Tweakable Enciphering Schemes From Stream Ciphers With IV

Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: palash@isical.ac.in

Abstract. We present the first construction of a tweakable enciphering scheme from a stream cipher supporting an initialization vector. This construction can take advantage of the recent advances in hardware efficient stream ciphers to yield disk encryption systems with a very small hardware footprint. Such systems will be attractive for resource constrained devices.
Keywords: stream cipher with IV, tweakable encryption, disk encryption.

1 Introduction

A length preserving permutation is a bijective function from binary strings to binary strings such that the lengths of the input and the output strings are equal. A length preserving encryption function is a family $\{\mathbf{E}_K\}_{K \in \mathcal{K}}$ of length preserving permuations which is indexed by a secret key K. Both $\mathbf{E}_K$ and its inverse $\mathbf{E}_K^{-1}$ have to be efficiently computable for every value of K. The security requirement is that for a uniform random K, the pair of functions $\mathbf{E}_K, \mathbf{E}_K^{-1}$ is computationally indistinguishable from $\mathbf{\Gamma}, \mathbf{\Gamma}^{-1}$, where $\mathbf{\Gamma}$ is a uniform random length preserving permutation of the set of all binary strings, i.e., for each possible i, $\mathbf{\Gamma}$ restricted to binary strings of length i is a uniform random permutation of $\{0,1\}^i$.

The above extends the notion of strong pseudorandom permutation (SPRP) that was introduced by Luby and Rackoff [11] to handle variable length strings. This extension was introduced by Halevi and Rogaway [7]. They also incorporated the idea of tweak from [10] into the above definition. A tweak is an additional input to $\mathbf{E}_K$ and the idea of the tweak is to provide for flexibility in applications. A tweakable length preserving encryption function satisfying the above notion of security is called a tweakable enciphering scheme (TES). An important practical application of a TES is that of disk encryption. In this application, the disk is encrypted sector-wise and the sector address corresponds to the tweak. Till date all known constructions of TES have used a block cipher in a particular mode of operation. There are several such constructions [7, 8, 12, 19, 4, 13, 17, 16, 18]. Out of these the constructions in [17, 18] are the fastest and [18] does not require the decryption module of the underlying block cipher.

Our Contributions. We present the first construction of a TES using a stream cipher with an initialization vector (IV). This has important consequences to the practical problem of disk encryption. In this application, the implementation is typically in hardware and the module resides just above the disk controller in the overall architecture.

Stream ciphers are nowadays designed for two widely different environments – for extremely fast software implementation and for very small hardware foot print. In the recently concluded estream [1] selection these are called Profile 1 and Profile 2 respectively. There are some hardware efficient stream ciphers in Profile 2. Our construction makes it possible to use such stream ciphers to build a TES for disk encryption system. Overall this would lead to a hardware which is significantly

smaller than any hardware implementing a block cipher. The new construction can also be used with Profile 1 stream ciphers providing very fast software based solutions.

The construction makes two calls to the stream cipher with three different IVs. These calls are made within a Feistel structure which builds a $2n$-bit to $2n$-bit permutation, where n is the length of the IV of the stream cipher. From the first of these calls, a string of sufficient length is generated to encrypt the portion of the message after the first $2n$ bits. Typically, a stream cipher with IV requires an initialization phase which can take much more time compared to the key generation phase. Consequently, it is useful to keep the number of initializations as small as possible. The two calls to the stream cipher that the TES construction makes require two initializations. This does not impact the performance too much.

Apart from the stream cipher, the TES construction uses an almost XOR universal (AXU) hash function. The role of this hash function is similar to that played in the block cipher based construction [18]. The differences are in the way the hashing keys are derived and the manner in which the tweak and the length are handled. When working with a block cipher, it is easy to apply the block cipher encryption to a string and consider the output to be a hash key. Doing the same with an IV based stream cipher can be time consuming, since it will involve a stream cipher initialization. This necessitated some changes to the actual hash function design compared to [18].

An AXU hash function can be instantiated by usual polynomial hashing or the more efficient hashing introduced by Bernstein [3] based on earlier work by Rabin and Winograd [14] which is called BRW hashing. BRW hashing requires about half the number of multiplications compared to usual polynomial hashing. Another design of AXU hash function [15] uses a tower field representation of $GF(2^n)$ and does not require any multiplication at all. The trade-off, however, is that the size of the hashing key is as long as the message. Later we show how to tackle this in the context of disk encryption. Combining a hardware efficient stream cipher with the AXU hash function from [15] gives rise to an efficient TES with a very small hardware footprint. This will be attractive for implementing disk encryption on resource constrained systems such as portable and handheld devices.

2 Construction

We follow the notation used in [17, 18]. $\mathsf{First}_r(Z)$ denotes the most significant r bits of the n-bit binary string Z and $\mathsf{pad}_i(Z)$ denotes the string obtained from Z by appending i zero bits; for $0 \leq \ell \leq 2^n - 1$, $\mathsf{bin}_n(\ell)$ denotes the n-bit binary representation of ℓ.

Let $\mathbb{F} = GF(2^n)$ be the finite field of 2^n elements. The addition operation over $\mathbb{F}$ will be denoted by $\oplus$. Elements of $\mathbb{F}$ can also be considered to be n-bit strings. Let $h : \mathcal{K} \times \mathbb{F}^m \rightarrow \mathbb{F}$ be a function satisfying the following three properties.

1. For $\mathbf{X}, \mathbf{X}' \in \mathbb{F}^m$, $\mathbf{X} \neq \mathbf{X}'$ and for any $\gamma \in \mathbb{F}$, $\Pr_\tau[h_\tau(\mathbf{X}) \oplus h_\tau(\mathbf{X}') = \gamma] \leq \epsilon$, where $h_\tau(\mathbf{X}) \stackrel{\Delta}{=} h(\tau, \mathbf{X})$ and the probability is over uniform random choice of τ. Such an h is called an ϵ-almost XOR universal (AXU) hash function.
2. The function h has to satisfy an ϵ-uniformity property: for any non-zero $\mathbf{X} \in \mathbb{F}^m$ and $\gamma \in \mathbb{F}$, $\Pr_\tau[h_\tau(\mathbf{X}) = \gamma] \leq \epsilon$.
3. If τ and τ' are independent and uniform random elements of $\mathcal{K}$; X and X' are both not zero and Y is any element of $\mathbb{F}$, then $\Pr_{\tau,\tau'}[h_\tau(X) \oplus h_{\tau'}(X') = Y] \leq \epsilon$. A similar property has been used in [9].

Possible instantiations of h which satisfy the above properties are usual polynomial hashing, BRW polynomials [3] and the constructions from [15]. Note that ϵ could depend on m (correspondingly, we

write ϵ_m). Usual polynomial hashing requires m multiplications and has $\epsilon_m = m/2^n$. If the BRW [3] polynomials are used to instantiate h then $\max(1, \lfloor m/2 \rfloor)$ multiplications are required (assuming that the terms $\tau^2, \tau^4, \ldots$ are pre-computed and available) and $\epsilon_m = m/2^{n-1}$. The instantiations of h from [15] has $\epsilon_m = 1/2^n$ for all m and does not require any multiplications; but, the length of the key τ is as long as the message. Later we discuss how to tackle this issue in the context of TES. For an ϵ-AXU function, this property is satisfied if h is linear which is the case for the above mentioned instantiations.

A hash function with double block output [18]. Define $\Upsilon_\tau : \cup_{i \geq 3} \mathbb{F}^i \rightarrow \mathbb{F}^2$ as follows

$$\Upsilon_\tau(X_1, X_2, X_3, \ldots, X_m) = (X_1 \oplus Z, X_2 \oplus Z) \tag{1}$$

where $Z = h_\tau(X_3, \ldots, X_m)$. It is not difficult to show that for $\mathbf{X} \neq \mathbf{X}'$, $\Pr_\tau[\Upsilon_\tau(\mathbf{X}) = \Upsilon_\tau(\mathbf{X}')] \leq \epsilon_{m-2}$. If $(A_1, A_2) = \Upsilon_\tau(X_1, X_2, X_3, \ldots, X_m)$, then $(X_1, X_2) = \Upsilon_\tau(A_1, A_2, X_3, \ldots, X_m)$, a property which is required for decryption.

Stream cipher with IV. Let $\mathrm{SC}_K : \{0,1\}^n \rightarrow \{0,1\}^L$ be a stream cipher with IV, i.e., for every choice of K, SC_K maps an IV of length n bits to a string of length L bits. The length L is assumed to be long enough for practical sized messages to be encrypted. Actual encryption of a plaintext P of ℓ bits with an IV V is done by XORing the first ℓ bits of $\mathrm{SC}_K(V)$ to the message. By a slight abuse of notation, we will write this as $P \oplus \mathrm{SC}_K(V)$. The key K is from a suitable key space and there is no restriction on this key space. The security proof will assume $\{\mathrm{SC}_K\}$ to be a family of pseudo-random functions, so that the output $\mathrm{SC}_K(V)$ can be assumed to be computationally indistinguishable from a uniform random string of length ℓ. This is the design goal of practical stream ciphers with IV. See [2] for further justification on this point.

Parsing. Let the length of the plaintext or the ciphertext be ℓ bits. Write $\ell = (m-1)n + r$, where $1 \leq r \leq n$. There are a total of m blocks $X_1, \ldots, X_m$, with $X_1, \ldots, X_{m-1}$ being full blocks and the length of the last block is r which is a possible partial block. *We require $\ell > 2n$ so that $m \geq 3$.*

SCTES. The details of the encryption and the decryption algorithms for the tweakable enciphering scheme are shown in Table 1. The required sub-routine is shown in Table 2. We denote the new construction by SCTES.

There are two invocations to SC_K within the Feistel structure. For each of these two invocations, the input IV to SC_K is an n-bit string. Formally, the output is an L-bit string. For the first invocation, the first $\ell - n$ bits are used and for the second invocation, only the first n bits are used. The first n bits of the output of the first stream cipher call is used within the Feistel structure, whereas, the rest $n - 2\ell$ bits are returned and used for encrypting the rest of the message. The key for all the three invocations is the same key K.

There are 4 invocations of the AXU hash function h. Two of these are in the Feistel structure and the other two are outside. The same hash key is used for the two hash calls outside the Feistel structure, while the two hash keys used inside the Feistel structure are independent of each other and also the outside hash key. The reason for using independent keys is that otherwise certain probability calculations do not go through.

Informally, h_τ ensures that with high probability the inputs to the Feistel structure are distinct. The Feistel structure itself realizes an SPRP. In the previous version of this document, the input and the output of the Feistel structure were XORed to form the input to a third invocation of SC_K. The output of this call was used to encrypt the rest of the message. We later realised that this is

Table 1. Encryption and decryption algorithms. The stream cipher key is K and the hash key is (τ, τ_1, τ_2). Let $\mathbf{K} = (K, \tau, \tau_1, \tau_2)$.

Algorithm $\mathsf{Encrypt}^T_{\mathbf{K}}(P_1, \ldots, P_m)$	**Algorithm** $\mathsf{Decrypt}^T_{\mathbf{K}}(P_1, \ldots, P_m)$
1. $M_m = \mathsf{pad}_{n-r}(P_m)$;	1. $U_m = \mathsf{pad}_{n-r}(C_m)$;
2. $(A_1, A_2) = \Upsilon_\tau(P_1, \ldots, P_{m-1}, M_m, T, \mathsf{bin}_n(\ell))$;	2. $(B_1, B_2) = \Upsilon_\tau(C_1, \ldots, C_{m-1}, U_m, T, \mathsf{bin}_n(\ell))$;
3. $(B_1, B_2, Z) = \mathsf{Feistel}_{K,\tau_1,\tau_2}(A_1, A_2, \ell - 2n)$;	3. $(A_1, A_2, Z) = \mathsf{Feistel}^{-1}_{K,\tau_1,\tau_2}(B_1, B_2, \ell - 2n)$;
4. $(C_3, \ldots, C_{m-1}, C_m)$;	4. $(P_3, \ldots, P_{m-1}, P_m)$;
$= (P_3, \ldots, P_{m-1}, P_m) \oplus Z$;	$= (C_3, \ldots, C_{m-1}, C_m) \oplus Z$;
5. $U_m = \mathsf{pad}_{n-r}(C_m)$;	5. $M_m = \mathsf{pad}_{n-r}(P_m)$;
6. $(C_1, C_2) = \Upsilon_\tau(B_1, B_2, C_3, \ldots, C_{m-1}, U_m, T, \mathsf{bin}_n(\ell))$;	6. $(P_1, P_2) = \Upsilon_\tau(A_1, A_2, P_3, \ldots, P_{m-1}, M_m, T, \mathsf{bin}_n(\ell))$;
return $(C_1, \ldots, C_m)$.	return $(P_1, \ldots, P_m)$.

Table 2. A four-round Feistel construction. (A similar construction was used in [18].) Different from a typical Feistel construction, this definition takes in a positive integer i as a third input and returns a third output Z of length equal to i.

$\mathsf{Feistel}_{K,\tau_1,\tau_2}(A_1, A_2, i)$	$\mathsf{Feistel}^{-1}_{K,\tau_1,\tau_2}(B_1, B_2, i)$
1. $H_1 = h_{\tau_1}(A_1)$;	1. $H_2 = h_{\tau_1}(B_2)$;
2. $F_1 = H_1 \oplus A_2$;	2. $F_2 = B_1 \oplus H_2$;
3. $(G_1, Z) = \mathrm{SC}_K(F_1)$;	3. $F_1 = B_2 \oplus \mathrm{SC}_K(F_2)$;
4. $F_2 = A_1 \oplus G_1$;	4. $(G_1, Z) = \mathrm{SC}_K(F_1)$;
5. $B_2 = F_1 \oplus \mathrm{SC}_K(F_2)$;	4. $A_1 = F_2 \oplus G_1$;
6. $H_2 = h_{\tau_2}(B_2)$;	5. $H_1 = h_{\tau_2}(A_1)$;
7. $B_1 = H_2 \oplus F_2$;	6. $A_2 = H_1 \oplus F_1$;
return (B_1, B_2, Z).	return (A_1, A_2, Z).

unnecessary. It is sufficient to take a suitable output from one of the two calls to SC made within the Feistel structure and use it to encrypt the rest of the message. This reduces the number of SC calls from three to two.

Note that the invertibility of the Feistel structure does not depend on the nature of SC_K. This property has also been used in [18] to construct a TES which does not require the decryption function of a block cipher. In fact, the construction in [18] can be considered to be a motivation for the current construction. A lot of the details, however, are different necessitating a separate security proof.

The Feistel construction uses two hashing keys τ_1 and τ_2. In the first version of this paper and also in the stream cipher based construction in [18], only one hash key is used in the Feistel layer, i.e., $\tau_1 = \tau_2 = \tau'$ and both the hash calls in the Feistel network uses τ' as the key. Unfortunately, this causes a problem. Denote the single hash key version of the Feistel by $\mathsf{Feistel}_{K,\tau'}$. We then have $\mathsf{Feistel}_{K,\tau'}(X_1, X_2) = \mathsf{Feistel}^{-1}_{K,\tau'}(X_2, X_1)$. As a consequence, the encryption of (X_1, X_2, X_3) will be equal to the decryption of (X_2, X_1, X_3) which will imply that the construction does not satisfy security as a strong pseudo-random permutation. Using separate keys for the two hashing layers and requiring the third condition on the hash function solves this problem.

This issue has also appeared in previous works. Iwata and Kurosawa [9] insist upon the hash function satisfying a property similar to the third condition that we use here. Even the block cipher based construction in [18] does not have any problem even though the same key is used for both the hashes in the Feistel network. This is because in that construction, the inputs and the outputs of the Feistel layer are masked with independent strings. That technique can also be applied in the current context and then the same hash key can be used for both the hashings within the Feistel structure.

2.1 Efficiency

The key for SCTES consists of the key for the stream cipher and the keys τ, τ_1, τ_2 for the hash functions. For polynomial based hashing or hashing using the BRW polynomials, τ, τ_1 and τ_2 are n-bit blocks.

Let $[H_m]$ denote the time for hashing m blocks using h; [ISC] the time for initializing the stream cipher SC; and $[\mathrm{SC}_i]$ the time for generating i bits using SC *after* it has been initialized. The time complexity of SCTES for encrypting an m-block message is $2([H_{m-2}]+[H_1])+2[\mathrm{ISC}]+[\mathrm{SC}_\ell]$. The derivation of this cost is explained below.

For some stream ciphers, the initialization time [ISC] can be quite high. This is the reason for separately accounting for this. The two initializations are incurred due to the application of the two SC_K calls in the Feistel structure. The first call generates a string of length $\ell - n$, while the second call generates a string of length n. The total time for this generation is accounted for by the term $[\mathrm{SC}_\ell]$. The two calls to h_τ requires time $2[H_{m-2}]$. The two calls to h within the Feistel structure take $[H_1]$ time each. Thus, the total time for hashing is $2([H_{m-2}]+[H_1])$.

There is a restriction that the message length must be greater than $2n$ bits. But, there is no upper bound on the length and varying length messages can be handled. For disk encryption and other practical applications, the restriction of more than $2n$ bits on the message length is not a concern.

Relation to block cipher based TES. For an n-bit block cipher based TES, we must have n to be at least 128 (and in certain cases possibly 192 or 256). This implies that all finite field arithmetic will be over $GF(2^{128})$. On the other hand, for use with a stream cipher, n is the size of the IV and can be 80 bits especially for some hardware oriented stream ciphers. With $n = 80$, all finite field multiplications will be over $GF(2^{80})$. A hardware implementation will require to implement an 80-bit mulitplier which is smaller than a 128-bit multiplier. Overall, this will reduce the size of the hardware.

The other issue is the comparison between the n block cipher calls and the time for the three initializations of SC and the generation of the keystream from the third initialization. This comparison depends to a large extent on the actual primitives (block versus stream) under consideration as well as the particular platform been used. On the whole, for a proper choice of SC, we do expect the corresponding TES to be at least as fast as an AES based TES.

2.2 Disk Encryption

A disk is encrypted sector-wise and the sector address is the tweak. The length of each sector is fixed (typical value is 512 bytes, which correspond to 32 128-bit blocks). It has been mentioned in [7] that the hardware for disk encryption resides just above the disk controller. For such a hardware implementation, it might be worthwhile to use the hashing scheme from [15]. This scheme does not require any multiplications and being parallelizable is quite efficient to implement in hardware using registers and XOR gates. The downside is that this scheme requires a key which is as long as the message. For disk encryption, this means that the key τ used by h is also required to be 512 bytes, whereas the key (τ_1, τ_2) used by h in the Feistel structure is $16 \times 2 = 32$ bytes (if $n = 128$). Note, however, that the same 512 byte key will be used for all the sectors and it is not the case that each sector requires a different key.

The maintenance of a 512-byte key can be a problem. A way out is to generate this key using the stream cipher itself. Let κ be an n-bit string and then generate τ as $\tau = \mathrm{SC}_K(\kappa)$. This τ is

used for the actual hashing. In this case, κ is now the secret key from which the actual hashing key is derived. The time for generating τ from κ is [ISC]+[$\mathsf{SC}_{2^{12}}$]. In practice, τ will be generated once during a read/write session and hence, the time for generating τ will amortized over the number of sectors which are processed in one session. Effectively, the effect of the time for generating τ from κ on the overall time will be negligible. Also, the effect of this strategy on the security bound is negligible.

3 Security

The basic encryption primitive that is used in the construction of SCTES is a stream cipher with IV. This is defined to be a family of functions $\{\mathsf{SC}_K\}_{K\in\mathcal{K}}$, where $\mathsf{SC}_K : \{0,1\}^n \to \{0,1\}^L$. The proper formal model for this primitive is a PRF [2]. This means that a computationally bounded adversary is unable to distinguish the output of SC_K for a uniform random K from the output of a uniform random function $\rho : \{0,1\}^n \to \{0,1\}^L$.

The security analysis of SCTES that we perform in the paper is information theoretic with SC_K being replaced by a uniform random function ρ. Hence, without loss of generality, the adversary $\mathcal{A}$ can be considered to be a deterministic algorithm. Passing from information theoretic security to computational security for SCTES is based on the computational security of SC_K. This task is quite standard and follows along lines similar to that taken in moving from information theoretic to computational security for other constructions. See [7, 5] for details.

The security model for a tweakable enciphering scheme (which can handle inputs of length greater than $2n$ bits) is described below. An adversary $\mathcal{A}$ interacts with the encryption and the decryption oracles of the tweakable enciphering scheme and finally outputs either 0 or 1. Oracles are written as superscripts. The encryption oracle $\mathbf{\Pi}$ takes as input (T, P), where T is an n-bit string and P is a string of length greater than $2n$ and returns C which is of length equal to that of P. Similarly, the decryption oracle $\mathbf{\Pi}^{-1}$ takes as input (T, C) and returns P.

There are some "natural" query restrictions. The first is that the adversary may not repeat a query. Secondly, if the adversary receives C as a response to an encrypt query (T, P), then it is not allowed to make the decrypt query (T, C); similarly, if the adversary receives P as a response to a decrypt query (T, C), then it is not allowed to make the encrypt query (T, P). The disallowed queries are without of loss of generality, since, the adversary already knows the answers to such queries.

The notation $\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1$ denotes the event that the adversary $\mathcal{A}$, interacts with the oracles $\mathbf{\Pi}$ and $\mathbf{\Pi}^{-1}$, and finally outputs the bit 1. We consider an adversary's advantage in distinguishing a tweakable enciphering scheme from an oracle which simply returns random bit strings. This advantage is defined in the following manner.

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathbf{\Pi}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(.,.),\$(.,.)} \Rightarrow 1\right]$$

where $\$(., P)$ returns random bits of length $|P|$. See [7, 6] for other definitions of advantages and the relation to the above definition of advantage.

The query complexity σ_n of an adversary is defined to be the total number of n-bit blocks it provides in all its encryption and decryption queries. This includes the plaintext and ciphertext blocks as well as the n-bit tweak. By $\mathbf{Adv}(q, \sigma_n)$ (with suitable sub and super-scripts) we denote the maximum advantage of any adversary which makes a total of q queries and has query complexity σ_n.

Theorem 1. *Fix n and σ_n to be positive integers. Suppose that an adversary uses a total of σ_n blocks in all its queries, where each block is an n-bit string.*

1. *If $\epsilon_m \leq cm/2^n$ for some constant c, then*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{SCTES}}(\sigma_n) \leq \frac{7q^2 + 4cq\sigma_n}{2^n}. \tag{2}$$

2. *If $\epsilon_m = 1/2^n$ for all m, then*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathsf{SCTES}}(\sigma_n) \leq \frac{8q^2}{2^n}. \tag{3}$$

For the usual polynomial based instantiation of h, the constant c in the above statement is 1 while for BRW based hashing the constant c is 2. In both cases, we get a $q\sigma_n/2^n$ type bound which is significantly better than what has been previously obtained for other tweakable enciphering schemes. If we use an AXU function for which $\epsilon_m = 1/2^n$ (as given in [15]), then the bound improves to a constant multiple of $q^2/2^n$. A few words will be in order to place these bounds in the proper perspective.

Informally, the reason why we obtain such improved bounds is that we work with a stream cipher where we can assume the "long" output to be indistinguishable from a uniform random string. For block cipher based constructions, for any invocation we can only assume that we obtain a uniform random string of length n bits. Thus, for long strings, the bound invariably is of the type $\sigma_n^2/2^n$. This may seem to suggest that using a stream cipher gives a better bound. But, the correct view is that the improvement in the bound on information theoretic security will probably be compensated by the greater computational advantage of an adversary in attacking the PRF-property of the stream cipher with IV. So, on the whole, we would say that the computational bounds for both block and stream cipher based construction will be similar.

The other point is the difference in the bound when ϵ_m is $1/2^n$ versus when it is bounded above by $cm/2^n$. In this case, the bound with $\epsilon_m = 1/2^n$ is actually better. But, the downside is that for such hash functions the key is equal to the length of the message. In the context of disk encryption, the way to tackle such long key length has been discussed in Section 2.2. This method requires one invocation of the stream cipher with IV and has a miniscule effect on the security which is accounted for in the constant 15 given in the theorem statement.

3.1 Proof of Theorem 1

We need to consider collisions among internal variables and so we define the internal variables.

Internal variables. There are two invocations of ρ in the Feistel structure and one invocation of ρ for encryption (or decryption) from the third block onwards. We will call these the first and the second invocation in the order they are performed during encryption. Denote the input to the first invocation by F_1 and the output by (G_1, Z); similarly, denote the input to the second invocation by F_2 and the output by G_2. The quantities F_1, F_2 and G_1, G_2, Z can be expressed in terms of the plaintext and ciphertext blocks.

$$\begin{aligned}
F_1 &= h_{\tau_1}(A_1) \oplus A_2; & F_2 &= B_1 \oplus h_{\tau_2}(B_2);\\
G_1 &= A_1 \oplus F_2 = A_1 \oplus B_1 \oplus h_{\tau_2}(B_2); & G_2 &= B_2 \oplus F_1 = B_2 \oplus A_2 \oplus h_{\tau_1}(A_1);\\
Z &= (P_3, \ldots, P_{m-1}, P_m) \oplus (C_3, \ldots, C_{m-1}, C_m);
\end{aligned}$$

where $M_m = \mathsf{pad}_{n-r}(P_m)$, $U_m = \mathsf{pad}_{n-r}(C_m)$ and

$$A_1 = P_1 \oplus h_\tau(P_3, \ldots, P_{m-1}, M_m, T, \mathsf{bin}_n(\ell)); \;\; A_2 = P_2 \oplus h_\tau(P_3, \ldots, P_{m-1}, M_m, T, \mathsf{bin}_n(\ell));$$
$$B_1 = C_1 \oplus h_\tau(C_3, \ldots, C_{m-1}, U_m, T, \mathsf{bin}_n(\ell)); \;\; B_2 = C_2 \oplus h_\tau(C_3, \ldots, C_{m-1}, U_m, T, \mathsf{bin}_n(\ell)).$$

Notation. The adversary makes a total of q queries of possibly different lengths. We use the superscript $^{(s)}$ to denote quantities corresponding to the s-th query. For example, the length is $\ell^{(s)}$; the number of blocks is $m^{(s)}$; the tweak is $T^{(s)}$; the plaintext blocks are $P_1^{(s)}, \ldots, P_{m^{(s)}}^{(s)}$ and the ciphertext blocks are $C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)}$. Similarly, we denote the internal variables. A query can be either an encryption or a decryption query. The variable $\mathsf{ty}^{(s)}$ denotes the type of the query, i.e., if the query is an encryption query, then $\mathsf{ty}^{(s)} = \mathsf{enc}$, and if the query is a decryption query, then $\mathsf{ty}^{(s)} = \mathsf{dec}$.

The oracles $\mathbf{\Pi}$ and $\mathbf{\Pi}^{-1}$ are built using the uniform random function ρ. Suppose all queries made by the adversary $\mathcal{A}$ are answered in the manner shown in Table 3. For both encrypt and

Table 3. Response to queries from an adversary.

$\mathsf{ty}^{(s)} = \mathsf{enc}$
1. choose $C_1^{(s)}, C_2^{(s)}$ to be independent and uniform random n-bit strings;
2. choose $Z^{(s)}$ to be an independent and uniform random $((m^{(s)}-2)n)$-bit string;
3. set $C_3^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, D_{m^{(s)}}^{(s)}$ to be $(P_3, \ldots, P_{m-1}, P_m \| 0^{n-r^{(s)}}) \oplus Z^{(s)}$;
4. set $C_{m^{(s)}}^{(s)} = \mathsf{First}_{r^{(s)}}(D_{m^{(s)}}^{(s)})$;
5. return $C_1^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, C_{m^{(s)}}^{(s)}$ to the adversary.

$\mathsf{ty}^{(s)} = \mathsf{dec}$
1. choose $P_1^{(s)}, P_2^{(s)}$ to be independent and uniform random n-bit strings;
2. choose $Z^{(s)}$ to be independent and uniform random $((m^{(s)}-2)n)$-bit string;
3. set $P_3, \ldots, P_{m^{(s)}-1}^{(s)}, V_{m^{(s)}}^{(s)}$ to be $(C_3^{(s)}, \ldots, C_{m-1}, C_m \| 0^{n-r^{(s)}}) \oplus Z^{(s)}$;
4. set $P_{m^{(s)}}^{(s)} = \mathsf{First}_{r^{(s)}}(V_{m^{(s)}}^{(s)})$;
5. return $P_1^{(s)}, \ldots, P_{m^{(s)}-1}^{(s)}, P_{m^{(s)}}^{(s)}$ to the adversary.

decrypt queries, the adversary obtains independent and uniform random strings. Then clearly $\mathcal{A}$ cannot distinguish between $\mathbf{\Pi}, \mathbf{\Pi}^{-1}$ and $\$(\cdot,\cdot), \$(\cdot,\cdot)$. But, answering queries in this manner may not always be consistent with the fact that ρ is a uniform random permutation.

Let $\mathbf{E}$ be the event that the random variables $F_1^{(1)}, F_2^{(2)}, \ldots, F_1^{(q)}, F_2^{(q)}$ take distinct values. Conditioned on the event $\mathbf{E}$, the uniform random function ρ is applied to "new" and distinct values. Consequently, the outputs are independent and uniformly distributed. As a result, using the definition of Z, if $\mathsf{ty}^{(s)} = \mathsf{enc}$, then $C_3^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, C_{m^{(s)}}^{(s)}$ is independent of the other random variables and is distributed uniformly over $\{0,1\}^{\ell^{(s)}-2n}$; if $\mathsf{ty}^{(s)} = \mathsf{dec}$, then $P_3^{(s)}, \ldots, P_{m^{(s)}-1}^{(s)}, P_{m^{(s)}}^{(s)}$ is independent of the other random variables and is distributed uniformly over $\{0,1\}^{\ell^{(s)}-2n}$.

Let $\mathcal{D}$ be the set of random variables $\{F_1^{(1)}, F_2^{(1)}, \ldots, F_1^{(q)}, F_2^{(q)}\}$. Let Coll be the event that two random variables in $\mathcal{D}$ take the same value. Note that $\overline{\mathsf{Coll}}$ is equivalent to the event $\mathbf{E}$. For any adversary $\mathcal{A}$, we clearly have

$$\Pr[\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1 | \overline{\mathsf{Coll}}] = \Pr[\mathcal{A}^{\$,\$} \Rightarrow 1].$$

In other words, if Coll does not occur, then in the real game, i.e. during the interaction with $\mathbf{\Pi}, \mathbf{\Pi}^{-1}$, the adversary gets independent and uniform random strings as responses which is the same as it would get while interacting with oracles that return independent and uniform random strings.

$$\begin{aligned}\Pr[\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1] &= \Pr[(\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1) \wedge (\mathsf{Coll} \vee \overline{\mathsf{Coll}})] \\ &= \Pr[(\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1)|\mathsf{Coll}]\Pr[\mathsf{Coll}] + \Pr[(\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1)|\overline{\mathsf{Coll}}]\Pr[\overline{\mathsf{Coll}}] \\ &\leq \Pr[\mathsf{Coll}] + \Pr[\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1|\overline{\mathsf{Coll}}] \\ &= \Pr[\mathsf{Coll}] + \Pr[\mathcal{A}^{\$,\$} \Rightarrow 1].\end{aligned}$$

This gives $\mathsf{Adv}_{\mathbf{\Pi}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{\Pi},\mathbf{\Pi}^{-1}} \Rightarrow 1] - \Pr[\mathcal{A}^{\$,\$} \Rightarrow 1] \leq \Pr[\mathsf{Coll}]$. We next obtain an upper bound for $\Pr[\mathsf{Coll}]$ which would then complete the proof.

Collision analysis. Below we prove some results on the probability of certain kinds of collisions.

Claim. Let τ_1, τ_2 be chosen independently and uniformly at random from $\mathbb{F}$. For $1 \leq s, t \leq q$, $\Pr_{\tau_1,\tau_2}[F_1^{(s)} = F_2^{(t)}] = 1/2^{n-1}$.

Proof of claim. The s-th and the t-th queries can be either encryption or decryption queries. We identify two cases. In Case A, the s-th query is an encryption query and the t-th query is a decryption query while Case B consists of all the other three situations. The arguments for the two cases are given separately below.
Case A: $A_1^{(s)} = P_1 \oplus h_\tau(P_3, \ldots, P_{m-1}, M_m, T, \mathsf{bin}_n(\ell))$. Since $\ell > 2n$, $\mathsf{bin}_n(\ell)$ is a non-zero string and so h_τ is applied on a non-zero input. By the uniformity property of h, the probability that $A_1^{(s)} = 0$ is $1/2^n$.

$$\begin{aligned}\Pr[F_1^{(s)} = F_2^{(t)}] &= \Pr[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}] \\ &= \Pr[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}|A_1^{(s)} = 0]\Pr[A_1^{(s)} = 0] \\ &\quad + \Pr[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}|A_1^{(s)} \neq 0]\Pr[A_1^{(s)} \neq 0] \\ &\leq \Pr[A_1^{(s)} = 0] + \Pr[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}|A_1^{(s)} \neq 0] \\ &\leq \frac{1}{2^n} + \Pr[h_{\tau_1}(A_1^{(s)}) \oplus h_{\tau_2}(B_2^{(t)}) = A_2^{(s)} \oplus B_1^{(t)}|A_1^{(s)} \neq 0] \\ &\leq \frac{1}{2^n} + \frac{1}{2^n} = \frac{1}{2^{n-1}}.\end{aligned}$$

The last inequality follows from the third property of h.
Case B: $A_1^{(s)} = P_1^{(s)} \oplus \mathsf{rest}^{(s)}$ and $B_2^{(t)} = C_2^{(t)} \oplus \mathsf{rest}_1^{(t)}$, where $\mathsf{rest}^{(s)}$ and $\mathsf{rest}_1^{(t)}$ are independent of both $P_1^{(s)}$ and $C_2^{(t)}$. So, $A_1 = B_2$ holds if and only if $P_1^{(s)} \oplus C_2^{(t)} = \mathsf{rest}^{(s)} \oplus \mathsf{rest}_1^{(t)}$. If the t-th query is an encrypt query, then $C_2^{(t)}$ is an independent and uniform random string, or, if the s-th query is a decrypt query then $P_1^{(s)}$ is an independent and uniform random string. So, irrespective of whether s equals t or not, one of $P_1^{(s)}$ and $C_2^{(t)}$ is a uniform random string which is independent of the other as also independent of $\mathsf{rest}^{(s)} \oplus \mathsf{rest}_1^{(t)}$. This shows $\Pr[A_1^{(s)} = B_2^{(t)}] \leq 1/2^n$.

$$\begin{aligned}\Pr[F_1^{(s)} = F_2^{(t)}] &= \Pr_{\tau_1,\tau_2}[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}] \\ &= \Pr_{\tau_1,\tau_2}\left[\left(h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}\right) \wedge ((A_1^{(s)} = B_2^{(t)}) \vee (A_1^{(s)} \neq B_2^{(t)}))\right]\end{aligned}$$

$$
\begin{aligned}
&= \Pr_{\tau_1,\tau_2}\left[\left(h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}\right) | (A_1^{(s)} = B_2^{(t)})\right] \Pr_{\beta_1}[A_1^{(s)} = B_2^{(t)}] \\
&\quad + \Pr_{\tau_1,\tau_2}\left[\left(h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}\right) | (A_1^{(s)} \neq B_2^{(t)})\right] \Pr[A_1^{(s)} \neq B_2^{(t)}] \\
&\leq \Pr[A_1^{(s)} = B_2^{(t)}] + \Pr_{\tau_1,\tau_2}\left[\left(h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_2}(B_2^{(t)}) \oplus B_1^{(t)}\right) | (A_1^{(s)} \neq B_2^{(t)}))\right] \\
&\leq \frac{2}{2^n}.
\end{aligned}
$$

The last inequality follows from the third property of h. □

Claim. Suppose that τ, τ_1 and τ_2 are chosen independently and uniformly at random from $\mathbb{F}$; $1 \leq s < t \leq q$ and suppose without loss of generality that $\ell^{(s)} \leq \ell^{(t)}$.

1. If $(P_1^{(s)}, \ldots, P_{m-1}^{(s)}, M_m^{(s)}, T^{(s)}, \mathsf{bin}_n(\ell^{(s)})) \neq (P_1^{(t)}, \ldots, P_{m-1}^{(t)}, M_m^{(t)}, T^{(t)}, \mathsf{bin}_n(\ell^{(t)}))$, then $\Pr_{\tau,\tau_1}[F_1^{(s)} = F_1^{(t)}] \leq \epsilon_{m^{(t)}} + 1/2^n$.
2. If $(C_1^{(s)}, \ldots, C_{m-1}^{(s)}, U_m^{(s)}, T^{(s)}, \mathsf{bin}_n(\ell^{(s)})) \neq (C_1^{(t)}, \ldots, C_{m-1}^{(t)}, U_m^{(t)}, T^{(t)}, \mathsf{bin}_n(\ell^{(t)}))$, then $\Pr_{\tau,\tau_2}[F_2^{(s)} = F_2^{(t)}] \leq \epsilon_{m^{(t)}} + 1/2^n$.

Proof of claim. We prove (1), the proof of (2) being similar. Note that $F_1^{(s)} = h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)}$ and $F_1^{(t)} = h_{\tau_1}(A_1^{(t)}) \oplus A_2^{(t)}$.

If at least one of $\mathsf{ty}^{(s)}$ or $\mathsf{ty}^{(t)}$ is equal to dec, then one of $P_2^{(s)}$ and $P_2^{(t)}$ is a uniform random string which is independent of all other random variables. So, in this case, it follows from the definition of F_1 that the probability that $F_1^{(s)}$ equals $F_1^{(t)}$ is $1/2^n$.

Now suppose that both $\mathsf{ty}^{(s)}$ and $\mathsf{ty}^{(t)}$ are equal to enc.

Case $(P_3^{(s)}, \ldots, P_{m-1}^{(s)}, M_m^{(s)}, T^{(s)}, \mathsf{bin}_n(\ell^{(s)})) = (P_3^{(t)}, \ldots, P_{m-1}^{(t)}, M_m^{(t)}, T^{(t)}, \mathsf{bin}_n(\ell^{(t)}))$. Then necessarily $(P_1^{(s)}, P_2^{(s)}) \neq (P_1^{(t)}, P_2^{(t)})$ (as otherwise the two queries would be same). If $P_1^{(s)} = P_1^{(t)}$ (and so, $P_2^{(s)} \neq P_2^{(t)}$), then $A_1^{(s)} = A_1^{(t)}$; $P_2^{(s)} \neq P_2^{(t)}$ implies $A_2^{(s)} \neq A_2^{(t)}$ whereby we have $F_1^{(s)} \neq F_1^{(t)}$. If $P_2^{(s)} = P_2^{(t)}$ (and so, $P_1^{(s)} \neq P_1^{(t)}$), then $A_2^{(s)} = A_2^{(t)}$; $P_1^{(s)} \neq P_1^{(t)}$ implies $A_1^{(s)} \neq A_1^{(t)}$ whereby $F_1^{(s)} = F_1^{(t)}$ implies $h_{\tau_1}(A_1^{(s)}) = h_{\tau_1}(A_1^{(t)})$. Since $A_1^{(s)} \neq A_1^{(t)}$, by the XOR-universality of h, the last condition holds with probability $1/2^n$.

Case $(P_3^{(s)}, \ldots, P_{m-1}^{(s)}, M_m^{(s)}, T^{(s)}, \mathsf{bin}_n(\ell^{(s)})) \neq (P_3^{(t)}, \ldots, P_{m-1}^{(t)}, M_m^{(t)}, T^{(t)}, \mathsf{bin}_n(\ell^{(t)}))$. By the XOR-universality of h, $\Pr_\tau[A_1^{(s)} = A_1^{(t)}] \leq \epsilon_m$. Also, note that $A_1^{(s)}, A_2^{(s)}, A_1^{(t)}$ and $A_2^{(t)}$ are independent of τ_1.

$$
\begin{aligned}
\Pr_{\tau,\tau_1}[F_1^{(s)} = F_1^{(t)}] &= \Pr_{\tau,\tau_1,\tau_2}[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_1}(A_1^{(t)}) \oplus A_2^{(t)}] \\
&\leq \Pr_{\tau}[A_1^{(s)} = A_1^{(t)}] + \Pr_{\tau_1}[h_{\tau_1}(A_1^{(s)}) \oplus A_2^{(s)} = h_{\tau_1}(A_1^{(t)}) \oplus A_2^{(t)} | (A_1^{(s)} \neq A_1^{(t)})] \\
&\leq \epsilon_m + \frac{1}{2^n}.
\end{aligned}
$$

The last relation holds due to the XOR-universality of h. □

Claim. 1. If $\epsilon_m \leq cm/2^n$ for some constant c, then $\Pr[\mathsf{Coll}] \leq 7q^2/2^n + 4cq\sigma/2^n$.
2. If $\epsilon_m = 1/2^n$ for all m, then $\Pr[\mathsf{Coll}] \leq 8q^2/2^n$.

Proof of claim. $\mathcal{D}$ contains $2q$ random variables. Apart from the $q(q-1)$ pairs of random variables of the form $(F_1^{(s)}, F_1^{(t)})$ and $(F_2^{(s)}, F_2^{(t)})$, the probability that any other pair of random variables take the same value is at most $1/2^{n-1}$. The probability that a pair of the form $(F_j^{(s)}, F_j^{(t)})$, $j = 1, 2$, $s \neq t$ take the same value is $\epsilon_{m^{(t)}} + 1/2^n$. If $\epsilon_m = 1/2^n$ for all m, then the result easily holds.

Suppose that $\epsilon_m \leq cm/2^n$. Let $m^{(1)} \geq m^{(2)} \geq \cdots \geq m^{(q)}$. Then the probability that any pair of the form $(F_j^{(s)}, F_j^{(t)})$, $j = 1, 2$, $s \neq t$, take the same value is at most

$$\begin{aligned} 2 \times \frac{2c}{2^n} \times \sum_{s=0}^{q-1} (q-s) m_{s+1} &\leq 2 \times \frac{2qc}{2^n} \times \sum_{s=0}^{q-1} m_{s+1} \\ &\leq \frac{4cq\sigma}{2^n}. \end{aligned}$$

Now the statement of Theorem 1 follows. □

Acknowledgement

The previous version of this document used three IV set-ups. Mridul Nandi suggested a different construction which used two IV set-ups. Following up on this suggestion, we realised that one of the IV set-ups in our construction was redundant which lead to the current version.

References

1. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.
2. Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.
3. Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.
4. Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006. full version available at `http://eprint.iacr.org/2007/028`.
5. Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008.
6. Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.
7. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.
8. Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.
9. Tetsu Iwata and Kaoru Kurosawa. How to construct super-pseudorandom permutations with short keys. *IEICE Transactions*, 90-A(1):2–13, 2007.
10. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
11. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
12. David A. McGrew and Scott R. Fluhrer. The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. `http://eprint.iacr.org/`.
13. Kazuhiko Minematsu and Toshiyasu Matsushima. Tweakable enciphering schemes from hash-sum-expansion. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 252–267. Springer, 2007.
14. Michael O. Rabin and Shmuel Winograd. Fast evaluation of polynomials by rational preparation. *Communications on Pure and Applied Mathematics*, 25:433–458, 1972.
15. Palash Sarkar. A new multi-linear hash family. *Designs, Codes, and Cryptography.* to appear.

16. Palash Sarkar. A general mixing strategy for the ECB-Mix-ECB mode of operation. *Inf. Process. Lett.*, 109(2):121–123, 2008.
17. Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Transactions on Information Theory*, 55(10):4749–4759, 2009.
18. Palash Sarkar. Tweakable enciphering schemes using only the encryption function of a block cipher. *Inf. Process. Lett.*, 111(19):945–955, 2011.
19. Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.