An Inverse-free Single-Keyed Tweakable Enciphering Scheme

Ritam Bhaumik and Mridul Nandi
bhaumik.ritam@gmail.com mridul.nandi@gmail.com

Indian Statistical Institute, Kolkata

Abstract. In CRYPTO 2003, Halevi and Rogaway proposed CMC, a tweakable enciphering scheme (TES) based on a blockcipher. It requires two blockcipher keys and it is not inverse-free (i.e., the decryption algorithm uses the inverse (decryption) of the underlying blockcipher). We present here a new *inverse-free*, *single-keyed* TES. Our construction is a tweakable strong pseudorandom permutation (TSPRP), i.e., it is secure against chosen-plaintext-ciphertext adversaries assuming that *the underlying blockcipher is a pseudorandom permutation* (PRP), i.e., secure against chosen-plaintext adversaries. In comparison, SPRP assumption of the blockcipher is required for the sprp security of CMC. Our scheme can be viewed as a mixture of type-1 and type-3 Feistel cipher and so we call it FMix or mixed-type Feistel cipher.

Keywords: (Tweakable Strong) pseudorandom permutation, Coefficient H Technique, Encipher, CMC, Fiestel Cipher.

1 Introduction

A **tweakable enciphering scheme** (TES) is a *length-preserving encryption scheme* that takes a *tweak* as an additional input. In other words, for each tweak, TES computes a ciphertext preserving length of the plaintext. Preserving length can be very useful in applications such as disk-sector encryption (as addressed by the IEEE SISWG P1619), where a length-preserving encryption preserves the file size after encryption. When a tweakable enciphering scheme is used, the disk sectors can serve as tweaks. Other applications of enciphering schemes could include bandwidth-efficient network protocols and security-retrofitting of old communication protocols.

Examples based on Paradigms. There are four major paradigms of tweakable enciphering schemes. Almost all enciphering schemes fall in one of the following categories.

- **Feistel structure:** 2-block Feistel design was used in early block ciphers like Lucifer[4][22] and DES[23]. Luby and Rackoff gave a security proof of Feistel ciphers[12], and later the design was generalised to obtain inverse-free enciphering of longer messages[17]. *Examples:* Naor-Reingold Hash[16], GFN[10], matrix representations[1].

– **Hash-counter-hash:** Two layers of universal hash with a counter mode of encryption in between. *Examples:* XCB[13], HCTR[25], HCH[2].

– **Hash-encrypt-hash:** Two layers of universal hash with an ECB mode of encryption in between. *Examples:* PEP[3], TET[6], HEH[21].

– **Encrypt-mix-encrypt:** Two encryption layers with a mixing layer in between. *Examples:* EME[8], EME*[5] (with ECB encryption layer), CMC[7] (with CBC encryption layer).

Among all these constructions, the examples from Feistel cipher and Encrypt-mix-encrypt paradigms are based on blockciphers alone (i.e., no field multiplication or other primitive is used). Now we take a closer look at CMC encryption.

CMC. In CRYPTO 2003, Halevi and Rogaway proposed CMC, a tweakable enciphering scheme (TES) based on a blockcipher. It accepts only plaintexts of size a multiple of n, the size of the underlying blockcipher. We call each n-bit segment of the plaintext a *block*. The CMC construction has the following problems:

– For an encryption using e_K, the decryption needs e_K^{-1}. In a **combined hardware implementation**, the footprint size (e.g., the number of gates or slices) goes up;
– The security proof of CMC relied on the stronger assumption SPRP (Strong Pseudo-Random Permutation) on the underlying blockcipher;
– Tweak is processed using an **independent key**, and the proposed single-key variant uses an **extra call** to the blockcipher.

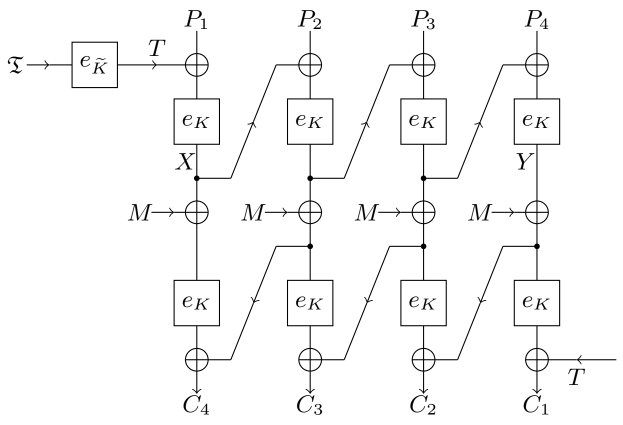

Fig. 1. CMC for four blocks, with tweak $\mathfrak{T}$ and $M = 2(X \oplus Y)$. Here 2 represents a primitive element of a finite field over $\{0,1\}^n$.

Feistel Cipher: An Inverse-free Cipher. To resolve the first issue mentioned above, one can fall back on a Feistel network. For inverse-free constructions, the main approach so far has been to generalise the classical 2-block Feistel network to work for longer messages. Two of the interesting approaches were the type-1 Feistel network and the type-3 Feistel network. In [10], it is shown that to encrypt ℓ block plaintext, type-1 and type-3 need $4\ell - 2$ and $2\ell + 2$ rounds respectively for achieving birthday security, which translates to $4\ell - 2$ and $2\ell^2 - 2$ invocations of the underlying blockcipher. However, their result is meant for providing a security performance trade-off and there is a provision for having beyond-birthday security.

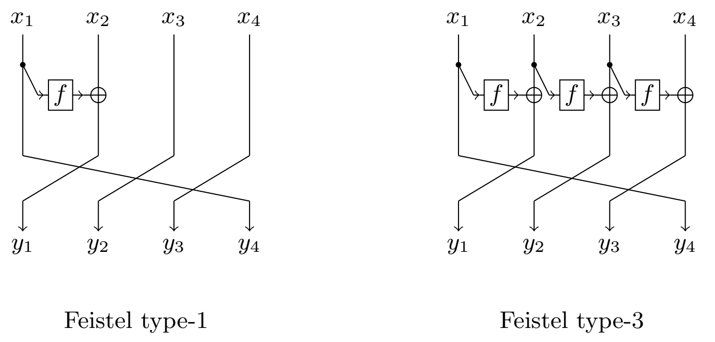

Fig. 2. The round function of two types of generalised Feistel networks for four block inputs. Similar definition can be applied for any number of blocks.

One recent inverse-free construction based on Feistel networks is the AEZ-core, which forms part of the implementation of AEZ[9]. It belongs to the Encrypt-Mix-Encrypt paradigm, where the encryption uses a Feistel structure. It requires five blockcipher calls for every two plaintext blocks, but is highly parallelizable.

1.1 Our Contribution

In this paper, we address all the issues present in CMC in our construction. We use a mixture of type-1 and type-3 for our construction (hence the name FMix) to have an inverse-free construction which minimizes the number of blockcipher calls. FMix applies a simple balanced regular function b. Except for this, it looks exactly like the composition of $\ell + 1$ rounds of type-1 and one round of type-3 Feistel cipher. The features of FMix can be summarized as follows (see Table 1 for a comparison study):

1. FMix is **inverse-free**, i.e., it needs the same f for both encryption and decryption, having low footprint in the combined hardware implementation.

2. Because it is inverse-free, an important improvement is on the **security requirement** of e_K. CMC relies upon an SPRP-secure e_K, while our construction just needs a PRF-secure e_K. This can have significant practical implications in reducing the cost of implementation.
3. The tweak is processed through the same f, removing the requirement of an extra independent blockcipher key.
4. To encrypt a message with ℓ blocks and a tweak (a single block), CMC needs $2\ell+1$ calls to the blockcipher e. Its variant (which eliminates the independent key), however uses $2\ell+2$ calls to e. Our construction requires $2\ell+1$ calls, without needing the independent key.

Table 1. A Comparison of some blockcipher based TES. The description of the columns are as follows: (1) Number of blockcipher calls, (2) Number of keys, (3) How many sequential layers with full parallelization, (4) Security assumption of the underlying blockcipher, (5) Whether it is inverse-free. (CMC' is a "natively tweakable" variant of CMC, as described in [7]).

Schemes	#BC	#Key	#Layers	BC-security	Inverse-free?
CMC	$2\ell+1$	2	$\ell+2$	SPRP	NO
CMC'	$2\ell+2$	2	$\ell+2$	SPRP	NO
EME	$2\ell+3$	1	4	SPRP	NO
GFN-1	$4\ell-2$	$4\ell-2$	$4\ell-2$	PRP	YES
GFN-3	$2\ell^2-2$	$2\ell^2-2$	$2\ell-2$	PRP	YES
AEZ-core	$\sim\frac{5}{2}\ell$	1	5	PRP	YES
FMix (this paper)	$2\ell+1$	1	$\ell+3$	PRP	YES

2 Preliminaries

2.1 Tweakable Encryption Schemes

This paper proposes a new **tweakable encryption scheme**, so we begin by describing what we mean by that. Formally, with a tweakable (deterministic) encryption scheme we associate four finite sets of binary strings: the message space $\mathcal{M}$, the tweak space $\mathcal{T}$, the ciphertext space $\mathcal{C}$, and the key space $\mathcal{K}$. The encryption function $\mathfrak{e} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \longrightarrow \mathcal{C}$ and the corresponding decryption function $\mathfrak{d} : \mathcal{K} \times \mathcal{T} \times \mathcal{C} \longrightarrow \mathcal{M}$ are required to satisfy the following (known as the **correctness requirement**):

$$\forall (K, \mathfrak{T}, P) \in \mathcal{K} \times \mathcal{T} \times \mathcal{M}, \quad \mathfrak{d}(K, \mathfrak{T}, \mathfrak{e}(K, \mathfrak{T}, P)) = P.$$

We also write $\mathfrak{e}(K, \mathfrak{T}, P)$ by $\mathfrak{e}_K(\mathfrak{T}, P)$ and $\mathfrak{d}(K, \mathfrak{T}, C)$ by $\mathfrak{e}_K^{-1}(\mathfrak{T}, C)$. We call a tweakable encryption scheme **tweakable enciphering scheme** (TES) if for all plaintext P, key $K \in \mathcal{K}$ and tweak $\mathfrak{T} \in \mathcal{T}$, $|\mathfrak{e}(K, \mathfrak{T}, P)| = |P|$ (i.e., it preserves length).

Random Function. In the heart of most encryption schemes lies the notion of a **random function**. Given a domain $\mathcal{D}$ and a range $\mathcal{R}$, a random function

$$f : \mathcal{D} \xrightarrow{*} \mathcal{R}$$

is a function chosen uniformly from the class of all functions from $\mathcal{D}$ to $\mathcal{R}$ (denoted $\mathcal{R}^{\mathcal{D}}$). Some elementary calculations show that for distinct $x_1, ..., x_n \in \mathcal{D}$, $f(x_1), ..., f(x_n)$ are independent and uniformly distributed over $\mathcal{R}$. More generally, we define the following:

Definition 1. *Let $\mathscr{C} \subseteq \mathcal{R}^{\mathcal{D}}$ be a class of functions from $\mathcal{D}$ to $\mathcal{R}$. A random $\mathscr{C}$-function*

$$f : \mathcal{D} \xrightarrow{\mathscr{C}} \mathcal{R}$$

is a function chosen uniformly from $\mathscr{C}$.

Note that choosing a function uniformly from a class $\{f_\alpha\}_{\alpha \in I}$ indexed by some finite set I can be achieved by choosing α_0 uniformly from I and then picking f_{α_0} as the chosen function.

Tweakable Random Permutation. When $\mathcal{R} = \mathcal{D}$, a popular choice of $\mathscr{C}$ is $\Pi_{\mathcal{D}}$, the class of all permutations on $\mathcal{D}$ (i.e., bijections from $\mathcal{D}$ to itself). A random permutation over $\mathcal{D}$ is a $\Pi_{\mathcal{D}}$-random function. It is an ideal choice corresponding to an encryption scheme over $\mathcal{D}$. The ideal choice corresponding to a tweakable enciphering scheme over $\mathcal{D}$ with tweak space $\mathcal{T}$ is called tweakable random permutation $\tilde{\pi}$ which is chosen uniformly from the class $\Pi_{\mathcal{D}}^{\mathcal{T}}$. For each tweak $\mathfrak{T} \in \mathcal{T}$, we choose a random permutation $\pi_{\mathfrak{T}}$ independently, and $\tilde{\pi}$ is a stochastically independent collection of random permutations $\{\pi_{\mathfrak{T}}; \mathfrak{T} \in \mathcal{T}\}$.

2.2 Pseudorandomness and Distinguishing Games

It should be noted that a random function or a random permutation is an ideal concept, since in practice the sizes of $\mathcal{R}^{\mathcal{D}}$ or $\Pi_{\mathcal{D}}$ are so huge that the cost of simulating a uniform random sampling on them is prohibitive. What is used instead of a truly random function is a **pseudorandom function** (PRF), a function whose behaviour is so close to that of a truly random function that no algorithm can effectively distinguish between the two. An adversary for a pseudorandom function f_1 is a deterministic algorithm $\mathcal{A}$ that tries to distinguish f_1 from a truly random f_0.

Security Notions. To test the pseudorandomness of f_1, $\mathcal{A}$ plays the **PRF distinguishing game** with an oracle $\mathcal{O}$ simulating (unknown to $\mathcal{A}$) either f_1 or f_0. For this, $\mathcal{A}$ makes q queries, in a deterministic but possibly adaptive manner. It is well known that there is no loss in assuming a distinguisher deterministic as unbounded time deterministic distinguisher is as powerful as a probabilistic distinguisher. Thus, the first query $x_1 = \mathfrak{q}_1()$ is fixed, and given the responses

$y_j = \mathcal{O}(x_j), j \in \{1, ..., i-1\}$, the i-th query becomes $x_i = \mathsf{q}_i(y_1, ..., y_{i-1})$, where q_i is a deterministic function for choosing the i-th query for $i \in \{1, ..., q\}$. Finally, a deterministic decision function examines $y_1, ..., y_q$ and chooses the output $b \in \{0, 1\}$ of $\mathcal{A}$. $\mathcal{A}$ wins if $\mathcal{O}$ was simulating f_b. An equivalent way to measure this winning event is called prf-advantage defined as

$$\Delta_{\mathcal{A}}(f_0\ ;\ f_1) := \mathbf{Adv}^{\mathrm{prf}}_{f_1}(\mathcal{A}) = |\Pr_{f_0}[\mathcal{A}^{f_0} \to 1] - \Pr_{f_1}[\mathcal{A}^{f_1} \to 1]|,$$

where $\Pr_f[.]$ denotes the probability of some event when $\mathcal{O}$ imitates f. The above definition can be extended for more than one oracles. We can analogously define **pseudorandom permutation** (PRP) advantage $\mathbf{Adv}^{\mathrm{prp}}_{f_1}(\mathcal{A})$ of f_1 in which case f_0 is the random permutation. When f_1 is an enciphering scheme and $\mathcal{A}$ is interacting with both f_1 and its inverse f_1^{-1} (or with f_0 and f_0^{-1}) we have **strong pseudorandom permutation** (SPRP) advantage

$$\mathbf{Adv}^{\mathrm{sprp}}_{f_1}(\mathcal{A}) = |\Pr_{f_0}[\mathcal{A}^{f_0, f_0^{-1}} \to 1] - \Pr_{f_1}[\mathcal{A}^{f_1, f_1^{-1}} \to 1]|.$$

Finally, for a tweakable enciphering schemes with the strong pseudorandom property as above, we analogously define the **tweakable strong pseudorandom permutation** (TSPRP) advantage $\mathbf{Adv}^{\mathrm{tsprp}}_{f_1}(\mathcal{A})$.

Pointless Adversaries. In addition to the adversary being deterministic, we also assume that it does not make any pointless queries. An adversary $\mathcal{A}$ making queries to a tweakable encryption scheme f and f^{-1} is called pointless if either it makes a duplicate query or it makes an f-query $(\mathfrak{T}, P)$ and obtains response C and f^{-1}-query $(\mathfrak{T}, C)$ and obtains response P (the order of these two queries can be reversed). We can assume that adversary is not pointless since the responses are uniquely determined for these types of queries.

Theorem 1. *[11] Let f_1 be a TES over a message space $\mathcal{M} \subseteq \{0,1\}^*$ and f_0 and f_0' be two independently chosen random functions. Then for any adversary non-pointless distinguisher $\mathcal{A}$ making at most q queries, we have,*

$$\mathbf{Adv}^{\mathrm{tsprp}}_{f_1}(\mathcal{A}) \leq \Delta_{\mathcal{A}}((f_1, f_1^{-1})\ ;\ (f_0, f_0')) + \frac{q(q-1)}{2^{m+1}}$$

where $m = \min\{\ell : \mathcal{M} \cap \{0,1\}^\ell \neq \emptyset\}$.

The above result says that an uniform length-preserving random permutation is very close to an uniform length-preserving random function.

2.3 Domain Extensions and Coefficient H Technique

The notion of pseudorandomness, while giving us an approximate implementation of random functions, introduces a new problem. In general, it is very hard to decide whether or not there is an adversary that breaks the pseudorandomness of a particular function, since there is no easy way of exhaustively covering all possible adversaries in an analysis, and since there is no true randomness in a practically implemented function, probabilistic arguments cannot be used.

The common get-around is to assume we have PRFs $f_1, ..., f_n$ each with domain $\mathcal{D}$ and use them to obtain an F with domain $\mathcal{D}' \supset \mathcal{D}$, such that a PRF-attack on F leads to a PRF-attack on one of $f_1, ..., f_n$. Now, there are known functions on small domains (like AES, for instance) which have withstood decades of attempted PRF-attacks and are believed to be reasonably secure against PRF-attacks. Choosing $\mathcal{D}$ suitably to begin with and using the known PRFs in our construction, we can find a PRF F with domain $\mathcal{D}'$ that is secure as long as the smaller functions are secure. This technique is known as a **domain extension**.

Here, the central step in a proving the security of F is the **reduction** of an adversary of F to an adversary of one of $f_1, ..., f_n$. This reduction is achieved by assuming $f_1, ..., f_n$ to be truly random, and giving an information-theoretic proof that the distinguishing advantage of any adversary at F is small. Thus, if an adversary thus distinguish F from random with a reasonable advantage, we must conclude that $f_1, ..., f_n$ are not truly random. Thus, all we need to show is that when the underlying functions are truly random, F behaves like a truly random function.

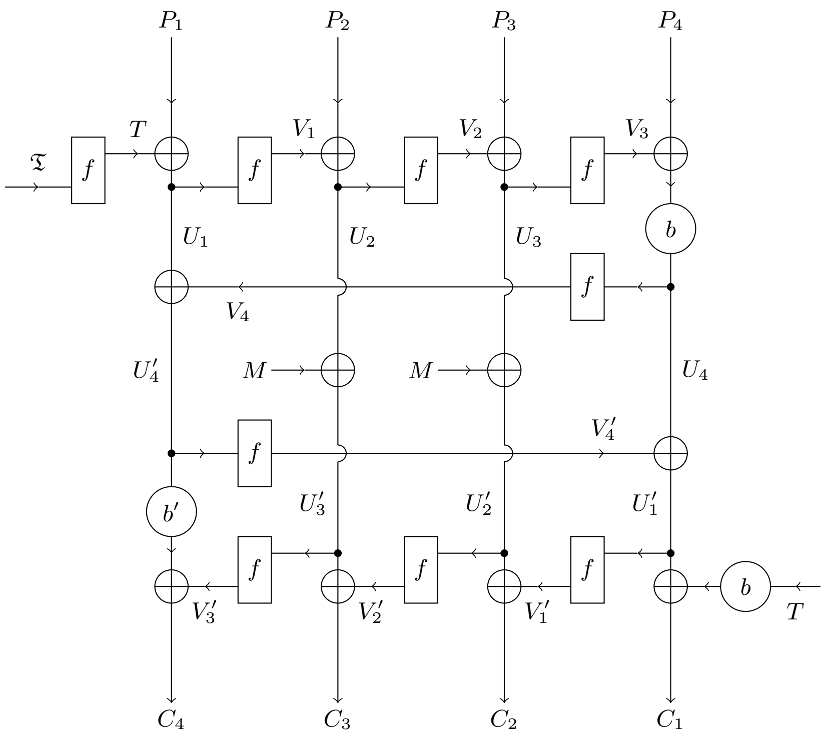

Fig. 3. The FMix construction for four blocks, with $M = V_4 + V_4'$

Patarin's Coefficient H Technique. There are several techniques for showing this. The one we use is based on the **Coefficient H Technique**, due to Jacques Patarin, which we briefly describe here. We look at the queries $x_1, ..., x_q$ and the outputs $y_1, ..., y_q$, and note that the adversary's decision will be based solely on the $2q$-tuple $(x_1, ..., x_q, y_1, ..., y_q)$. Now, if F_0 is the truly random function F is trying to emulate, then $F_0^{(q)}$ is also truly random, so on input $(x_1, ..., x_q)$, $F_0^{(q)}(x_1, ..., x_q)$ will be uniform over $\mathcal{R}'$, $\mathcal{R}$ being the range of F. Thus when $\mathcal{D}' = \mathcal{R}' = \{0,1\}^m$,

$$\Pr[F_0^{(q)}(x_1, ..., x_q) = (y_1, ..., y_q)] = \frac{1}{2^{mq}}.$$

If we can now show that $\Pr[F^{(q)}(x_1, ..., x_q) = (y_1, ..., y_q)]$ (which we call its **interpolation probability** after Bernstein) is "very close" to $\frac{1}{2^{mq}}$ for most $2q$-tuples $(x_1, ..., x_q, y_1, ..., y_q)$, we can conclude that no adversary can distinguish F from F_0 with a reasonable advantage. One way to formalize "very close" is that the interpolation probability is at least $(1-\epsilon)2^{-mq}$. Moreover, this may not happen for all possible views. (A view consists of all input and output blocks taken together. Informally, it is the portion of the computations visible to the adversary after completing all the queries.) So we may need to restrict the interpolation calculation on so called good views. This is the central idea of Patarin's technique.

Let $\mathsf{view}(\mathcal{A}^{\mathcal{O}})$ denote the the view obtained by the adversary $\mathcal{A}$ interacting with $\mathcal{O}$.

Theorem 2 (Coefficient H Technique[19]). *Suppose the interpolation probabilities follow the inequality*

$$IP^{f}_{\boldsymbol{FMix}}(\mathcal{V}) \geq (1-\epsilon) \cdot 2^{-nL}$$

for all views $\mathcal{V} \in \mathcal{V}_{good}$ (set of good views). Then for an SPRP-adversary $\mathcal{A}$, we have

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{SPRP}}(F) \leq \epsilon + \epsilon'$$

where ϵ' denotes the probability $\Pr[\mathit{view}(A^{F_0,F}) \notin \mathcal{V}_{good}]$.

This technique was first introduced by Patarin's PhD thesis[18] (as mentioned in [24]). Later it has been formalized in [19].

3 The FMix Construction

We are now in a position to describe our encryption scheme FMix. We use one underlying block function, chosen from a keyed family of PRFs $\{f_K : \{0,1\}^n \longrightarrow \{0,1\}^n\}_{K \in \mathcal{K}}$. The extended domain, which serves as both $\mathcal{M}$ and $\mathcal{C}$, is $\cup_{l \geq 2}\{0,1\}^{ln}$, all strings consisting of two or more n-bit blocks. In addition to a key and a plaintext, the encryption algorithm also takes a tweak $\mathfrak{T}$ as input, which is also supplied to the decryption algorithm. Encryption is length-preserving: for $m \in \{0,1\}^{l_0 m}$, $\mathfrak{e}(K, m, \mathfrak{T}) \in \{0,1\}^{l_0 m}$ as well. The basic structure of the construction is based on that of CMC: a CBC encryption layer, followed by a layer of mixing, followed by a CBC decryption layer. However, using a generalisation of the Feistel scheme, we eliminate the need for f_K^{-1} during decryption, making do with f_K instead, thus making this construction inverse-free.

```
input : A tweak $\mathfrak{T}$, an integer $l \geq 2$, $l$ plaintext blocks $P_1, ..., P_l$
output: $l$ ciphertext blocks $C_1, ..., C_l$
begin
    $T \leftarrow \texttt{f}(\mathfrak{T})$
    $V_0 \leftarrow T$
    for $i \leftarrow 1$ to $l-1$ do
        $U_i \leftarrow V_{i-1} \oplus P_i$
        $V_i \leftarrow \texttt{f}(U_i)$
    end
    $U_l \leftarrow \texttt{b}(V_{l-1} \oplus P_l)$
    $V_l \leftarrow \texttt{f}(U_l)$
    $U'_l \leftarrow V_l \oplus U_1$
    $V'_l \leftarrow \texttt{f}(U'_l)$
    $M \leftarrow V_l \oplus V'_l$
    $U'_{l-1} \leftarrow U_2 \oplus M$
    $V'_{l-1} \leftarrow \texttt{f}(U'_{l-1})$
    $C_l \leftarrow V'_{l-1} \oplus \texttt{b'}(U'_l)$
    for $i \leftarrow 3$ to $l-1$ do
        $U'_{l+1-i} \leftarrow U_i \oplus M$
        $V'_{l+1-i} \leftarrow \texttt{f}(U'_{l+1-i})$
        $C_{l+2-i} \leftarrow V'_{l+1-i} \oplus U'_{l+2-i}$
    end
    $U'_1 \leftarrow U_l + V'_l$
    $V'_1 \leftarrow \texttt{f}(U'_1)$
    $V'_0 \leftarrow T$
    $C_2 \leftarrow V'_1 \oplus U'_2$
    $C_1 \leftarrow \texttt{b}(V'_0) \oplus U'_1$
end
```

Algorithm 1: FMix Encryption Algorithm. The decryption algorithm is exactly same as the encryption except that the $b(T)$ is computed in the first layer and only T is used in the second.

The details of the construction are demonstrated in the figure, which shows a four-block FMix construction. The algorithm for general l is described in the box. Here, b is a balanced linear permutation, which we define below, and b' is b^{-1}. Decryption is almost identical, just with T and $b(T)$ switching roles.

Definition 2. *A permutation $b : \{0,1\}^n \longrightarrow \{0,1\}^n$ will be called a* **balanced linear permutation** *if both $t \mapsto b(t)$ and $t \mapsto t + b(t)$ are linear permutations.*

One choice of b could be multiplication by a primitive α, but this is not very software-friendly. A more software-friendly choice is $(t_1, t_2) \mapsto (t_1 \oplus t_2, t_1)$, where t_1 and t_2 are the higher and lower halves of t.

Notation for Our Construction. For our analysis we will assume the underlying PRF to be a truly random function f. We now model the encryption scheme in terms of computations based on f. An encryption is a computation

$$\mathbf{C} \longleftarrow \mathfrak{e}^f(\mathfrak{T}, \mathbf{P}),$$

where $\mathfrak{T} \in \{0,1\}^n$, and $\mathbf{P}, \mathbf{C} \in \{0,1\}^{ln}$ for some $l \geq 2$. Similarly, a decryption is a computation

$$\mathbf{P} \longleftarrow \mathfrak{d}^f(\mathfrak{T}, \mathbf{C}),$$

which inverts $\mathfrak{e}^f$, for any tweak $\mathfrak{T}$. The plaintext $\mathbf{P}$ is denoted $(P_1, ..., P_l)$, where each P_i is an n-bit block of $\mathbf{P}$. Simlarly, the ciphertext $\mathbf{C}$ is denoted $(C_1, ..., C_l)$.

In the TSPRP game, the adversary makes q queries to the oracle $\mathcal{O}$. Each query is of the form $(\delta, \mathfrak{T}, \mathbf{X})$, where $\delta \in \{e, d\}$ denotes the direction of the query, $\mathfrak{T} \in \{0,1\}^n$ is the tweak, and $\mathbf{X} \in \{0,1\}^{nl}$ for some l is the input. If $\mathcal{O}$ is imitating FMIX, $\mathcal{O}(e, \mathfrak{T}, \mathbf{X})$ returns $\mathcal{E}^f(\mathfrak{T}, \mathbf{X})$, and $\mathcal{O}(d, \mathfrak{T}, \mathbf{X})$ returns $\mathcal{D}^f(\mathfrak{T}, \mathbf{X})$. If $\mathcal{O}$ is imitating a tweaked PRP Π, $\mathcal{O}(e, \mathfrak{T}, \mathbf{X})$ returns $\Pi(\mathfrak{T}, \mathbf{X})$, and $\mathcal{O}(d, \mathfrak{T}, \mathbf{X})$ returns $\Pi^{-1}(\mathfrak{T}, \mathbf{X})$. The output of $\mathcal{O}$ is denoted $\mathbf{Y}$.

All the queries and their outputs taken together form what we call a **view**. We use the following notation in a view. For the i-th query, δ^i denotes the direction of the query, $\mathfrak{T}^i$ denotes the tweak, and l^i denotes the number of blocks in $\mathbf{X}$. When $\delta^i = e$, the blocks of $\mathbf{X}$ are denoted $P_1, ..., P_{l^i}$ and those of $\mathbf{Y}$ are denoted $C_1, ..., C_{l^i}$. When $\delta^i = d$, this notation is reversed, i.e., the blocks of $\mathbf{Y}$ are denoted $P_1, ..., P_{l^i}$ and those of $\mathbf{X}$ are denoted $C_1, ..., C_{l^i}$. In the analysis, the tweak $\mathfrak{T}$ is denoted both P_0^i and C_0^i.

4 TSPRP Security Analysis of FMix

4.1 Good Views and Interpolation

Our first task is to formulate the version of Patarin's Coefficient H Technique we shall use for our proof. We begin by restricting our attention to a particular class of views.

Pointless View. A view is an indexed set of tuples

$$\mathcal{V} = \{(\delta^i, \mathfrak{T}^i, l^i, P_j^i, C_j^i) | 1 \leq i \leq q, 1 \leq j \leq l^i\}.$$

Here δ^i can take values e and d only. The l^i's are positive integers and $\mathfrak{T}^i, P_j^i, C_j^i \in \{0,1\}^n$, called *blocks*. The P_j^i and C_j^i mean the j^{th} block of plaintext and ciphertext respectively on the i^{th} query. We denote $\mathfrak{T}^i$ by both P_0^i and C_0^i. For any $0 \leq a \leq b \leq l_i$, we write $P_{a..b}^i$ to represent the tuple $(P_a^i, \ldots, P_b^i)$ and P^i to denote $P_{0..l_i}^i$. Similar notation for C^i and $C_{a..b}^i$. A view $\mathcal{V}$ is said to be *pointless* if at least one of the followings holds:

1. $\exists i \neq i'$ such that $\delta^i = \delta^{i'} = e$, $P^i = P^{i'}$.
2. $\exists i \neq i'$ such that $\delta^i = \delta^{i'} = d$, $C^i = C^{i'}$.
3. $\exists i' < i$ such that $\delta^i = e$, $\delta^{i'} = d$, $P^i = P^{i'}$.
4. $\exists i' < i$ such that $\delta^i = d$, $\delta^{i'} = e$, $C^i = C^{i'}$.

The first two cases are for duplicate queries. The third holds when we obtain a response $P^{i'}$ for some decryption query $C^{i'}$ and then make an encryption query $P^i := P^{i'}$. (The fourth case is the third case with the order of the queries reversed.) It is easy to see that when an adversary $\mathcal{A}$ is interacting with a TES, the view obtained is pointless if and only if $\mathcal{A}$ is pointless.

As we do not allow a pointless adversary we can restrict ourselves to non-pointless views only. Now we define good and bad views among this class.

Definition 3 (Good and Bad Views). *A view* $\{(\delta^i, \mathfrak{T}^i, l^i, P^i_j, C^i_j) | 1 \leq i \leq q, 1 \leq j \leq l^i\}$ *is said to be* **good** *if it is not pointless and*

$$(\forall i \text{ with } \delta^i = e)(\nexists i' < i)(C^i_1 = C^{i'}_1), \ \textit{and} \ (\forall i \text{ with } \delta^i = d)(\nexists i' < i)(P^i_1 = P^{i'}_1).$$

A view that is not good and not pointless is called **bad**.

The proof revolves around showing that the good views have a near-random distribution, and the bad views occur with a low probability. For the rest of the analysis, we fix a good view $\mathcal{V}$.

Interpolation Probability. Now we consider the interpolation probability for FMix construction. It is easy to see that

$$\mathrm{IP}^f_{\mathsf{FMix}}(\mathcal{V}) = \Pr_f[\mathsf{FMix}^f(\mathfrak{T}^i, P^i_{1..l_i}) = C^i_{1..l_i}, 1 \leq i \leq q]$$

where the probability is taken under the randomness of f chosen uniformly from the set of all functions from $\{0,1\}^n$ to itself. Similarly, the interpolation probability for an ideal random function $\mathrm{IP}_*(\mathcal{V})$ is 2^{-nL} where $L = \sum_{i=1}^q l_i$. This corresponds to the case where $\mathcal{O}$ imitating a truly random function. Now we state a result the proof of which is deferred to the next section.

Proposition 1. *For any good view* $\mathcal{V}$*,*

$$IP^f_{\mathsf{FMix}}(\mathcal{V}) \geq (1-\epsilon) \times 2^{-nL}, \ \textit{where } \epsilon = \frac{\binom{2L}{2}}{2^n}.$$

Armed with this result and the Coefficient H Technique, we are now ready to state and prove the main result of this paper.

Theorem 3. *For any SPRP-adversary $\mathcal{A}$ making q queries with L blocks in all,*

$$\mathbf{Adv}^{\mathrm{tsprp}}_{\mathsf{FMix}}(\mathcal{A}) \leq \frac{\binom{2L}{2} + \binom{q}{2}}{2^n}.$$

Proof. When a non-pointless adversary $\mathcal{A}$ is interacting with a pair of independent random functions (f_0, f'_0), it obtains a bad view has probability upper bounded by $\binom{q}{2}$. To see this, let the bad event occurs for the first time at the i^{th} query. If it is an encryption query (similar proof can be carried out for the decryption query) then C_1^i is chosen randomly from $\{0,1\}^n$ and so it matches with one of the previous first ciphertext block is at most $(i-1)/2^n$. So $\Pr_{f_0,f'_0}[\mathsf{view}(\mathcal{A}^{f_0,f'_0})$ is a bad view$] \leq \sum_{i=1}^{q} \frac{i-1}{2^n} = \frac{q(q-1)}{2^{n+1}}$. By using Coefficient H Technique (see in section 2.3) and the proposition stated above we have proved our theorem. □

Corollary 1. *Let FMix_K denote the FMix construction based on the keyed blcokcipher f_K. For any TSPRP-adversary $\mathcal{A}$ making q queries with L blocks in all there exists an adversary $\mathcal{A}'$ making at most L encryption queries (and similar time as $\mathcal{A}$)*

$$\mathbf{Adv}^{\text{tsprp}}_{\mathsf{FMix}_K}(\mathcal{A}) \leq \mathbf{Adv}^{\text{prp}}_{f_K}(\mathcal{A}') + \frac{\binom{2L}{2} + \binom{q}{2}}{2^n}.$$

This follows from the standard hybrid argument.

4.2 Extension of FMix for Partial Block Input

In section 3, we define our construction only for complete block inputs. In practice, messages-lengths m may not be a multiple of block-length n. For a complete enciphering scheme, our message space needs to be extended to include these partial block inputs. Two known methods for message-space extension of a cipher were XLS[20] and Nandi's scheme[14]. XLS is now known to be insecure[15], so we use Nandi's generic scheme for extending the message-space. The generic construction requires two additional blockcipher keys. We write these blockciphers as f_2 and f_3. The blockcipher f_1 is used in FMix. Given any partial block x, $1 \leq |x| \leq n-1$, we write $\mathsf{pad}(x) = x1\|0^{n-1-|x|}$. Similarly, $\mathsf{chop}_r(x)$ denotes the first r bits of x.

```
input  : A tweak $\mathfrak{T}$, an integer $l \geq 2$, $l-1$ complete plaintext blocks $P_1, ..., P_{l-1}$,
         partial last plaintext block $p_l$
output: $l-1$ complete ciphertext blocks $C_1, ..., C_{l-1}$, partial last ciphertext block $c_l$
begin
    $P'_{l-1} \leftarrow f_2(\mathsf{pad}(p_l)) \oplus P_{l-1}$
    $(C_1, \ldots, C_{l-2}, C'_{l-1}) \leftarrow \mathsf{FMix}^{f_1}(P_1, \ldots, P_{l-2}, P'_{l-1})$
    $c_l \leftarrow \mathsf{chop}_{|p'_l|}(f_3(P'_{l-1} \oplus C'_{l-1})) \oplus p_l$
    $C_{l-1} \leftarrow f_2(\mathsf{pad}(c_l)) \oplus C'_{l-1}$
end
```

Theorem 4. *For any SPRP-adversary $\mathcal{A}$ making q queries with L blocks (including incomplete) in all,*

$$\mathbf{Adv}^{\text{tsprp}}_{\mathsf{FMix}}(\mathcal{A}) \leq \frac{\binom{2L}{2} + \binom{q}{2}}{2^n} + \frac{3q(q-1)}{2^{n+1}}.$$

The proof of the statement is immediate from Theorem 1 and the generic conversion as described in [14].

5 Proof of Proposition 1

In this section we provide the proof of Proposition 1.

Proposition. For any good view $\mathcal{V}$,

$$\mathrm{IP}^{f}_{\mathsf{FMix}}(\mathcal{V}) \geq (1-\epsilon) \times 2^{-nL} \text{ where } \epsilon = \frac{\binom{2L}{2}}{2^n}.$$

We find a lower bound for the probability on the left by counting the choices of f that give rise to $\mathcal{V}$. For this counting, we find the number of internal states (simulations) σ that can result in $\mathcal{V}$, and for each σ, the number of choices of f compatible with it. As it turns out, slightly undercounting the simulations (counting only what we call admissible simulations) will suffice to prove our security bound.

5.1 Simulations

We shall develop an effective way of calculating the interpolation probability of $\mathcal{V}$. We begin by introducing the notion of variables. Let E be the set of all encryption query indices, i.e., $E = \{i|\delta^i = e\}$. Similarly, let D be the set of all decryption query indices. In identifying and labelling internal blocks, we continue using superscripts to denote query indices. Thus, for a query i, the $2l^i$ inputs of f (other than $\mathfrak{T}^i$) are denoted $U^i_1, ..., U^i_{l^i}, U'^i_1, ..., U'^i_{l^i}$, and the $2l^i+1$ outputs of f are denoted $V^i_0, V^i_1, ..., V^i_{l^i}, V'^i_1, ..., V'^i_{l^i}$. For ease of notation, we shall write both U^i_0 and U'^i_0 to denote $\mathfrak{T}^i$.

Variables and Derivables. We pick a set of output blocks

$$\mathcal{S} = \{V^i_j | i \in E, j \in \{1, ..., l^i\}\} \cup \{V'^i_j | i \in D, j \in \{1, ..., l^i\}\}.$$

$\mathcal{S}$ will be our set of **primary variables**, or simply **variables**. Any non-trivial linear combination of variables, optionally including blocks from $\mathcal{V}$ as well, will be called a **derivable**. While the proof will be primarily depend on variables, derivables will serve in the proof mainly to simplify notation and make the proof easier to grasp. Examples of derivables would be U^2_3, $\sum_i V'^i_1$ and $V^2_2 + P^1_1$. Note that a linear combination of view blocks alone, say $C^3_2 + C^2_3$, will not be considered a derivable, since it's value has already been fixed by choosing $\mathcal{V}$.

Let us assume for now that the input block and its corresponding output block are unrelated. We note that all input and output blocks of f are either variables or derivables. Thus, if we assign values to the variables, all the inputs and outputs of f over all queries are linearly determined. Thus, the variables linearly generate the entire set of input and output blocks, while themselves being linearly independent. We now formalise the notion of value assignment to variables.

Definition 4. *A* **transcript** τ *is a collection of* **variable-value** *pairs* (Z, v) *such that no two pairs in the collection contain the same variable. For every* $(Z, v) \in \tau$*, the variable* Z *is said to be assigned the value* v *under* τ*. We denote this as* $Z|_\tau = v$*. The* **domain** $\mathbb{D}(\tau)$ *of a transcript* τ *is defined as* $\{Z|(\exists v)(Z, v) \in \tau\}$*. Given a set* S *of variables, a transcript* τ *with* $\mathbb{D}(\tau) = S$ *is said to be an* **instantiation of** S*.*

For a transcript τ and a derivable Z' whose value only depends on the variables in $\mathbb{D}(\tau)$, τ effectively determines a value for Z'. This value is denoted by $Z'||_\tau$. For ease of notation, for any view block X, $X||_\tau$ will simply denote the value of X fixed in $\mathcal{V}$. An instantiation σ of $\mathcal{S}$ will be called a **simulation**, since it determines all inputs and outputs of f and thus describes a complete simulation of the internal computations that resulted in view $\mathcal{S}$.

Not all simulations make sense, however, when we consider the connection between and input block and its corresponding output block. A dependence now creeps in among the variables, owing to the key observation below, which poses the only non-trivial questions in the entire proof.

Wherever the inputs of f are identical, so are its outputs.

There can be simulations which violate this rule, and thus describe internal computations that can never occur. A simulation which actually describes a possible set of internal computations is called **realisable**. It is immediately clear that our observation holds for all realisable simulations. The problem of calculating the interpolation probability of $\mathcal{V}$ boils down to counting the number of realisable simulations.

5.2 Admissibility

All realisable simulations can be difficult to count, however. We shall focus instead on a smaller class of simulations, called admissible simulations, which are easy to count and yet are abundant enough to give us the desired result. Before that, we let us formulate in specific terms the ramifications of this observation. The immediate consequence is what we call pre-destined collisions. Let $\mathcal{I} = \cup_i \{U_0^i, U_1^i, ..., U_{l^i}^i, U_1'^i, ..., U_{l_i}'^i\}$ be the set of all input blocks of f.

Definition 5. *A pair of input blocks* $Z_1, Z_2 \in \mathcal{I}$ *is said to constitute a* **predestined collision** *if for any realisable simulation* σ*,*

$$Z_1||_\sigma = Z_2||_\sigma.$$

All other collisions between input blocks are called **accidental collisions**. Our next task is to identify all pre-destined collisions. For that we'll need some more definitions.

Definition 6. *Query indices* i *and* i' *are called* k**-encryption equivalent** *for some* $k < \min(l^i, l^{i'})$ *if either* $i = i'$*, or*

$$(P_0^i, ..., P_k^i) = (P_0^{i'}, ..., P_k^{i'}).$$

*This is denoted as $i \sim_{e_k} i'$. Similarly, i and i' are called k-***decryption equivalent** *for some $k < \min(l^i, l^{i'})$ if either $i = i'$, or*

$$(C_0^i, ..., C_k^i) = (C_0^{i'}, ..., C_k^{i'}).$$

This is denoted as $i \sim_{d_k} i'$.

Note that if $i \sim_{e_k} i'$, then $(\forall k' < k)(i \sim_{e_{k'}} i')$, and similarly for decryption equivalence. Our choice of $\mathcal{V}$ as a good view ensures that $i \in E$ whenever $i \sim_{e_1} i'$ for some $i' < i$, and $i \in D$ whenever $i \sim_{d_1} i'$ for some $i' < i$. We can now make a list of pre-destined collisions:

– $(U_k^i, U_k^{i'}), 0 \le k < \min(l^i, l^{i'}), i \sim_{e_k} i'$
– $(U_k'^i, U_k'^{i'}), 0 \le k < \min(l^i, l^{i'}), i \sim_{d_k} i'$

Substituting $V_{k-1}^i + P_k^i$ for U_k^i and $V_{k-1}'^i + C_k^i$ for $U_k'^i$, we can re-write the pre-destined collisions as

– $(V_{k-1}^i + P_k^i, V_{k-1}^{i'} + P_k^{i'}), 0 \le k < \min(l^i, l^{i'}), i \sim_{e_k} i'$
– $(V_{k-1}'^i + C_k^i, V_{k-1}'^{i'} + C_k^{i'}), 0 \le k < \min(l^i, l^{i'}), i \sim_{d_k} i'$

List of Pre-destined Collision. By our Observation, a pre-destined collision on inputs naturally entails a collision on the corresponding outputs. This leads to a corresponding set of **pre-destined output collisions**, which we write in the form of equations over derivables and view blocks:

(a) $(i \sim_{e_k} i') \to (V_k^i = V_k^{i'}), 0 \le k < \min(l^i, l^{i'})$,
(b) $(i \sim_{d_k} i') \to (V_k'^i = V_k'^{i'}), 0 \le k < \min(l^i, l^{i'})$.

The pre-destined output collisions linearly follow from the pre-destined collisions, but are formulated separately here, because they'll later be useful as a class of constraints on realisable simulations. Finally, we define the class of admissible simulations.

Definition 7 (Admissible). *A simulation σ is called* **admissible** *if, for any $Z_1, Z_2 \in \mathcal{I}$ that do not constitute a pre-destined collision, $Z_1||_\sigma \neq Z_2||_\sigma$.*

Thus, in an admissible simulation, no two input blocks of f can accidentally collide, and the only collisions are the pre-destined ones.

5.3 Basis and Extension

We now identify a subclass B of the variables which are linearly independent under assumption of admissibility, and such that an instantiation τ_B of B admits a unique extension $\mathbb{E}(\tau_B)$ to a realisable simulation. We shall call B a **basis** of X. First, we'll need one more definition.

Definition 8. *A query index i, $1 \le i \le q$, is called k-***fresh***, $k \ge 0$ if $k = l^i$, or $k < l^i$ and $\nexists i' \le i$ with $k < l^{i'}$ such that $i \sim_{e_k} i'$ or $i \sim_{d_k} i'$.*

The set E_k of k-fresh encryption queries is defined as $\{i|\delta^i = e, i\ k\text{-fresh}\}$. Similarly, the set D_k of k-fresh decryption queries is defined as $\{i|\delta^i = d, i\ k\text{-fresh}\}$. Clearly, $E = \cup_k E_k$, and $D = \cup_k D_k$, since any i is l^i-fresh.

We are now in a position to choose our basis B. Let $l = \max_i l^i$. We define the following:

$$B_j = \{V_j^i | i \in E_j\}, 0 \le j \le l,$$

$$B'_j = \{V_j'^i | i \in D_j\}, 0 \le j \le l.$$

Finally, we define our basis as

$$B = \bigcup_{j=0}^{l} (B_j \cup B'_j).$$

We next show how to obtain $\sigma = \mathbb{E}(\tau_B)$ given instantiation τ_B of B. To simplify the description, we shall use a couple of new definitions.

Definition 9. *The* **encryption k-ancestor** *of a query index i is defined as*

$$A_k^e(i) = \min_{i \sim_{e_k} i'} i'.$$

Similarly, the **decryption k-ancestor** *of a query index i is defined as*

$$A_k^d(i) = \min_{i \sim_{d_k} i'} i'.$$

Clearly, if i is k-fresh, then i is its own k-ancestor.

Definition 10. *For a query index i and a transcript τ, the* **query slice at i of τ** *is defined as*

$$Q_i(\tau) = \{(Z^i, v) | (Z^i, v) \in \tau\}.$$

Thus, a query slice is the portion of a transcript that refers to a specific query. The query slices of a transcript form a partition of it.

We are now ready to describe how to uniquely obtain σ. To begin with, for all $Z \in B$, we set

$$Z|_\sigma = Z|_{\tau_B}.$$

This gives us, among other things, the complete $Q_1(\sigma)$. (To see why, assume without loss of generality that $\delta^1 = e$. Then $1 \in E_j$ for every j, so $V_j^1 \in B$ for $0 \le j \le l^1$.) We proceed inductively to determine $Q_i(\sigma)$ based on $Q_1(\sigma), ..., Q_{i-1}(\sigma)$.

Suppose we have determined $Q_{i'}(\sigma)$ for all $i' < i$. For $0 \le j \le l^i$, let i_j denote $A_j^{\delta^i}(i)$. Clearly, $\{i_j\}_j$ form a non-decreasing sequence, and $i_{l^i} = i$. Let

$$k = \min_{i_j = i} j.$$

Suppose without loss of generality that $\delta^i = d$. Thus, for all $j \geq k$, $i \in D_j$. So $V_j'^i \in B$ for every $k \leq j \leq l^i$. For $0 \leq j < k$, since $i \sim_{d_j} i_j$, and i_j is decryption j-fresh, we use 4.3 (b) to set

$$V_j'^i|_\sigma = V_j'^{i_j}||_\sigma.$$

Finally, we set

$$V_0^i|_\sigma = V_0'^i||_\sigma.$$

This completes our extension of τ_B.

To show that σ indeed is a simulation, we just observe that if $\cup_1^{i-1} Q_i(\sigma)$ is realisable, and $\delta^i = d$, then $Q_i(\sigma)$ cannot violate 4.3 (a) (which concerns encryption queries only), and $Q_i(\sigma)$ is chosen so as to conform to 4.3 (b).

5.4 Extension Equations

We observe that in extending τ_B to $\mathbb{E}(\tau_B)$, once we've set the basis variables in accordance with τ_B, none of the steps we perform thereafter depend on the specific instantiation τ_B. Thus, for each variable we can identify an equation relating it to the basis variables, so that a simulation can be obtained by simply plugging in an appropriate instantiation of B. We call these equations the **extension equations**.

Pick $i \in E, j \in \{0, ..., l^i\}$. Then V_j^i is a variable. Let b_1 be j, and a_1 be $A_j^e(i)$. Having obtained $b_1, ..., b_k$ and $a_1, ..., a_k$, we stop if k is odd and $a_k \in E$, or if k is even and $a_k \in D$. Otherwise, let $b_{k+1} = l^{a_k} - 1 - b_k$, and a_{k+1} be $A_{b_{k+1}}^{\delta^{a_k}}(a_k)$. Since $a_{k+1} > a_k$, this terminates after finitely many steps, say upon obtaining a_{k_0}. Then we call $((b_1, a_1), ..., (b_{k_0}, a_{k_0}))$ the **extension chain** of V_j^i, denoted $\mathfrak{C}(V_j^i)$.

To obtain the extension equation of V_j^i from $\mathfrak{C}(V_j^i)$, note that $V_j^i = V_j^{a_1}$, and for any even $k \leq k_0$, $V_j'^{a_k} = V_j'^{a_{k-1}}$, and (if $k < k_0$) $V_j^{a_{k+1}} = V_j^{a_k}$. To bridge these equations, we just need to recall the equations relating $V_j^{i'}$ to $V_{l^{i'}-1-j}'^{i'}$ for arbitrary i' with $l^{i'} \geq j$.

From our algorithm, $V_0'^{i'} = V_0^{i'}$, $V_{l^{i'}}'^{i'} = b(V_{l^{i'}-1}^{i'} + V_0^{i'} + P_{l^{i'}}^{i'}) + C_1^{i'}$ and $V_{l^{i'}-1}'^{i'} = b(V_{l^{i'}}^{i'} + V_0^{i'} + P_1^{i'}) + C_{l^{i'}}^{i'}$.

For $1 \leq j \leq l^{i'} - 2$, recall the masking equation

$$V_j'^{i'} = V_{l_i-j-1}^{i'} + V_{l^{i'}}^{i'} + V_{l^{i'}}'^{i'} + P_{l^{i'}-j}^{i'} + C_{j+1}^{i'}.$$

On replacing $V_{l^{i'}}'^{i'}$ by $b(V_{l^{i'}-1}^{i'} + V_0^{i'} + P_{l^{i'}}^{i'}) + C_1^{i'}$, this becomes

$$V_j'^{i'} = V_{l_i-j-1}^{i'} + V_{l^{i'}}^{i'} + b(V_{l^{i'}-1}^{i'} + V_0^{i'} + P_{l^{i'}}^{i'}) + C_1^{i'} + P_{l^{i'}-j}^{i'} + C_{j+1}^{i'}.$$

The extension equations can be computed inductively using the above. Similarly, for derivables, we can get the extension equations by writing it in terms of variables, and expanding these variables through their corresponding extension equations. We'll mostly be interested in the set of basis variables appearing in the extension equation of an input derivable Z, called the **base** $\mathfrak{B}(Z)$ of Z.

We'll show that whenever for two input derivables Z and Z', $\mathfrak{B}(Z) = \mathfrak{B}(Z')$, (Z, Z') is either a pre-destined collision, or Z and Z' cannot collide. This'll show that every accidental input collision corresponds to a linear equation on the basis variables and view blocks. Note that this linear equation actually corresponds to n linear equations in terms of the bits, all of which should be dodged. For most of the analysis, this distinction will not matter, and it'll only become important when we deal with two special cases in the very end.

Lemma 1. *Every accidental input collision imposes a non-trivial linear equation on the basis variables.*

The proof of the lemma is postponed to the end of this section. It basically considers all cases for accidental collision and shows that it gives a non-trivial linear equation.

5.5 Bringing it all Together

We are now ready to wrap up our proof of the proposition 1. Let L denote $\sum_i l^i$. Let $\epsilon = \frac{\binom{2L}{2}}{2^n}$. The total number of output bits $\mathcal{V}$ in is nL, so clearly

$$\mathrm{IP}_*(\mathcal{V}) = \frac{1}{2^{nL}}.$$

Now, let $\mathcal{F} \subset (\{0,1\}^n)^{\{0,1\}^n}$ be such that $(f \in \mathcal{F}) \longleftrightarrow$ (choosing f results in $\mathcal{V}$). We see that

$$\mathrm{IP}^f_{\mathsf{FMix}}(\mathcal{V}) = \frac{\#\text{ choices of } f \text{ which result in } \mathcal{V}}{\#\text{ choices of } f \text{ in all}} = \frac{|\mathcal{F}|}{(2^n)^{2^n}}.$$

Let $\mathfrak{A}$ be the set of all admissible simulations. For an admissible simulation σ, let $\mathcal{F}_\sigma$ denote the subset of $\mathcal{F}$ such that $(f \in \mathcal{F}_\sigma) \longleftrightarrow$ (choosing f results in $\mathcal{V}$ and σ). With this notation, we can write

$$|\mathcal{F}| \geq \sum_{\sigma \in \mathfrak{A}} |\mathcal{F}_\sigma|.$$

To calculate $|\mathcal{F}_\sigma|$, we note that σ fixes the values f for $L + |B|$ distinct inputs. Thus,

$$|\mathcal{F}_\sigma| = (2^n)^{2^n - L - |B|}.$$

Since this does not depend on σ, we can write

$$\sum_{\sigma \in \mathfrak{A}} |\mathcal{F}_\sigma| = |\mathfrak{A}| \cdot (2^n)^{2^n - L - |B|}.$$

Now, each admissible simulation is $\mathbb{E}(\tau_B)$ for some instantiation τ_B of B. To ensure $\mathbb{E}(\tau_B) \in \mathfrak{A}$, we just have to choose τ_B such that it dodges all the linear equations corresponding to accidental input collisions. As there can be at most $\binom{2L}{2}$ such equations, we conclude that

$$|\mathfrak{A}| \geq 2^{n|B|} - \binom{2L}{2} \cdot 2^{n(|B|-1)} = 2^{n|B|}(1-\epsilon).$$

Putting all of this together, we get

$$|\mathcal{F}| \geq (2^n)^{2^n - L} \cdot (1-\epsilon) = (1-\epsilon) \cdot \mathrm{IP}_*(\mathcal{V}) \cdot (2^n)^{2^n},$$

from which the Proposition follows.

5.6 Proof of Lemma 1

Proof. We'll divide the possible input pairs into several cases, which we'll further subdivide into groups, and we write out the proof only for the first case in each group, and the rest follow from it. The classifying factors are as follows:

- Whether they both occur in the same layer (**encryption layer** $\{U_j^i\}$ or **decryption layer** $\{U_j^i\}$), or in different layers;
- Whether they occur in the **right layer** (encryption layer of an encryption query, or decryption layer of decryption query) or the **wrong layer**;
- Whether their **first-cross indices** match (this would be the current query index if in the wrong layer, and the index after the first backward jump during extension if in the right layer).

We begin with an easy group of cases, where both occur in the right layer, and their first-cross indices do not match:

Case 1a. $(U_j^i, U_{j'}^{i'}), i, i' \in E, a = A_{j-1}^e(i) < A_{j'-1}^e(i') = a'$

$\mathfrak{B}(U_j^i) = \mathfrak{B}(V_{j-1}^i)$ can only contain basis variables with query indices $\leq a$. Since $\mathfrak{B}(U_{j'}^{i'}) = \mathfrak{B}(V_{j'-1}^{i'})$ will contain either $V_{l^{a'}}^{\prime a'}$ or $V_{j'}^{a'}$, $\mathfrak{B}(U_j^i) \neq \mathfrak{B}(U_{j'}^{i'})$.

Case 1b. $(U_j^{\prime i}, U_{j'}^{\prime i'}), i, i' \in D, a = A_{j-1}^d(i) < A_{j'-1}^d(i') = a'$

Case 1c. $(U_j^i, U_{j'}^{\prime i'}), i \in E, i' \in D, a = A_{j-1}^e(i) < A_{j'-1}^d(i') = a'$

Case 1d. $(U_j^{\prime i}, U_{j'}^{i'}), i \in D, i' \in E, a = A_{j-1}^d(i) < A_{j'-1}^e(i') = a'$

We next turn to another easy group, where exactly one of them is in the right layer, and first-cross indices do not match:

Case 2a. $(U_j^i, U_{j'}^{\prime i'}), i, i' \in E, a = A_{j-1}^e(i) \neq i'$

If $a < i'$, $V_{l^{i'}}^{i'}$ is in $\mathfrak{B}(U_{j'}^{\prime i'})$ but not in $\mathfrak{B}(U_j^i)$. If $a > i'$, either V_{j-1}^a is in $\mathfrak{B}(U_j^i)$ but not in $\mathfrak{B}(U_{j'}^{\prime i'})$, or $V_{l^a}^{\prime a}$ is in $\mathfrak{B}(U_j^i)$ but not in $\mathfrak{B}(U_{j'}^{\prime i'})$.

Case 2b. $(U_j^i, U_{j'}^{\prime i'}), i, i' \in D, i \neq A_{j'-1}^d(i') = a'$

Case 2c. $(U_j^i, U_{j'}^{i'}), i \in E, i' \in D, a = A_{j-1}^e(i) \neq i'$

Case 2d. $(U'^i_j, U'^{i'}_{j'}), i \in E, i' \in D, i \neq A^d_{j'-1}(i') = a'$

The next group is even easier: both in the wrong layer, with non-matching first-cross indices. This takes care of all cases with non-matching first-cross indices.

Case 3a. $(U^i_j, U^{i'}_{j'}), i, i' \in D, i < i'$

$V'^{i'}_{l^{i'}}$ is in $\mathfrak{B}(U^{i'}_{j'})$ but not in $\mathfrak{B}(U^i_j)$.

Case 3b. $(U'^i_j, U'^{i'}_{j'}), i, i' \in E, i < i'$

Case 3c. $(U^i_j, U'^{i'}_{j'}), i \in D, i' \in E, i < i'$

Case 3d. $(U'^i_j, U^{i'}_{j'}), i \in E, i' \in D, i < i'$

Next we turn to a slightly trickier group, where they are in the same layer, both in the right layer, and first-cross indices match.

Case 4. $(U^i_j, U^{i'}_{j'}), i, i' \in E, A^e_{j-1}(i) = A^e_{j'-1}(i')$

Consider $\mathfrak{C}(V^i_{j-1}) = ((b_1, a_1), ..., (b_{k_0}, a_{k_0}))$, $\mathfrak{C}(V^{i'}_{j'-1}) = ((b'_1, a'_1), ..., (b'_{k'_0}, a'_{k'_0}))$. If the chains follow the same query paths (i.e., if $k_0 = k'_0$ and $(\forall k \leq k_0)(a_k = a'_k)$), assuming without loss of generality k_0 is odd and $k_0 \in E$ (from the chain-termination condition), we have $V^{a_{k_0}}_{b_{k_0}} \in \mathfrak{B}(U^i_j)$, and $V^{a_{k_0}}_{b'_{k_0}} \in \mathfrak{B}(U^{i'}_{j'})$, all other basis variables in the two extension equations being the same. Thus, if $b_{k_0} \neq b'_{k_0}$, $\mathfrak{B}(U^i_j) \neq \mathfrak{B}(U^{i'}_{j'})$, and if $b_{k_0} = b'_{k_0}$, $(U^i_j, U^{i'}_{j'})$ is either a pre-destined collision (if $P^i_j = P^{i'}_{j'}$) or it cannot be a collision. If the chains do not follow the same query path, we can find k such that $a_k \neq a'_k$, which reduces to one of the previous cases.

Case 4a. $(U'^i_j, U'^{i'}_{j'}), i, i' \in D, A^d_{j-1}(i) = A^d_{j'-1}(i')$

The next group is much simpler, where they are in different layers, both in the right layer, and first-cross indices match.

Case 5. $(U^i_j, U'^{i'}_{j'}), i \in E, i' \in D, a = A^e_{j-1}(i) = A^d_{j'-1}(i')$

Without loss of generality, $a \in E$. So $V^a_{l^a}$ is in $\mathfrak{B}(U^{i'}_{j'})$ but not in $\mathfrak{B}(U^i_j)$.

Case 5a. $(U'^i_j, U^{i'}_{j'}), i \in D, i' \in E, A^d_{j-1}(i) = A^e_{j'-1}(i')$

We're almost done with the proof at this point. We wrap up with the few remaining cases. In the next group, they come from different layers, exactly one of them in the right layer, and first-cross indices match.

Case 6. $(U^i_j, U'^{i'}_{j'}), i, i' \in E, A^e_{j-1}(i) = i'$

Here, $V^{i'}_{l^{i'}}$ is in $\mathfrak{B}(U^{i'}_{j'})$ but not in $\mathfrak{B}(U^i_j)$.

Case 6a. $(U^i_j, U'^{i'}_{j'}), i, i' \in D, i = A^d_{j'-1}(i')$

The four cases of the final group can be proved using the extension-chain-comparison technique of Case 4. In this group, they are in the same layer, at least one in the wrong layer, and first-cross indices match. (If they are both in the wrong layer, and first-cross indices match, they occur at the same query, so they cannot be in different layers, so this wraps up the case analysis.)

Case 7. $(U^i_j, U^{i'}_{j'}), i \in E, i' \in D, A^e_{j-1}(i) = i'$

Case 7a. $(U'^i_j, U'^{i'}_{j'}), i \in E, i' \in D, i = A^d_{j'-1}(i')$

Case 7b. $(U^i_j, U^i_{j'}), i \in D$

Case 7c. $(U'^i_j, U'^i_{j'}), i \in E$

This leaves only a few boundary cases (involving the likes of $U^i_{l^i}$), which can be easily verified. We just point out two special cases which underline the importance of choosing b as a balanced permutation. For the pair (U^i_1, U'^i_1) for some i, if $P^i_1 = C^i_1$, the condition for an accidental collision becomes $V^i_0 + b(V^i_0) = 0$, which is still n independent linear equations in terms of the bits, by choice of b. Similarly, if $i \sim_{e_{l^i-1}} i'$, and $b(P^i_{l^i}) = P^{i'}_{l^i}$, the pair $(U^i_{l^i}, U^{i'}_{l^i})$ yields the equation $b(V^i_{l^i-1}) + V^{i'}_{l^i-1} = 0$, which again is n independent linear equations in terms of the bits.

Thus we establish our lemma. □

6 Conclusion and Future Works

In this paper we propose a new Feistel type length preserving tweakable encryption scheme. Our construction, called FMix, has several advantages over CMC and other blockcipher based enciphering scheme. It makes an optimal number of blockcipher calls using single keyed PRP blockcipher. The only drawback compare to EME is that the first layer of encryption, like CMC, is sequential. We can view our construction as a composition of type-1 and type-3 Feistel ciphers.

There are several possible scopes of future work. When we apply a generic method to encrypt last partial block message, we need an independent key. (This is always true for generic construction.) However, one can have a very specific way to handle partial block message keeping only one blockcipher key. The presence of the function b helps us to simplify the security proof. However, we do not know of any attack if we do not use this function (except for handling the tweak in the bottom layer - that use is necessary). So it would be interesting to see whether our proof can be extended for the variant without using the function b.

References

1. ThierryP. Berger, Marine Minier, and Gal Thomas. Extended generalized feistel networks using matrix representation. In Tanja Lange, Kristin Lauter, and Petr Lisonk, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 289–305. Springer Berlin Heidelberg, 2014.
2. Debrup Chakraborty and Palash Sarkar. Hch: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer Berlin Heidelberg, 2006.

3. Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew Robshaw, editor, *Fast Software Encryption*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer Berlin Heidelberg, 2006.
4. H. Feistel. Block cipher cryptographic system, March 19 1974. US Patent 3,798,359.
5. Shai Halevi. Eme*: Extending eme to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer Berlin Heidelberg, 2005.
6. Shai Halevi. Invertible universal hashing and the tet encryption mode. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer Berlin Heidelberg, 2007.
7. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer Berlin Heidelberg, 2003.
8. Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *Topics in Cryptology CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer Berlin Heidelberg, 2004.
9. VietTung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption aez and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer Berlin Heidelberg, 2015.
10. VietTung Hoang and Phillip Rogaway. On generalized feistel networks. In Tal Rabin, editor, *Advances in Cryptology CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 613–630. Springer Berlin Heidelberg, 2010.
11. Moses Liskov, RonaldL. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin Heidelberg, 2002.
12. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, April 1988.
13. DavidA. McGrew and ScottR. Fluhrer. The security of the extended codebook (xcb) mode of operation. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327. Springer Berlin Heidelberg, 2007.
14. Mridul Nandi. A generic method to extend message space of a strong pseudorandom permutation. *Computación y Sistemas*, 12(3):285–296, 2009.
15. Mridul Nandi. Xls is not a strong pseudorandom permutation. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 478–490. Springer Berlin Heidelberg, 2014.
16. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Lubyrackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.
17. Kaisa Nyberg. Generalized feistel networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 91–104. Springer Berlin Heidelberg, 1996.
18. J. Patarin. Etude des Générateurs de Permutations Basés sur le Schéma du D.E.S. Phd Thèsis de Doctorat de l'Université de Paris 6, 1991.
19. Jacques Patarin. The coefficients h technique. In RobertoMaria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer Berlin Heidelberg, 2009.

20. Thomas Ristenpart and Phillip Rogaway. How to enrich the message space of a cipher. In Alex Biryukov, editor, *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 101–118. Springer Berlin Heidelberg, 2007.
21. Palash Sarkar. Improving upon the tet mode of operation. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007*, volume 4817 of *Lecture Notes in Computer Science*, pages 180–192. Springer Berlin Heidelberg, 2007.
22. Arthur Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia*, 8(1):22–42, 1984.
23. Data Encryption Standard. Fips pub 46. *Appendix A, Federal Information Processing Standards Publication*, 1977.
24. S. Vaudenay. Decorrelation: A theory for block cipher security. In *Journal of Cryptology 16(4)*, Lecture Notes in Computer Science, pages 249–286, New York, 2003. Springer-Verlag.
25. Peng Wang, Dengguo Feng, and Wenling Wu. Hctr: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer Berlin Heidelberg, 2005.