary can win. The dictionary has length u^n. Hence, the dictionary attack has complexity $u^n + \left(\frac{m}{u}\right)^n$, which is optimal for $u \sim \sqrt{m}$. The complexity becomes $m^{\frac{n}{2}}$. Hence, for an s-bit security we need

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

B.3 Generic Distinguishers

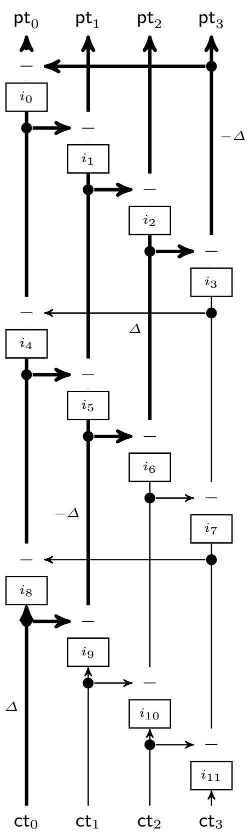
Chosen Ciphertext Distinguisher ($w = 0$ and $w' = 1$ case). We construct a good chosen ciphertext distinguisher in the $w = 0$ case when using up to $n = \ell(\ell-1)$ layers: let ct and ct' be two different ciphertexts such that $\mathsf{ct}' - \mathsf{ct}$ is of form $(?, 0, \ldots, 0)$ (which we write $(?0^{\ell-1})$). I.e., only ct_0 differs. Let $\mathsf{pt} = \mathsf{Dec}[S_0,\ldots,S_{n-1}](\mathsf{ct})$ and $\mathsf{pt}' = \mathsf{Dec}[S_0,\ldots,S_{n-1}](\mathsf{ct}')$. We easily show that $\mathsf{pt}_{\ell-1} - \mathsf{pt}'_{\ell-1} = (-1)^{\ell-1}(\mathsf{ct}_0 - \mathsf{ct}'_0)$. (See Fig. 9 for an example with $\ell = 4$. Paths with differing values are bold.) Hence, we have a distinguisher with advantage $1 - \frac{a^{\ell-1}-1}{a^\ell - 1}$ (which is approximately $1 - \frac{1}{a}$) using 2 chosen ciphertexts.

Chosen ciphertext distinguisher for $n = \ell(\ell-1)$:

```
1: pick $\mathsf{ct}$ and $\mathsf{ct}'$ be two different ciphertexts such that $\mathsf{ct}_i = \mathsf{ct}'_i$ for
   $i = 1,\ldots,\ell-1$
2: query $\mathsf{pt} \leftarrow \mathsf{OD}(\mathsf{ct})$
3: query $\mathsf{pt}' \leftarrow \mathsf{OD}(\mathsf{ct}')$
4: return $1_{\mathsf{pt}_{\ell-1} - \mathsf{pt}'_{\ell-1} = (-1)^{\ell-1}(\mathsf{ct}_0 - \mathsf{ct}'_0)}$
```

When OD is the decryption oracle, the distinguisher always returns 1. When OD is a random permutation, the probability to return 1 is the probability that $\mathsf{pt}_0 = \mathsf{pt}'_0$ for two random different tuples pt and pt', hence $\frac{a^\ell(a^{\ell-1}-1)}{a^\ell(a^\ell-1)}$. Clearly, we need $n > \ell(\ell-1)$ for security.

We extend the previous attack with $n - \ell(\ell-1)$ more layers at the beginning. We can guess the Sboxes in those extra layers with complexity $m^{n-\ell(\ell-1)}$. To filter bad guesses, we need to apply the above distinguisher on many pairs of plaintexts. With a ciphertexts with same $\mathsf{ct}_1,\ldots,\mathsf{ct}_{\ell-1}$, we can form $\frac{a(a-1)}{2}$ pairs. Hence, with N chosen ciphertexts we can form $N\frac{a-1}{2}$ pairs. Since a wrong set of Sboxes passes with probability $a^{-N\frac{a-1}{2}}$ and we have $m^{n-\ell(\ell-1)}$ many, we need $N = 2\frac{n-\ell(\ell-1)}{a-1} \times \frac{\log m}{\log a}$ plaintexts



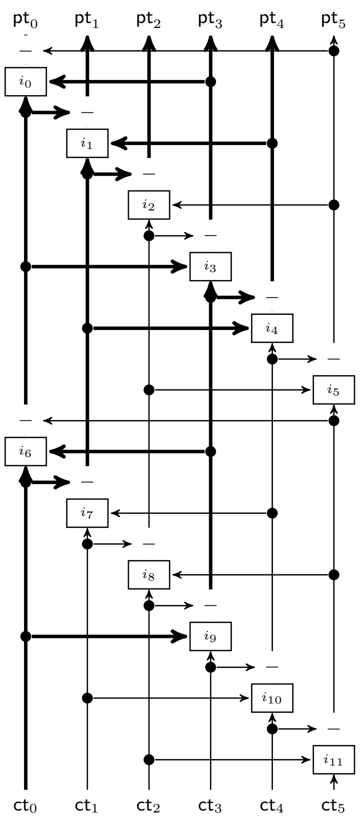


Fig. 9. Chosen Ciphertext Attack with $w = 0$, $w' = 1$, $\ell = 4$, $n = \ell(\ell - 1)$ (left) and $w = 3$, $w' = 1$, $\ell = 6$, $n = (w - 1)\ell$ (right)

to filter the right set of Sboxes. We can neglect the number of plaintexts in the complexity of $m^{n-\ell(\ell-1)}$. Hence, for s-bit security, we need $n > \ell(\ell-1) + \frac{s}{\log_2 m}$.

We could rather try to guess parts of the tables of the extra Sboxes in a way similar to the Durak-Vaudenay optimized exhaustive search [19]. Essentially, we need partial tables for $\sqrt{a}$ points for each Sbox. With $\sqrt{a}$, we start observing contradictions and we can filter some bad guesses. By continuing with more ciphertexts, we recover the extra Sboxes. We estimate this attack requires a ciphertexts and a time complexity of $a^{(n-\ell(\ell-1))\sqrt{a}}$. We essentially replaced m by $a^{\sqrt{a}}$. Hence, for s-bit security, we need

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

This $n = \Omega(\ell^2)$ is quite demanding. This is why we introduced a non-zero w to lower the requirements on n. With our parameters, the $w = 0$ case only appears for $\ell = 2$. So this bound is not applicable for $\ell > 2$.

Framework for a General Differential Distinguisher. We generalize the previous distinguisher by following the notion of *truncated differentials* by Knudsen [25]. Given a pair of values in $\mathbf{Z}_a$, we represent their difference with a ? when it is non-zero and with a 0 otherwise. If we have a pair of inputs to an Sbox, the difference representation of the input and of the output are the same: a 0-input difference always gives a 0-output difference and a ?-input difference always gives a ?-output difference. If we have a pair of inputs to an addition (or a subtraction), we can easily see that $0 + 0 = 0$, $0+? =?$, and $? + 0 =?$. However, $?+?$ gives ? most of the time, but it can give 0 with probability $(a-1)^{-1}$.

Given a plaintext pt, we define a sequence x by $\mathsf{pt} = (x_0, x_1, \ldots, x_{\ell-1})$ and

$$x_i = S_i(S_i(x_{i-\ell} + x_{i-w'}) - x_{i-\ell+w})$$

for $i \geq \ell$ so that $\mathsf{ct} = (x_n, x_{n+1}, \ldots, x_{n+\ell-1})$. For two plaintexts defining two sequences, we represent their difference $\Delta = (\Delta_0, \ldots, \Delta_{n+\ell-1})$ in $\{0, ?\}^{n+\ell}$ component-wise. This is a *difference characteristic.* For each i, Δ_i is computed from $\Delta_{i-\ell}$, $\Delta_{i-w'}$, and $\Delta_{i-\ell+w}$.

- If $\Delta_{i-\ell} = \Delta_{i-w'} = \Delta_{i-\ell+w} = 0$, then $\Delta_i = 0$.
- If a single value among $\Delta_{i-\ell}$, $\Delta_{i-w'}$, and $\Delta_{i-\ell+w}$ is non-zero, then $\Delta_i =?$.
- Otherwise, in most of cases we have $\Delta_i =?$ but we call $\Delta_i = 0$ an *event.* More precisely, if exactly two values among $\Delta_{i-\ell}$, $\Delta_{i-w'}$, and

$\Delta_{i-\ell+w}$ are non-zero, then $\Delta_i = 0$ with probability $(a-1)^{-1}$. However, if $\Delta_{i-\ell} = \Delta_{i-w'} = \Delta_{i-\ell+w} =?$, then $\Delta_i = 0$ with probability $(a-1)^{-1}(1-(a-1)^{-1})$.

We define $\mathsf{event}(\Delta)$ the number of events in Δ.

To visualize what is happening in Δ, we split Δ into blocks of ℓ values. Each block can be represented by a table of $\left\lceil \frac{\ell}{w'} \right\rceil$ rows with w' columns, in which the last row may be incomplete. When w' divides ℓ, the last row is complete. In Fig. 10 we depict a case with w' dividing ℓ for simplicity. Hence, Δ_i is at the same position as $\Delta_{i-\ell}$ in the previous block, $\Delta_{i-w'}$ is the value in the previous row and same column as Δ_i (if in the same block), but $\Delta_{i-\ell+w}$ is "shifted" compared to Δ_i.

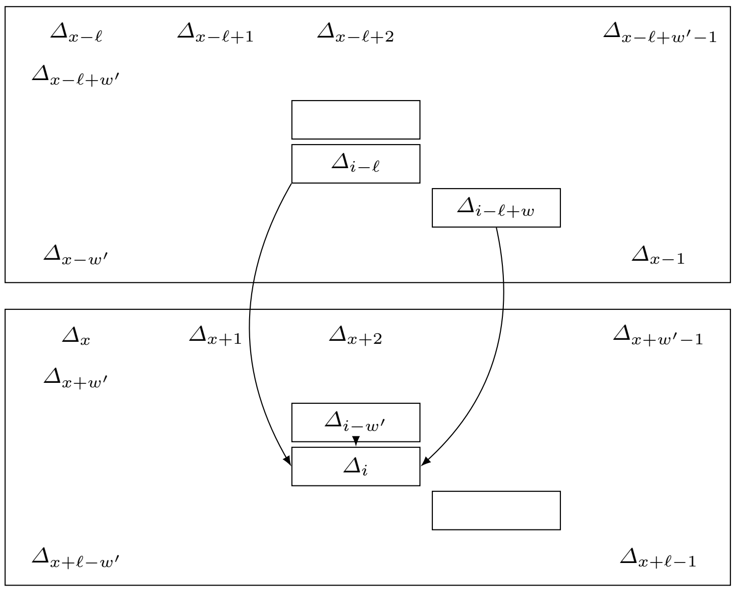


Fig. 10. Differential Pattern (w' dividing ℓ)

As events occur with a probability bounded by $(a-1)^{-1}$, the probability that Δ holds given that $(\Delta_0, \ldots, \Delta_{\ell-1})$ holds is bounded by $(a-1)^{-\mathsf{event}(\Delta)}$.

When we are interested in the decryption computation, the above formalism is similar but the role of w and w' is exchanged. Also, the order of columns should be reversed.

We define a function hw which returns the number of ? in the input of hw.

We formalize a differential chosen plaintext attack as follows. (Chosen ciphertext attacks are similar.) We assume a differential characteristic $\varDelta$ and we write $\varDelta_{\mathsf{pt}} = (\varDelta_0, \ldots, \varDelta_{\ell-1})$ and $\varDelta_{\mathsf{ct}} = (\varDelta_n, \ldots, \varDelta_{n+\ell-1})$.

Differential chosen plaintext distinguisher:

```
1: pick $\mathsf{pt}$ and $\mathsf{pt}'$ be two different ciphertexts such that $\mathsf{pt}' - \mathsf{pt} \in \varDelta_{\mathsf{pt}}$
2: query $\mathsf{ct} \leftarrow \mathsf{OE}(\mathsf{pt})$
3: query $\mathsf{ct}' \leftarrow \mathsf{OE}(\mathsf{pt}')$
4: **return** $1_{\mathsf{ct}'-\mathsf{ct}\in\varDelta_{\mathsf{ct}}}$
```

The attack uses N chosen plaintexts by taking $Na^{-\mathsf{hw}(\varDelta_{\mathsf{pt}})}$ packets of $a^{\mathsf{hw}(\varDelta_{\mathsf{pt}})}$ plaintexts such that each pair in the same packet belongs to $\varDelta_{\mathsf{pt}}$. Hence, the number of pairs with $\mathsf{pt}' - \mathsf{pt} \in \varDelta_{\mathsf{pt}}$ is roughly $M = \frac{N}{2}a^{\mathsf{hw}(\varDelta_{\mathsf{pt}})}$. We count the number of pairs with $\mathsf{pt}' - \mathsf{pt} \in \varDelta_{\mathsf{pt}}$ and $\mathsf{ct}' - \mathsf{ct} \in \varDelta_{\mathsf{ct}}$. We call the counted pairs which did not follow the characteristic as *bad pairs*. Counted pairs which follow the characteristic are *good pairs*. With the ideal cipher, a pair is counted with probability $p_{\mathsf{noise}} = a^{\mathsf{hw}(\varDelta_{\mathsf{ct}})-\ell}$. The number of counted pairs is the sum of M independent identically distributed Bernoulli random variables. This number has expected value Mp_{noise} and standard deviation $\sqrt{Mp_{\mathsf{noise}}}$ with the ideal cipher. With the real cipher, a pair is good with probability $p_{\mathsf{good}} = (a-1)^{-\mathsf{event}(\varDelta)}$. Hence, a pair is counted with probability $p_{\mathsf{good}}(1 - p_{\mathsf{noise}}) + p_{\mathsf{noise}}$. We approximate the standard deviation of the number of counted pairs to be the same for the real cipher and the ideal cipher. We can distinguish the real cipher from the ideal one with constant advantage when the difference of expected values match the standard deviation, hence when $Mp_{\mathsf{good}} \approx \sqrt{Mp_{\mathsf{noise}}}$. We obtain $N = 2a^{-\mathsf{hw}(\varDelta_{\mathsf{pt}})+\mathsf{hw}(\varDelta_{\mathsf{ct}})-\ell}(a-1)^{2\mathsf{event}(\varDelta)}$. We add t more layers and we do an exhaustive search, either on the Sbox indices of those layers (there are m^t possibilities), or on the permutations of $\mathbf{Z}_a$ (there are $(a!)^t$ possibilities). Hence, $\varDelta$ is now on $n-t$ layers. To have security, we want $N \min(m, a!)^t > 2^s$. Hence, we need

$$\begin{aligned}&2 \cdot \mathsf{event}(\varDelta)\log(a-1) + t\log\min(m, a!) > \\ &(\mathsf{hw}(\varDelta_{\mathsf{pt}}) - \mathsf{hw}(\varDelta_{\mathsf{ct}}) + \ell)\log a + (s-1)\log 2\end{aligned}$$

where $\varDelta$ is a characteristic over $n-t$ layers.

Chosen Plaintext Attack ($w > 1$, w and w' coprime). We apply the previous formalism on chosen plaintext. Our list $\varDelta$ is represented with w' columns. We assume that w' divides ℓ for simplicity (other cases generalize what follows). We start with the initial block $\varDelta_{\mathsf{pt}}$ of form $0^{\ell-1}?$. Hence, the first rows are all-zero but the last row has form $(0, \ldots, 0, ?)$. Then, we easily see that the list $\varDelta$ is a list of rows which repeat $\frac{\ell-w'}{w'}$ times. For $w =$

$w'+1$, the rows are of form $0^{w-i}?^i$ with incrementing i. (This generalizes whenever w and w' are coprime.) We consider $n = (\ell - w')(w'+1) + t$ layers. The hw value of the block before the last t layers is $\ell - 1$. We have $\mathsf{hw}(\Delta_{\mathsf{pt}}) = 1$, $\mathsf{hw}(\Delta_{\mathsf{ct}}) = \ell - 1$, and $\mathsf{event}(\Delta) = 0$. The inequality in our general framework with $n = (\ell - w')(w'+1) + t$ gives

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

With chosen ciphertext attacks, the same bound with w instead of w' applies. (See Fig. 9 for a chosen ciphertext attack with $w' = 1$.)

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

Chosen Plaintext Attack ($w + w'$ dividing ℓ). We consider the case of $w+w'$ dividing ℓ, which can happen. The following attack can be adapted to other parameters too. Given two plaintexts pt and pt' such that $\mathsf{pt}' - \mathsf{pt} \in (0^{w-1}?(0^{w+w'-1}?)^{\frac{\ell}{w+w'}-1}0^{w'})$, we can see that the $w+w'-1$ first layers only shift the difference pattern and that the last one does not create any new ? difference with probability $\frac{1}{a-1}$. Hence, this fits to our general framework with one event every $w+w'$ layers. We have $\mathsf{hw}(\Delta_{\mathsf{pt}}) = \mathsf{hw}(\Delta_{\mathsf{ct}}) = \frac{\ell}{w+w'}$ and $\mathsf{event}(\Delta) = \frac{n-t}{w+w'}$. Hence, we need

$$n > t + (w+w')\frac{s - 1 + \ell\log_2 a - t\log_2 \min(m, a!)}{2\log_2(a-1)}$$

for all $t \geq 0$, with $n > t$. Depending on the sign of $2\log(a-1) - (w + w')\log\min(m, a!)$, the largest lower bound is reached for $t = 0$ or t making the second term vanish. Hence,

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

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

Chosen Plaintext Attack (w' dividing $\ell + 1$). We consider the case of w' dividing $\ell+1$, which can happen. The following attack can be adapted to other parameters too. Given two plaintexts pt and pt' such that $\mathsf{pt}' - \mathsf{pt} \in (0^{w-1}?0^{\ell-w})$, i.e. the second row of the differential block is $?0^{w'-1}$ and all others are null, we can see that the $2w'-1$ first layers only shift the difference pattern then starts copying a ? in every w' layers. This fits to our general framework with no event. We have $\mathsf{hw}(\Delta_{\mathsf{pt}}) = 1$, $\mathsf{hw}(\Delta_{\mathsf{ct}}) = \frac{n-t-w'}{w'}$, and $\mathsf{event}(\Delta) = 0$. Hence, we need

$$n > t + w' \frac{s - 1 + (\ell + 2)\log_2 a - t \log_2 \min(m, a!)}{\log_2 a}$$

for all $t \geq 0$ with $n > t$. Depending on the sign of $\log(a) - w' \log \min(m, a!)$, the largest lower bound is reached for $t = 0$ or t making the second term vanish. Hence,

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

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

B.4 Differential and Linear Cryptanalysis

To study differential cryptanalysis, we consider differentials of a particular form, corresponding to our framework. Each $\mathbf{Z}_a$ input difference can be either 0 or something random but non-zero, which we denote by a question mark ?. We use the rules $0 + 0 = 0$, $0+? =?$, and $?+?$ is 0 with probability $\frac{1}{a-1}$ and ? with probability $1 - \frac{1}{a-1}$. Rules are similar for subtraction. Furthermore, differences of 0 or ? traverse Sboxes. That is, a zero input difference leads to a zero output difference, and a non-zero input difference leads to a non-zero output difference for sure. Like this, we can consider all $2^\ell - 1$ possible patterns of differences in $\mathbf{Z}_a^\ell$ with 0 and ? and compute the probability that one pattern in $\{0, ?\}^\ell - \{0^\ell\}$ gives one other pattern in $\{0, ?\}^\ell - \{0^\ell\}$ after one layer. We hence populate a $(2^\ell - 1) \times (2^\ell - 1)$ stochastic matrix T. By raising this matrix to the power n, we obtain the probability that an input difference pattern gives an output difference pattern after n layers. We can then form the matrix T^* of the ideal cipher in which the entry corresponding to input difference Δ_{pt} and output difference Δ_{ct} has value $\frac{(a-1)^x}{a^\ell - 1}$, where x is the number of ? in Δ_{ct}, and compute $T^n - T^*$.[16] For any input difference pattern Δ_{pt} and any set of output difference patterns Δ_{ct}, we can then consider the following attack:

```
1: pick two random plaintext $X$ and $X'$ such that $X' - X \in \Delta_{\mathsf{pt}}$
2: query $Y = \mathsf{OE}(X)$ and $Y' = \mathsf{OE}(X')$
3: **return** $1_{Y' - Y \in \Delta_{\mathsf{ct}}}$
```

[16] We easily show that $T^*T^* = T^*$. We also have $T^*T = T$ but $TT^* \neq T$.

where $X' - X \in \mathit{\Delta}_{\mathsf{pt}}$ means that for every i, the component $X'_i - X_i$ belongs to the ith component of $\mathit{\Delta}_{\mathsf{pt}}$, belonging to 0 means being equal to 0, and belonging to ? means being non-zero. The advantage is

$$\mathsf{Adv} = \frac{1}{2} \sum_{\mathit{\Delta} \in \mathit{\Delta}_{\mathsf{ct}}} |(T^n - T^*)_{\mathit{\Delta}, \mathit{\Delta}_{\mathsf{pt}}}|$$

Hence, the best advantage is half of the largest L_1 norm of a column of $T^n - T^*$. We do the same with OD instead of OE.

This type of attack covers **truncated differentials**, **impossible differentials**, and regular **differential cryptanalysis**. We stress that we considered **differentials** and not only **differential characteristics**.

We want that the advantage of any such attack to be at most 2^{-s}. We computed below the lowest value r such that $n = r\ell$ gives no such distinguisher with advantage better than 2^{-s} for $s = 128$. On Table 3 we report this minimal number of rounds. It includes the analysis for both matrices with OE and OD and also the linear cryptanalysis to be made below. As an example, for $a = 10$ and $\ell = 5$, this technique found that $r = 23$ rounds is insufficient. We checked that this comes from a distinguisher of advantage $2^{-126.97}$ using OD (chosen ciphertext) with $\mathit{\Delta}_{\mathsf{ct}} = [0000?]$ and

$$\mathit{\Delta}_{\mathsf{pt}} = \left\{ \begin{array}{l} [000??]], [00???], [0?000], [0?00?], [0?0??], [0??0?], \\ [0????], [?000?], [?00?0], [?00??], [?0???], [??00?], \\ [??0?0], [??0??], [???0?], [?????] \end{array} \right\}$$

In addition to this, there are ad hoc ways to get more layers "for free" by choosing some nonzero inputs of particular values instead of being random (e.g. some difference δ to be added to a difference $-\delta$). For that, we now consider any $\mathbf{Z}_a$ difference in $\mathbf{Z}_a \cup \{?\}$. Addition and subtractions are the ones from $\mathbf{Z}_a$ with the additional absorbing term ?. However, traversing Sboxes no longer come for free. We assume that a nonzero input difference of $x \neq ?$ to an Sbox gives the output difference ?. Unfortunately, we cannot form $((a+1)^\ell - 1) \times ((a+1)^\ell - 1)$ matrices. However, we can see that it is only in the first layer that the adversary can arrange the addition of a non-zero non-random difference x to another non-zero non-random difference y. After the first layer, one term on the addition will come out of an Sbox from the previous layer, hence will be either 0 or ?. The w-branch further affects the output of an Sbox. Hence, the arithmetic in $\mathbf{Z}_a \cup \{?\}$ can only give one free layer. We consider the existence of this free layer to be negligible in our analysis.

In the chosen ciphertext mode, additions and subtractions are only on the output of an Sbox. Hence, no layer comes for free.

Table 3. Minimal Number of Secure Rounds for Differential and Linear Cryptanalysis

ℓ:	2	3	4	5	6
$a = 4$	81	50	52	46	48
$a = 5$	65	39	40	37	37
$a = 6$	56	34	34	32	31
$a = 7$	50	30	30	29	28
$a = 8$	47	28	28	27	25
$a = 9$	43	26	26	25	24
$a = 10$	41	24	24	24	22
$a = 11$	39	23	23	22	21
$a = 12$	38	22	22	22	20
$a = 13$	37	22	21	21	20
$a = 14$	35	21	21	20	19
$a = 15$	34	20	20	20	19
$a = 16$	34	20	20	19	18
$a = 100$	20	12	12	12	11
$a = 128$	19	11	11	11	10
$a = 256$	17	10	10	10	9
$a = 1000$	14	8	8	9	8
$a = 1024$	14	8	8	9	8
$a = 10000$	10	6	7	7	6
$a = 65536$	9	6	6	6	5

Linear Cryptanalysis. We made a linear cryptanalysis in a way similar to Baignères et al. [5]. Given a function f from a group G to itself, we define

$$\begin{aligned}\mathsf{LP}^f(\alpha,\beta) &= |E_X[\overline{\alpha(X)}\beta(f(X))]|^2\\ &= E_{X,X'}\left[\overline{\alpha(X'-X)}\beta(f(X')-f(X))\right]\end{aligned}$$

for characters $\alpha, \beta \in \hat{G}$. For $G = \mathbf{Z}_a^\ell$, we can associate $\alpha = (\alpha_0, \ldots, \alpha_{\ell-1})$, $\beta = (\beta_0, \ldots, \beta_{\ell-1})$, and

$$\mathsf{LP}^f(\alpha,\beta) = \left|E_X\left[e^{\frac{2i\pi}{a}(-\alpha_0X_0-\cdots-\alpha_{\ell-1}X_{\ell-1}+\beta_0f_0(X)+\cdots+\beta_{\ell-1}f_{\ell-1}(X))}\right]\right|^2$$

We write $\theta = e^{\frac{2i\pi}{a}}$ and we have

$$\mathsf{LP}^f(\alpha,\beta) = \left|E_X\left[\theta^{-\alpha_0X_0-\cdots-\alpha_{\ell-1}X_{\ell-1}+\beta_0f_0(X)+\cdots+\beta_{\ell-1}f_{\ell-1}(X)}\right]\right|^2$$

It is well known that the complexity of linear cryptanalysis against a function f has an order of magnitude which is the inverse of $\mathsf{LP}^f(\alpha,\beta)$. Hence, we want that for all non-zero α and β, $\mathsf{LP}^f(\alpha,\beta) \le 2^{-s}$.

It was proven [5] that

$$E_S[\mathsf{LP}^{\mathsf{CEnc}[S_1,\ldots,S_n]}(\alpha_0,\alpha_n)] = \sum_{\alpha_1,\ldots,\alpha_{n-1}} \prod_{i=1}^{n} E_{S_i}[\mathsf{LP}^{\mathsf{CEnc}[S_i]}(\alpha_{i-1},\alpha_i)]$$

In a single layer $\mathsf{LP}^{\mathsf{CEnc}[S_i]}(\alpha_{i-1},\alpha_i)$ is either 0, 1, or some product of form $\mathsf{LP}^{S_i}(u,v)\mathsf{LP}^{S_i}(v,w)$ for u,v,w non-zero. We keep in the sum only the non-zero terms. This defines some rules to obtain a list of possible α_{i-1} from α_i. If we average over all terms, we can group the α_i's following their pattern of 0 and ?, like in differential cryptanalysis.

We prove in Appendix D the following result.

Lemma 8. *For any permutation S of $\mathbf{Z}_a$ and uniform random $u,v,w \in \mathbf{Z}_a - \{0\}$, we have*

$$E_{u,v,w\neq 0}[\mathsf{LP}^{S_i}(u,v)\mathsf{LP}^{S_i}(v,w)] = \frac{1}{(a-1)^2}$$

We use this result to say that an output mask $(\beta_0,\ldots,\beta_{\ell-2},0)$ of a layer gives an input mask $(0,\beta_1,\ldots,\beta_{\ell-2})$ with "probability" 1 and an output mask of $(\beta_0,\ldots,\beta_{\ell-2},?)$ gives $(a-1)^2$ input masks

$$(?,\beta_1,\ldots,\beta_{w-2},\beta_{w-1}+?,\beta_w,\ldots,\beta_{\ell-3},\beta_{\ell-2}+?)$$

with "probability" $(a-1)^{-2}$. Depending on β_{w-1} and $\beta_{\ell-2}$, this creates up to 4 masks, each of different weight. We can then form a stochastic matrix like for differential cryptanalysis and perform the same treatment.

We checked that the values in Table 3 are well approximated by the empirical formula $r \approx \frac{s}{\sqrt{\ell}\ln(a-1)}$. We deduce the following criterion:

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

B.5 Fixed Point Attack

If all selected Sboxes have 0 as a fixed point, then the full encryption has $(0,\ldots,0)$ as a fixed point. We can thus have a distinguisher with advantage equal to the probability that all selected Sboxes have 0 as a fixed point. In a known-S setting, we know the number u of Sboxes with 0 as a fixed point. Hence, with probability $\left(\frac{u}{m}\right)^n$, all Sboxes in CEnc are in this set. Given that u should be close to $\frac{m}{a}$, we have a distinguisher of advantage a^{-n}. Hence, we need

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

Using several tweaks, this advantage amplifies linearly.

B.6 Side-Channel Attacks

For information, we put here a few side-channel attacks. We assume implementations to be immune to these attacks.

Leakage in the S generation. Our Setup_1 algorithm may leak because of the way we generate Sboxes. However, it is solely based on coins coming from AES computations and our model tolerates known S. Hence, the worst case is that the coins fully leak which is still safe. The robustness of AES protects against this type of attack. More precisely, we do not need to implement Setup_1 in constant-time.

Fault analysis. If an adversary can make the encryptor make a mistake when accessing one Sbox, he can see in repeating the same encryption if this Sbox occurs in the very last layers. When it happens, he deduces the input difference of the next Sbox and can see the two outputs. For instance, if the ciphertext $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\ell-1})$ has an incorrect $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\ell-1})$ and we can see that $\mathsf{ct}_i = \mathsf{ct}'_i$ for $i = 0, \ldots, j-1, j+2, \ldots, \ell-1$, we deduce that the last Sbox S computing ct_{j+1} has an input difference $\mathsf{ct}'_j - \mathsf{ct}_j$ giving ct_{j+1} and ct'_{j+1}. We deduce

$$S^{-1}(\mathsf{ct}'_{j+1}) - S^{-1}(\mathsf{ct}_{j+1}) = \mathsf{ct}'_j - \mathsf{ct}_j$$

By collecting a bit more than a relations like this with same j, we deduce a permutation π such that $S(x) = \pi(x+c)$ for a constant c. We reconstruct all Sboxes up to some constant like this. By inverting the last layers with recovered Sboxes, the attack can iterate to recover all Sboxes in the reverse order as they are used.

Leakage. Similarly, assuming some information about the input of the Sbox in the last layers leak (for instance, the Hamming weight, or the existence of a carry in the previous addition modulo a), we can reconstruct the Sboxes similarly.

The addition modulo a should be implemented with care, as it is most likely to leak information.

B.7 Collision Attacks

An adversary can select arbitrarily a set of plaintexts with entropy larger than the entropy of K_{SEQ}. Then, for many tweaks, a chosen plaintext attack on this set will yield two tweaks which collide on K_{SEQ}. Once they are spotted, the adversary can observe that new chosen plaintexts lead to the same ciphertext with both tweaks. This is a property of our cipher which is not satisfied by a random permutation family. Hence, this is a distinguisher with large advantage. The number of queries has an order of magnitude $\sqrt{2^{L_1}}$, thanks to the birthday paradox. Hence, requiring $L_1 \geq 2s$ defeats this attack.

C Parameters for Quantum Security

For quantum $s = 128$ security, the adversary can run quantum algorithms. We replace s by $2s$ in all equations (to resist to Grover-like algorithms) except $L_1 = L_2 = 2s$ which is replaced by $L_1 = L_2 = 3s$ as it is used for collision resistance. We recommend $m = 256$, $a \geq 4$, $\ell \geq 2$, n and w selected as before by replacing $s = 128$ by $s = 256$. As for key lengths, we use $L = 2s$ and $L_1 = L_2 = 3s$. We do not need a 512-bit key for $L_1 = L_2$ as the limitation is a collision attack for which $3s$ bits are enough. Hence, K_{SEQ} and K_S are a 256-bit AES key together with a 128-bit IV. We obtain the number of rounds which is given in Table 4.

Since AES-CMAC only handles 128-bit keys, it should be replaced. PRNG_1 and PRNG_2 need no change, except for the size of their AES keys.

D Technical Lemma about Linear Cryptanalysis

Let S be a permutation of $\mathbf{Z}_a$. We compute below the average of this product over the non-zero u, v, w.

$$\begin{aligned} &E_{u,v,w\neq 0}[\mathsf{LP}^S(u,v)\mathsf{LP}^S(v,w)] \\ =&E_{\substack{u,v,w\neq 0\\ X,X',Y,Y'}}\left[\theta^{u(X'-X)+v(S_i(X)-S_i(X')+Y'-Y)+w(S_i(Y)-S_i(Y'))}\right] \end{aligned}$$

We split the expected value with the event $X = X'$ and obtain $\frac{1}{a}E_{X=X'} + \left(1-\frac{1}{a}\right)E_{X\neq X'}$, with

$$\begin{aligned} E_{X=X'} &= E_{\substack{v,w\neq 0\\ Y,Y'}}\left[\theta^{v(Y'-Y)+w(S_i(Y)-S_i(Y'))}\right] \\ &= \frac{1}{a} + \left(1-\frac{1}{a}\right)E_{\substack{v,w\neq 0\\ Y\neq Y'}}\left[\theta^{v(Y'-Y)+w(S_i(Y)-S_i(Y'))}\right] \end{aligned}$$

For $y \neq 0$, we have

$$E_{v\neq 0}\left[\theta^{vy}\right] = -\frac{1}{a-1}$$

Table 4. Number of Rounds for Quantum 128-bit Security

ℓ:	2	3	4	5	6	7	8	9	10	12	16	32	50	64	100
$a = 4$	330	270	234	209	191	177	165	156	148	135	117	83	74	73	73
$a = 5$	262	214	185	166	151	140	131	124	117	107	93	68	65	64	66
$a = 6$	225	184	160	143	130	121	113	107	101	92	80	62	60	60	63
$a = 7$	203	165	143	128	117	109	102	96	91	83	72	58	57	57	60
$a = 8$	187	152	132	118	108	100	94	88	84	76	66	55	55	55	59
$a = 9$	175	143	124	111	101	94	88	83	78	72	62	53	53	54	58
$a = 10$	165	135	117	105	96	89	83	78	74	68	59	52	52	53	57
$a = 11$	158	129	112	100	91	85	79	75	71	65	56	50	51	52	56
$a = 12$	151	124	107	96	88	81	76	72	68	62	54	49	50	51	55
$a = 13$	146	119	104	93	85	78	73	69	66	60	52	48	49	50	55
$a = 14$	142	116	100	90	82	76	71	67	64	58	51	48	48	50	54
$a = 15$	138	113	98	87	80	74	69	65	62	57	50	47	48	49	54
$a = 16$	134	110	95	85	78	72	67	64	60	55	49	46	47	49	54
$a = 100$	79	65	56	50	46	43	40	38	38	37	36	37	40	42	48
$a = 128$	75	62	53	48	44	40	38	37	36	36	35	36	39	42	48
$a = 256$	66	54	47	42	38	35	34	34	33	33	33	34	38	41	47
$a = 1000$	64	43	38	34	31	31	30	30	29	29	29	32	36	39	46
$a = 1024$	64	43	37	34	31	30	30	30	29	29	29	32	36	39	46
$a = 10000$	64	43	32	27	26	26	25	25	25	25	26	30	34	37	44
$a = 65536$	64	43	32	26	23	23	23	23	23	24	25	29	33	37	44

Hence,

$$E_{X=X'} = \frac{1}{a} + \left(1 - \frac{1}{a}\right)\left(-\frac{1}{a-1}\right)^2 = \frac{1}{a-1}$$

Similarly,

$$\begin{aligned}
&E_{X\neq X'} \\
&= -\frac{1}{a-1} E_{\substack{v,w\neq 0 \\ X\neq X',Y,Y'}} \left[\theta^{v(S_i(X)-S_i(X')+Y'-Y)+w(S_i(Y)-S_i(Y'))}\right] \\
&= -\frac{1}{a-1}\left(\frac{1}{a}E_{X\neq X',Y=Y'} + \left(1-\frac{1}{a}\right)E_{X\neq X',Y\neq Y'}\right)
\end{aligned}$$

with

$$E_{X\neq X',Y=Y'} = E_{\substack{v,w\neq 0 \\ X\neq X',Y=Y'}}\left[\theta^{v(S_i(X)-S_i(X'))}\right] = -\frac{1}{a-1}$$

and

$$\begin{aligned}
E_{X\neq X',Y\neq Y'} &= E_{\substack{v,w\neq 0 \\ X\neq X',Y\neq Y'}}\left[\theta^{v(S_i(X)-S_i(X')+Y'-Y)+w(S_i(Y)-S_i(Y'))}\right] \\
&= -\frac{1}{a-1}E_{\substack{v\neq 0 \\ X\neq X',Y\neq Y'}}\left[\theta^{v(S_i(X)-S_i(X')+Y'-Y)}\right] \\
&= -\frac{1}{a-1}E_{\substack{v\neq 0 \\ U,V\neq 0}}\left[\theta^{v(U-V)}\right]
\end{aligned}$$

with $U = S_i(x) - S_i(X')$ and $V = Y - Y'$, which we split into $U = V$ and $U \neq V$ to obtain

$$E_{X\neq X',Y\neq Y'} = -\frac{1}{a-1}\left(\frac{1}{a-1} - \left(1 - \frac{1}{a-1}\right)\frac{1}{a-1}\right) = -\frac{1}{(a-1)^3}$$

Finally,

$$E_{u,v,w\neq 0}[\mathsf{LP}^S(u,v)\mathsf{LP}^S(v,w)] = \frac{1}{(a-1)^2}$$