HCH: A New Tweakable Enciphering Scheme Using the Hash-Counter-Hash Approach

Debrup Chakraborty[1] and Palash Sarkar[2]

[1] Computer Science Department
CINVESTAV-IPN
Mexico, D.F., 07360, Mexico
email: debrup@cs.cinvestav.mx

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

Abstract. The notion of tweakable block ciphers was formally introduced by Liskov-Rivest-Wagner at Crypto 2002. The extension and the first construction, called CMC, of this notion to tweakable enciphering schemes which can handle variable length messages was given by Halevi-Rogaway at Crypto 2003. In this paper, we present HCH, which is a new construction of such a scheme. The construction uses two universal hash computations with a counter mode of encryption in-between. This approach was first proposed by McGrew-Viega to build a scheme called XCB and later used by Wang-Feng-Wu, to obtain a scheme called HCTR. Among the hash-Ctr-hash type constructions, an important advantage of HCH compared to the others is that HCH has a quadratic security bound; XCB does not provide any security bound while HCTR has a cubic security bound. A unique feature of HCH compared to all known tweakable enciphering schemes is that HCH uses a single key, can handle arbitrary length messages and has a quadratic security bound. An important application of a tweakable enciphering scheme is disk encryption. HCH is well suited for this application. We also describe a variant, which can utilize pre-computation and makes one less block cipher call. This compares favourably to other hash-encrypt-hash type constructions; supports better key agility and requires less key material.[3].
Keywords: modes of operations, tweakable encryption, strong pseudorandom permutation, disk encryption.

1 Introduction

A block cipher is one of the basic primitives used in cryptography. Depending upon application goals, there are many uses of a block cipher. A particular method of using a block cipher is called a mode of operation. The literature describes different modes of operations of a block cipher achieving goals such as confidentiality, authentication, authenticated encryption, etcetera. For several years, NIST of USA [1] has been running an open domain process to standardize modes of operations for achieving various functionalities. Currently, there are around twenty different modes of operations proposals for different tasks.

One particular interesting functionality is a tweakable enciphering scheme [6]. (We note that this functionality is currently not covered by NIST's standardization efforts.) This is based on the notion of tweakable block ciphers introduced in [8]. A tweakable enciphering scheme is a length preserving encryption protocol which can encrypt messages of varying lengths. The security goal is to satisfy the notion of the tweakable strong pseudorandom permutation (SPRP). As pointed out in [6], one of the most important applications of a tweakable enciphering scheme is disk encryption.

[3] An abridged version of the paper appears as [2].

Like other modes of operations, a tweakable enciphering scheme is constructed out of a block cipher. A block cipher can encrypt only fixed length strings (say n-bit strings). One of the goals is to be able to extend the domain to handle message of all possible lengths. On the other hand, for applications such as disk encryption, the requirement is to separately encrypt each sector, which is of fixed length and the value of the length is usually a power of two.

For practical applications, efficiency of the construction is very important. In particular, the times for encryption and decryption should be as small as possible. The other efficiency issue is of space, i.e., the size of hardware implementation and the amount of secure storage space required. The amount of secure storage space is determined by the amount of key material and also by the size of pre-computed tables and key schedules which may be required to speed up actual encryption and decryption. From this point of view, it is desirable not to increase the size of the secret key, i.e., to use the secret key of the block cipher as the only secret and nothing else.

The main security goal is to obtain a "quadratic" security bound. In other words, if the adversary uses σ_n blocks (where each block is an n-bit string) in all its queries, then its advantage of distinguishing the output of the tweakable enciphering scheme from random strings should be upper bounded by some constant times $\sigma_n^2/2^n$. This is a natural security goal, since this goal is also expected of the underlying block cipher.

A brief history of known constructions. The first work to present a strong pseudo random permutation is by Naor-Reingold [12]. They did not provide tweakable SPRPs since their work predates this notion.

The first construction of tweakable SPRP was provided by Halevi-Rogaway and was called CMC [6]. Parallelizable constructions called EME [7] and EME* [4] were later presented. These three constructions use two layers of encryption with an intermediate mixing layer. In CMC, the encryption layers are of cipher block chaining (CBC) type, whereas in EME and EME*, the encryption layers are of electronic codebook (ECB) type.

In contrast, the earlier construction of Naor-Reingold consisted of a single encryption layer sandwiched between two *invertible* universal hash computations. Encryption consisted of one layer of ECB. Later constructions of the hash-ECB-hash type are PEP [3] and the more recent construction TET [5].

A different class of constructions consists of two layers of universal hash functions (non invertible) with a counter mode of encryption in between. One advantage of using the counter mode of encryption is that it becomes easy to tackle variable length messages. The first construction of the hash-counter-hash type was XCB [10]. A later construction is HCTR [14]. Another construction which has a structure similar to XCB is ABL [11]. Even though the structure is similar to that of XCB, ABL is significantly slower – it consists of essentially three counter layers and two polynomial hash layers.

In the Naor-Reingold approach, both the universal hash and the ECB layers are invertible. On the other hand, in the counter based approach, the universal hash function is non-invertible and this can be considered to be closer to the Luby-Rackoff [9] approach, where non-invertible pseudo random functions are used to construct an invertible map.

The construction we present is also of the hash-Ctr-hash type. To understand our contribution, it might be helpful to have a brief idea of the previous two constructions – XCB and HCTR. (We do not consider ABL, since it is significantly slower than XCB.) The schematic diagrams for encryption using XCB and HCTR are given in Figures 1 and 2. For the exact details of XCB and HCTR, we refer the reader to the respective papers.

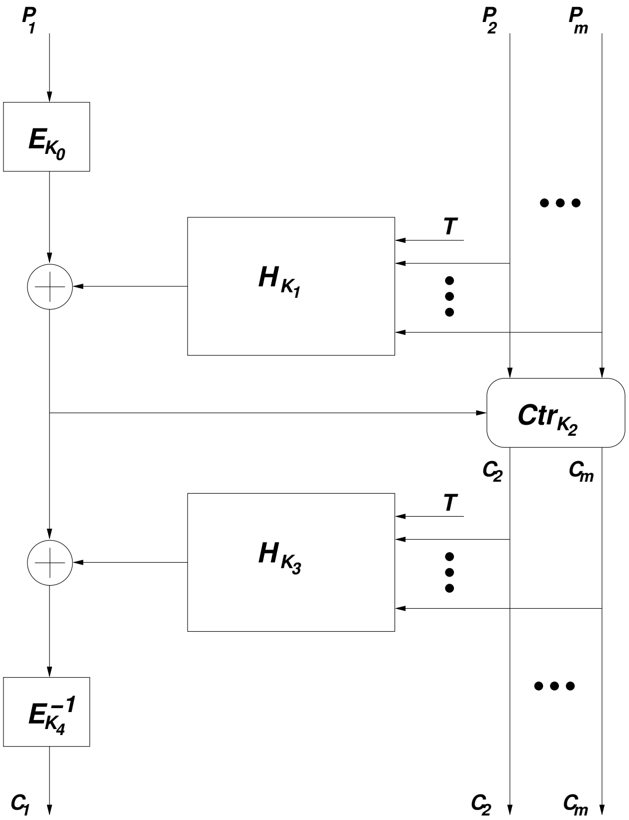

Fig. 1. Encryption using XCB. Here the five keys $K_0, \ldots, K_4$ are derived from a single key K using five block cipher invocations in the following manner: $K_0 = E_K(0^n)$, $K_1 = E_K(0^{n-1}||1)$, $K_2 = E_K(0^{n-2}||1||0)$, $K_3 = E_K(0^{n-2}||1^2)$, $K_4 = E_K(0^{n-3}||1||0^2)$. The keys K_0, K_2 and K_4 are used as block cipher keys whereas, the keys K_1 and K_3 are used as keys for the universal hash function computations.

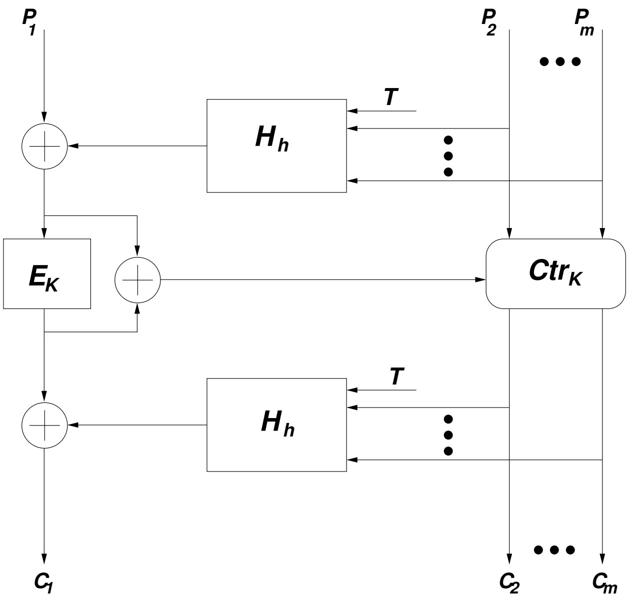

Fig. 2. Encryption using HCTR. Here K is the key for the block cipher $E_K()$ and h is the key for the universal hash function $H_h()$.

XCB: From the single key K for the block cipher, XCB derives five keys $K_0, \ldots, K_4$. The keys K_1 and K_3 are used as keys for the universal hash functions, i.e., the polynomials formed from the data are evaluated at K_1 and K_3. The keys K_0 and K_4 are used for a single encryption and decryption operation respectively of the underlying block cipher. The key K_2 is used as a key to the counter mode of operation. This means that K_2 is also used as a key for the underlying block cipher.

Viewed in this fashion, XCB has one limitation. Since an output of the block cipher is to be used as a key, this means that the length of the key and the block length should be the same. It will not be possible to use XCB when the key length is different from the block length, as for example in Rijndael with 192-bit key and 128-bit block lengths. Viewed differently, one can consider XCB as if it uses five keys $K_0, \ldots, K_4$. Then this problem of key-length block-length mis-match does not arise. On the other hand, a five key protocol is not very attractive which is perhaps why the designers chose to derive the five keys from a single one.

The tweak in XCB can be of arbitrary length. The message is padded with zeros to make it a multiple of the block length. Similarly, the tweak is also padded and appended to the message. Then the lengths of the message and the tweak are appended. The entire string now has a length which is a multiple of the block length. This string is hashed using the universal hash function.

An important issue regarding XCB is the lack of a concrete security bound. A sketch of the security proof is provided but no bound is worked out. The issue of obtaining concrete security bounds is usually considered to be important for various modes of operations.

HCTR: This work is later than XCB and there are a few differences. It uses a single key for the block cipher and a single key for both the layers of the universal hash function. The tweak and length are handled in a manner similar to that of XCB. On the other hand, the first block is handled differently. In particular, the decryption call during XCB encryption is no longer required. HCTR provides a security proof and a concrete security bound. The adversary's advantage is upper bounded by a constant multiple of $\sigma_n^3/2^n$, where σ_n is the number of n-bit blocks provided by the adversary in all its queries. This is higher than the usual quadratic bound which is of the type $\sigma_n^2/2^n$.

1.1 Our Contributions

In this paper, we present HCH, which is a construction of a new tweakable enciphering scheme. HCH is also of the hash-counter-hash type. A schematic diagram is shown in Figure 4.

Compared to XCB and HCTR, we introduce an extra block cipher call before initializing the counter mode. This block cipher call plays an important role in obtaining a quadratic security bound. Without this block cipher call, the counter mode is initialized with the output of a universal hash function. As a result, the inputs of all the block cipher calls as part of the counter mode are minor modifications of the universal hash output. In the collision analysis in HCTR, it is this feature which leads to a cubic security bound. For XCB, this could be a problem, if one wanted to obtain a security bound. On the other hand, in HCH, the extra block cipher call ensures that the counter mode is initialized with a "random" quantity. It is due to this that the collision analysis in HCH leads to a quadratic security bound.

The tweak and the length are also handled differently from both XCB and HCTR. HCH handles n-bit tweaks. The encryption of the tweak (called R) is XOR-ed to the binary expansion of the message length and the quantity is encrypted once more to obtain Q. The key for the universal hash function is taken to be R. This method of handling the tweak and the length is taken from

PEP. As a result of using this method, we do not require a separate key for the universal hash function.

More details on the comparison of HCH to the other modes of operations are given in Section 3. From the viewpoint of efficiency, the encrypt-mix-encrypt constructions (CMC, EME, EME*) use approximately two block cipher calls per message block while the hash-encrypt-hash constructions (XCB, HCTR, HCH and TET) use approximately one block cipher call and two $GF(2^n)$ multiplications per message block. The second approach is faster if one block cipher call takes more time than two $GF(2^n)$ multiplications. Thus, this is actually a comparison between the two approaches.

One desirable goal is to obtain a tweakable enciphering scheme which uses a single key, can encrypt arbitrary length messages and has a quadratic security bound. HCH achieves this combination of properties while none of the other known constructions satisfy all three of these properties (see Table 1).

One drawback of HCH compared to XCB and TET is that it is not possible to use pre-computation to speed up the polynomial hash computation. We note, however, that secure storage for pre-computed tables can be problem for multi-key environments and hardware implementations. Nevertheless, we describe a simple variant of HCH called HCHp for which it is possible to utilize pre-computation. HCHp uses a separate hashing key. Though HCHp uses more key material than HCH, this amount is still lower than the other proposals of the hash-encrypt-hash type constructions.

Disk encryption is an important application of tweakable SPRPs. For this application, the number of blocks in the message is fixed. We describe a variant HCHfp which works for fixed length messages; can utilize pre-computation; and reduces the number of block cipher calls by one. The efficiency (time for encryption and decryption) of this construction is similar to other known hash-encrypt-hash type constructions; on the other hand, HCHfp requires less key material and provides better key agility.

2 Specification of HCH

We construct the tweakable enciphering scheme HCH from a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and call it HCH[E]. The key space of HCH[E] is same as that of the underlying block cipher E and the tweak space is $\mathcal{T} = \{0,1\}^n$. The message space consists of all binary strings of length greater than n.

An n-bit string can be viewed as an element of $GF(2^n)$. We will consider each n-bit string in the specification of HCH as a polynomial over $GF(2)$ of degree less than n, and multiplication will be done modulo a fixed irreducible polynomial $\tau(x)$ of degree n. Thus, if A and B are n-bit strings, then by AB we will mean the n-bit string representing the product $A(x)B(x) \bmod \tau(x)$. Also, the notation xQ denotes the n-bit string representing $xQ(x) \bmod \tau(x)$. The operation $\oplus$ denotes addition over $GF(2^n)$.

For an n-bit string X, by $\mathsf{pad}_t(X)$ we denote the string $X||0^t$ and by $\mathsf{drop}_t(X)$ we denote the prefix of X obtained by dropping the last t bits of X. For $0 \le i \le 2^t - 1$, by $\mathsf{bin}_t(i)$ we denote the t-bit binary representation of the integer i.

Let $R, Q, A_1, \ldots, A_m$ be n-bit strings. We define

$$H_{R,Q}(A_1, \ldots, A_m) = Q \oplus A_1 \oplus A_2R^{m-1} \oplus A_3R^{m-2} \cdots \oplus A_{m-1}R^2 \oplus A_mR. \qquad (1)$$

The above operations are over $GF(2^n)$, i.e., $\oplus$ denotes addition over $GF(2^n)$ and terms of the form A_iR^{m-i+1} denote the product $A_i(x)R^{m-i+1}(x) \bmod \tau(x)$. The final value of $H_{R,Q}(A_1, \ldots, A_m)$ is an element of $GF(2^n)$ given by its n-bit string representation with respect to $\tau(x)$. From the definition of $H_{R,Q}()$, we have the following simple property which is required for proper decryption.

$$\text{If } B_1 = H_{R,Q}(A_1, A_2, \ldots, A_m), \text{ then } A_1 = H_{R,Q}(B_1, A_2, \ldots, A_m). \tag{2}$$

In the conference version [2] of this paper, the function $H_{R,Q}$ was defined as $Q \oplus A_1 \oplus A_2 R \oplus \ldots \oplus A_m R^{m-1}$. This is in the reverse order compared to (1). The present definition of $H_{R,Q}$ is useful in evaluating the hash function in the order of the message blocks using Horner's rule. The ordering of the message blocks do not affect the security proof since with both orderings we obtain polynomials of degree $(m-1)$ and in the proof we are interested in whether R is a root of such a polynomial. We note that XCB and HCTR also use the same ordering as (1), a fact which we had overlooked while writing the conference version.

HCH requires a counter mode of operation. Given an n-bit string S, we define a sequence $S_1, \ldots, S_m$, where each S_i depends on S. Given such a sequence and a key K we define the counter mode as follows.

$$\mathsf{Ctr}_{K,S}(A_1, \ldots, A_m) = (A_1 \oplus E_K(S_1), \ldots, A_m \oplus E_K(S_m)). \tag{3}$$

In XCB, S_i is defined from S_{i-1} in the following manner. The rightmost 32 bits of S_{i-1} are treated as a non-negative integer with the least significant bit on the right; then this value is incremented modulo 2^{32} to obtain S_i. This method limits the number of blocks in the message to at most 2^{32}.

For our proof, we will require the sequence S_i to satisfy two properties. To define these, we introduce a notation. For $i \geq 1$, let $S_i = f_i(S)$, where each f_i is a function from $\{0,1\}^n$ to $\{0,1\}^n$. Then we need S_i to satisfy the following two properties.

$$\left.\begin{array}{rcll} f_i(S) & \neq & f_j(S) & \text{for each } S \text{ and for } i \neq j; \\ \Pr[f_i(S^1) = f_j(S^2)] & = & \frac{1}{2^n} & \text{for random and independent } S^1, S^2. \end{array}\right\} \tag{4}$$

One simple way of defining f_i so as to satisfy (4) is to set $f_i(S) = S \oplus \mathsf{bin}_n(i)$ as has been done for HCTR. Both the methods of XCB and HCTR require the use of an adder. On the other hand, for nonzero S, we can also have $f_i(S) = L_i$, where L_i is the ith state of a maximal length LFSR initialized by S. With this definition, it is not too difficult to prove that the sequence S_i satisfies (4). Using an LFSR might be more efficient than using an adder for hardware implementation. In the following, we will work with S_i keeping in mind the properties in (4). Any efficient method for generating S_is from S such that the S_is satisfy (4) will serve our purpose.

Details on message parsing are as follows.

1. The message length is l bits, where $n < l < 2^n - 1$. We write $l = n(m-1) + r$, with $1 \leq r \leq n$.
2. The message consists of blocks $P_1, \ldots, P_m$,
   with $|P_1| = \cdots = |P_{m-1}| = n$ and $1 \leq |P_m| = r \leq n$.
3. The ciphertext is of the same length as the message, i.e.,
   the ciphertext blocks are $C_1, \ldots, C_m$, with $|C_i| = |P_i|$ for $1 \leq i \leq m$.

The complete encryption and decryption algorithm of HCH is given in Figure 3. A schematic diagram of encryption is given in Figure 4.

2.1 Messages of Length n

We have excluded the case of $l = n$ from the above specification. This is because if $l = n$, then there is only a single block message and the counter part becomes vacuous. As a result, the block cipher call to produce S is no longer required. (The quantity Q is still required.) Thus, this case needs to be tackled separately. The extension is the following.

Fig. 3. Encryption and decryption using HCH. The tweak is T and the key is K.

Algorithm $\mathbf{E}_K^T(P_1, \ldots, P_m)$	**Algorithm** $\mathbf{D}_K^T(C_1, \ldots, C_m)$
1. $R = E_K(T)$; $Q = E_K(R \oplus \mathsf{bin}_n(l))$;	1. $R = E_K(T)$; $Q = E_K(R \oplus \mathsf{bin}_n(l))$;
2. $M_m = \mathsf{pad}_{n-r}(P_m)$;	2. $U_m = \mathsf{pad}_{n-r}(C_m)$;
3. $M_1 = H_{R,Q}(P_1, \ldots, P_{m-1}, M_m)$;	3. $U_1 = H_{R,xQ}(C_1, \ldots, C_{m-1}, U_m)$;
4. $U_1 = E_K(M_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;	4. $M_1 = E_K^{-1}(U_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;
5. $(C_2, \ldots, C_{m-1}, D_m)$	5. $(P_2, \ldots, P_{m-1}, V_m)$
$\quad = \mathsf{Ctr}_{K,S}(P_2, \ldots, P_{m-1}, M_m)$;	$\quad = \mathsf{Ctr}_{K,S}(C_2, \ldots, C_{m-1}, U_m)$;
6. $C_m = \mathsf{drop}_{n-r}(D_m)$; $U_m = \mathsf{pad}_{n-r}(C_m)$;	6. $P_m = \mathsf{drop}_{n-r}(V_m)$; $M_m = \mathsf{pad}_{n-r}(P_m)$;
7. $C_1 = H_{R,xQ}(U_1, C_2, \ldots, C_{m-1}, U_m)$;	7. $P_1 = H_{R,Q}(M_1, P_2, \ldots, P_{m-1}, M_m)$;
8. return $(C_1, \ldots, C_m)$.	8. return $(P_1, \ldots, P_m)$.

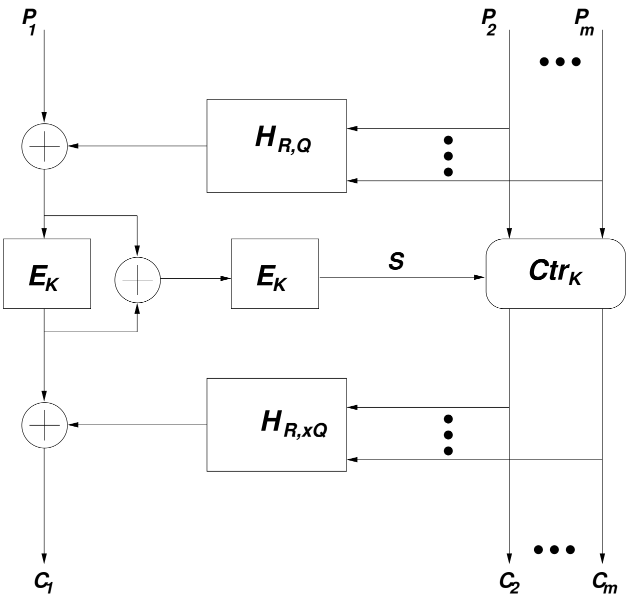

Fig. 4. Encryption using HCH. Here $R = E_K(T)$ and $Q = E_K(R \oplus \mathsf{bin}_n(l))$.

If $l > n$ use HCH to encrypt.
If $l = n$ set $C_1 = xQ \oplus E_K(P_1 \oplus Q)$.

In the case $l = n$, there is a single message block P_1 and a corresponding ciphertext block C_1 defined above. Decryption is simple. This requires a total of 3 block cipher calls (2 to produce Q and one for producing C_1).

The security of the above modification cannot be generically derived from that of HCH. We need to have a separate proof for it. On the other hand, this proof is very similar to that of HCH, with the only difference that we will have to take care of the possibilities of domain and range collisions due to single block adversarial queries. We do not actually present this separate proof. Instead, we present the complete proof for HCH with $l \geq n$, from which a reader can easily obtain a proof for the modified protocol.

The intuition behind this is quite easy. While considering internal collisions, the quantity $P_1 \oplus Q$ will be considered for possible collisions with other elements in the domain of the block cipher, while the quantity $C_1 \oplus xQ$ will have to be considered for possible collisions with other elements in the range of the block cipher. The value of Q depends on both the tweak and the length. Thus, for different length queries, the values of Q will be equal with probability $1/2^n$. This takes care of possible collisions between single block queries and queries with more blocks. Among single block queries, if the tweaks are different, then again the values of the corresponding Qs are different and so the probability of a collision is $1/2^n$. The values of Q will be same only for those blocks which have the same tweak. But in this case the corresponding plaintexts must be different (as otherwise the adversary has provided the same query twice) and hence the probability of a collision is zero.

3 Discussion and Comparison

The differences of HCH to XCB and HCTR were mentioned earlier. All three can handle arbitrary length strings. Regarding security bounds, XCB does not provide any such bound, HCTR provides a cubic bound, whereas HCH has a quadratic security bound. Among the hash-Ctr-hash constructions, this is perhaps the most important feature of HCH compared to the others.

One limitation of XCB is that it might not be possible to use it with a block cipher whose key length is different from the block length (unless one adopts a multi-key version of XCB as suggested earlier). Also, the upper bound on the maximum length of a message is possibly determined by the definition of the counter mode in XCB. HCTR uses one block cipher key plus one key for the universal hash function. In contrast, HCH uses a single block cipher key. The key for the universal hash function is derived from the tweak.

TET is a recent construction which is of the hash-ECB-hash type, where the universal hash function is designed to be bijective. The algorithm can handle arbitrary length tweaks and arbitrary length messages. It uses two block cipher keys – one for the ECB layer and the other for a pseudorandom function to process the tweak and also for other computations. However, even though TET can tackle arbitrary lengths, it is not very efficient in such situations. For each query having m blocks, TET needs to compute $\tau = \mathrm{PRF}_{K_1}(0, i)$ (where $\mathrm{PRF}_{K_1}()$ is a pseudo-random function with key K_1) and $\sigma = 1 \oplus \tau \oplus \cdots \oplus \tau^m$ successively for $i = 0, 1, \ldots$ until a non-zero σ is found. Then, this σ needs to be inverted. (This σ is not to be confused with σ_n, the total number of n-bit blocks provided by the adversary in all its queries.) These computations make TET unattractive for variable length situations.

In Table 1, we present a comparison of HCH with the previous algorithms. We compare to CMC, EME and EME* which are encrypt-mix-encrypt type constructions. XCB and HCTR are also included, since they are of the hash-counter-hash type. We do not include ABL since it is significantly slower than the other proposals. PEP and TET are hash-ECB-hash type constructions with PEP being slower than TET.

EME tackles only strings of very specific lengths. From the table, we see that EME*, XCB, HCTR, HCH and TET are the algorithms which can tackle arbitrary length strings (HCH can be modified to also tackle strings of length n, see Section 2.1.) As discussed earlier, TET is not interesting for encrypting variable length strings. EME* has a quadratic security bound, uses one block cipher key plus two n-bit strings as auxiliary key material. XCB uses a single key (with some restrictions) but does not provide any security bound. HCTR uses two keys and has a cubic security bound. On the other hand, HCH uses a single key and has a quadratic security bound.

The algorithms CMC and EME* are based on the encrypt-mask-encrypt approach, while XCB, HCTR and HCH are based on the hash-encrypt-hash approach. The first approach requires more block cipher calls while the second approach requires more finite field multiplications. The comparison is based on the relative cost of a block cipher call versus a finite field multiplication. Roughly speaking, the hash-encrypt-hash approach will be faster than the encrypt-mix-encrypt approach if one block cipher invocation is slower than two finite field multiplications. This is not true for software implementation when the block cipher is AES. On the other hand, a mode of operation is not intended to be used with only one block cipher. It is conceivable that there are (possibly proprietary) block ciphers for which this condition hold and using the hash-counter-hash approach will be better.

The number of passes gives the number of times the entire length of the message has to be read. In the encrypt-mix-encrypt type constructions the number of passes is two while in the hash-ECB-hash type constructions the number of passes is three. In the hash-Ctr-hash type constructions, the number of passes is also two. This is because, the second layer of universal hash computation can be combined with the encryption layer by rewriting the loop in a slightly different manner. (We note that for HCH, this is due to the current definition of $H_{R,Q}()$; using the definition of $H_{R,Q}()$ in the conference version [2] of this paper will require three passes.)

Irrespective of whether the number of passes is two or three, the important thing is that it is more than one. As a result, the encryption cannot be on-line, i.e., the ciphertext cannot be produced without reading (or processing) the entire message. This property is natural to expect from a *strong* pseudorandom permutation, since such a primitive tries to make each bit of the ciphertext depend on the entire message.

The number of memory accesses depends on the number of passes and also on the details of the algorithm. Suppose there are m blocks in the message to be encrypted (the reasoning about decryption is similar). A single pass algorithm will need to read the m message blocks and write the m ciphertext blocks. The behaviour of a multi-pass algorithm depends on the nature of the construction.

Encrypt-Mix-Encrypt: For such constructions, the m message blocks are read; their encryptions are written; these encryptions are read and the ciphertext blocks are written. Thus, $2m$ blocks are read and $2m$ blocks are written.

Hash-Ctr-Hash: For such constructions, the m message blocks are read and the output of the universal hash function is computed; then the encryption layer reads the message blocks once more and simultaneously computes the output of the second universal hash function and also writes the ciphertext blocks. Thus, in this case $2m$ many blocks are read but only m blocks are written.

Hash-ECB-Hash: These are three pass constructions. For PEP, $3m$ blocks are read and $3m$ blocks are written, while for TET $3m$ blocks are read and $2m$ blocks are written.

These values are given in Table 1. From the above discussion, we see that the hash-Ctr-hash constructions require the minimum number of writes. The total time required for encryption and decryption will be the time for computation as well as the time for reading and writing the blocks. Depending on the actual implementation, this value can vary quite a lot.

Table 1. Comparison of SPRPs for variable length messages using an n-bit block cipher and an n-bit tweak. Here m denotes the number of message blocks (full or partial). For HCH, we assume the modification in Section 2.1. For PEP, we assume $m \geq 3$. For TET, m' is the number of full blocks and $\imath$ is a value which depends on m (and K). We ignore constants for the security bounds. [BC]: block cipher invocation; [M]: $GF(2^n)$ multiplication; [I]: $GF(2^n)$ inversion; [R]: read a block; [W] write a block; [BCK] block cipher key; [AK] auxilliary n-bit key material; σ_n total number of n-bit blocks in all the queries. The construction types are the following. I: enc-mix-enc; II: hash-ECB-hash; III: hash-Ctr-hash.

Mode	type	sec. bnd.	comp. cost	read/write	keys	msg. len.	passes	enc. layers	parallel?
CMC	I	$\sigma_n^2/2^n$	$(2m+1)$[BC]	$2m$[R]+$2m$[W]	1[BCK]	mn, $m \geq 1$	2	2	No
EME	I	$\sigma_n^2/2^n$	$(2m+2)$[BC]	$2m$[R]+$2m$[W]	1[BCK]	mn, $1 \leq m \leq n$	2	2	Yes
EME*	I	$\sigma_n^2/2^n$	$(2m+\frac{m}{n}+1)$[BC]	$2m$[R]+$2m$[W]	1[BCK]+2[AK]	$\geq n$	2	2	Yes
PEP	II	$\sigma_n^2/2^n$	$(m+5)$[BC] +$(4m-6)$[M]+1[I]	$3m$[R]+$3m$[W]	1[BCK]	mn, $m \geq 1$	3	1	Yes
TET	II	$\sigma_n^2/2^n$	$\imath((m-1)$[M]+2[BC]) +$(m+2)$[BC] +$2m'$[M]+1[I]	$3m$[R]+$2m$[W]	2[BCK]	$\geq n$	3	1	partial
XCB	III	not given	$(m+6)$[BC] +$2(m+1)$[M]	$2m$[R]+m[W]	1[BCK]	$[n, 2^{39}]$	2	1	partial
HCTR	III	$\sigma_n^3/2^n$	m[BC] +$2(m+1)$[M]	$2m$[R]+m[W]	1[BCK]+1[AK]	$\geq n$	2	1	partial
HCH	III	$\sigma_n^2/2^n$	$(m+3)$[BC] +$2(m-1)$[M]	$2m$[R]+m[W]	1[BCK]	$\geq n$	2	1	partial

Parallelism: Some of the constructions in Table 1 are marked as being partially parallel. This means that the encryption layer is parallel but the hash computations are not. The hash computations can also be made parallel, but then Horner's rule cannot be applied for polynomial evaluation. For XCB, HCTR, HCH and TET, this increases the total number of $GF(2^n)$ mutliplications by roughly a factor of two.

Arbitrary Length Tweaks: CMC, EME, EME* and HCH support n-bit tweaks. On the other hand, XCB TET support arbitrary length tweaks, while HCTR supports arbitrary but fixed length tweaks. For HCH, it is easy to obtain a variant supporting arbitrary length tweaks by using a separate pseudorandom function which uses an independent key. This PRF will produce an n-bit digest of the tweak which will be used by the usual HCH construction. This approach is used in TET. The difference is that for TET, one needs the PRF even for n-bit tweaks. We note that n-bit tweaks are sufficient for disk encryption, since in this case sector addresses are used as tweaks.

3.1 Pre-Computation

In certain situations, one can use pre-computation to generate tables and use these during the actual encryption to speed up the computation. Two kinds of quantities may be pre-computed.

Block cipher key schedule. If the block cipher key is fixed, then it is possible to pre-compute the entire key schedule. This helps in reducing the time required for encrypting each block. While this improves speed, it also means secure storage for the pre-computed key schedules. Thus, a construction which uses more block cipher keys requires more storage space. Let us consider XCB as an example. The five keys that it requires does not depend on the message or the tweak and hence these can be pre-computed. Out of these, three are block cipher keys and to obtain good speed one may need to pre-compute and store the key schedule for all three of these. Similarly, TET uses two block cipher keys and will require to store the key schedule for both keys. Compared to this, EME*, HCTR and HCH use a single block cipher key and hence the storage space for the pre-computed key schedule will be lesser.

Table for computing polynomial hash. If one of the operands is fixed, then pre-computation can be used to speed up finite field multiplication [13]. XCB, HCTR and TET can use such pre-computation to speed up the computation of the polynomial hash. On the other hand, since HCH evaluates the polynomial at a value which depends on the tweak and the length, it is not possible to use pre-computation for speeding up polynomial arithmetic.

We note, however, that the general issue of pre-computation comes at a cost. One will require *secure* storage for the pre-computed tables (whether for block cipher key schedules or for polynomial evaluations), which can be a problem especially in multi-key situations or in hardware implementations. The related issue is of key agility. If the key changes often, then changing the tables each time can be problematic.

3.2 HCHp: A Variant Supporting Pre-Computation

Any construction of the hash-Ctr-hash type will basically require at least two keys – one for the block cipher and another one for the universal hash function. In HCH, the universal hash key is derived by encrypting the tweak and hence pre-computation cannot be used. On the other hand, if one wants to utilize pre-computation, then it is easy to modify HCH to obtain a variant for which this is possible. This variant is called HCHp and is described in Figure 5.

Fig. 5. Encryption and decryption using HCHp. The tweak is T and the key is (K, α), where K is the block cipher key and α is the universal hash key.

Algorithm $\mathbf{E}^T_{K,\alpha}(P_1, \ldots, P_m)$	**Algorithm** $\mathbf{D}^T_{K,\alpha}(C_1, \ldots, C_m)$
1. $R = E_K(T)$; $Q = E_K(R \oplus \mathsf{bin}_n(l))$;	1. $R = E_K(T)$; $Q = E_K(R \oplus \mathsf{bin}_n(l))$;
2. $M_m = \mathsf{pad}_{n-r}(P_m)$;	2. $U_m = \mathsf{pad}_{n-r}(C_m)$;
3. $M_1 = H_{\alpha,Q}(P_1, \ldots, P_{m-1}, M_m)$;	3. $U_1 = H_{\alpha,xQ}(C_1, \ldots, C_{m-1}, U_m)$;
4. $U_1 = E_K(M_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;	4. $M_1 = E_K^{-1}(U_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;
5. $(C_2, \ldots, C_{m-1}, D_m)$ $= \mathsf{Ctr}_{K,S}(P_2, \ldots, P_{m-1}, M_m)$;	5. $(P_2, \ldots, P_{m-1}, V_m)$ $= \mathsf{Ctr}_{K,S}(C_2, \ldots, C_{m-1}, U_m)$;
6. $C_m = \mathsf{drop}_{n-r}(D_m)$; $U_m = \mathsf{pad}_{n-r}(C_m)$;	6. $P_m = \mathsf{drop}_{n-r}(V_m)$; $M_m = \mathsf{pad}_{n-r}(P_m)$;
7. $C_1 = H_{\alpha,xQ}(U_1, C_2, \ldots, C_{m-1}, U_m)$;	7. $P_1 = H_{\alpha,Q}(M_1, P_2, \ldots, P_{m-1}, M_m)$;
8. return $(C_1, \ldots, C_m)$.	8. return $(P_1, \ldots, P_m)$.

The change to HCH is minimal. An n-bit (also considered to be an element of $GF(2^n)$) key α for the universal hash function is used. This is chosen uniformly at random from the set of all n-bit strings and is independent of the block cipher key K. The invocations of the hash function $H()$ is done using α instead of R. Since, α is a fixed quantity, a table can be pre-computed to speed up multiplications using α. Note that Q (which is computed from R) is used in the same manner as in HCH.

Security is not affected by the replacement of R by α in the hash function computation. The intuitive argument for this is the following. R depends upon the tweak while α does not. Thus, a problem may arise if two different queries are made with the same message but different tweaks. However, note that the $H_{\alpha,Q}()$ depends upon Q which in turn depends upon both the tweak and the length. Hence, if the tweaks are different, then the corresponding values of Q will be different, which will ensure that there is no collision for the internal variables corresponding to the two queries. More details on the collision analysis is provided in Section 6.5.

4 Fixed Length Variants

One important application of tweakable enciphering scheme is disk encryption. For such applications, the length of the message is fixed and is some multiple of the block length. The tweak is taken to be the sector address. For such applications, it is possible to simplify some of the constructions.

For XCB, as mentioned earlier, the three block cipher keys and the two polynomial hash keys can be pre-computed. (After pre-computing the keys, depending on the available secure storage space, one may or may not choose to pre-compute the key schedules and tables to speed up polynomial hash.) This makes the number of block cipher calls for an m-block message to be $(m+1)$. For TET, since the value of m is now fixed, the values of τ and σ^{-1} can be pre-computed and stored. Also, TET requires the encryption of the message length which can be pre-computed and stored. This reduces the number of block cipher calls by one and increases the storage requirement by three n-bit strings.

4.1 HCHfp: A Fixed Length Variant of HCHp

As mentioned earlier, in HCH, it is not possible to use pre-computed tables to speed up the hash function computation. The variant HCHp was introduced for this purpose. We next describe a fixed length variant HCHfp, which is a variant for fixed length messages and supports pre-computation. This is given in Figure 6.

In HCHfp, the key α for the universal hash function is chosen uniformly at random and is independent of the block cipher key. Moreover, this key is fixed for all messages and a table can be pre-computed to speed up the hash function computation. The dependence of the hash value on the tweak is achieved by invoking the hash function as $H_{\alpha,R}()$ and $H_{\alpha,xR}()$. The collision analysis of HCHfp is given in Section 6.6.

Remark: It is possible to derive α using the block cipher key by setting $\alpha = E_K(xE_K(0^n))$. This will make HCHfp a single key algorithm where the hash key (and a corresponding table) can be pre-computed if required. This may be desirable for some applications. The collision analysis of this variant is almost the same as that of HCHfp and hence is not provided.

Fig. 6. Encryption and decryption using HCHfp. The tweak is T and the key is (K, α), where K is the block cipher key and α is the universal hash key. The number of blocks m is fixed.

Algorithm $\mathbf{E}_{K,\alpha}^{T}(P_1, \ldots, P_m)$	**Algorithm** $\mathbf{D}_{K,\alpha}^{T}(C_1, \ldots, C_m)$
1. $R = E_K(T)$;	1. $R = E_K(T)$;
2. $M_m = \mathsf{pad}_{n-r}(P_m)$;	2. $U_m = \mathsf{pad}_{n-r}(C_m)$;
3. $M_1 = H_{\alpha,R}(P_1, \ldots, P_{m-1}, M_m)$;	3. $U_1 = H_{\alpha,xR}(C_1, \ldots, C_{m-1}, U_m)$;
4. $U_1 = E_K(M_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;	4. $M_1 = E_K^{-1}(U_1)$; $I = M_1 \oplus U_1$; $S = E_K(I)$;
5. $(C_2, \ldots, C_{m-1}, D_m)$ $= \mathsf{Ctr}_{K,S}(P_2, \ldots, P_{m-1}, M_m)$;	5. $(P_2, \ldots, P_{m-1}, V_m)$ $= \mathsf{Ctr}_{K,S}(C_2, \ldots, C_{m-1}, U_m)$;
6. $C_m = \mathsf{drop}_{n-r}(D_m)$; $U_m = \mathsf{pad}_{n-r}(C_m)$;	6. $P_m = \mathsf{drop}_{n-r}(V_m)$; $M_m = \mathsf{pad}_{n-r}(P_m)$;
7. $C_1 = H_{\alpha,xR}(U_1, C_2, \ldots, C_{m-1}, U_m)$;	7. $P_1 = H_{\alpha,R}(M_1, P_2, \ldots, P_{m-1}, M_m)$;
8. return $(C_1, \ldots, C_m)$.	8. return $(P_1, \ldots, P_m)$.

4.2 Comparison Among Fixed Length Variants

The comparison for fixed length inputs is given in Table 2. As mentioned earlier, the comparison between encrypt-mix-encrypt and hash-encrypt-hash approaches depends on the relative efficiency of a block cipher invocation to a finite field multiplication. If a block cipher invocation is slower than two multiplications, then the second approach is faster. Otherwise, the first approach is faster.

Table 2. Comparison of different tweakable SPRPs for fixed length inputs using an n-bit block cipher and an n-bit tweak. The fixed number of blocks is $m > 1$ and each block is of length n bits. [BC]: block cipher invocation; [M]: $GF(2^n)$ multiplication; [R]: read a block; [W]: write a block; [BCK]: block cipher key; [AK]: auxiliary n-bit string (including polynomial hash keys).

Mode	CMC	EME*	XCB	TET	HCH	HCHfp
sec bnd.	$\sigma_n^2/2^n$	$\sigma_n^2/2^n$	not given	$\sigma_n^2/2^n$	$\sigma_n^2/2^n$	$\sigma_n^2/2^n$
[BC]	$2m+1$	$2m+1+m/n$	$m+1$	$m+1$	$m+3$	$m+2$
[M]	–	–	$2(m+3)$	$2m$	$2(m-1)$	$2(m-1)$
number of passes	2	2	2	3	2	2
read/write	$2m$[R]+$2m$[W]	$2m$[R]+$2m$[W]	$2m$[R]+m[W]	$3m$[R]+$2m$[W]	$2m$[R]+m[W]	$2m$[R]+m[W]
[BCK]	1	1	3	2	1	1
[AK]	–	2	2	3	–	1
precomp (key sch)	yes	yes	yes	yes	yes	yes
precomp (mult table)	–	–	yes	yes	no	yes

Table 3. Efficiency of key change. For TET, the value of $\imath$ depends only on K (since the number of blocks m is fixed). [I]: inversion.

Mode	CMC	EME*	XCB	TET	HCH	HCHfp
comp. cost	–	–	5[BC]	$\imath((m-1)$[M]+2[BC])+1[BC]+1[I]	–	–
key sch.	1	1	3	1	1	1
mult. tab.	–	–	2	1	–	1

Let us now consider the constructions in the hash-encrypt-hash approaches. Table 2 lists two other constructions of this type – XCB and TET. XCB is of the hash-counter-hash type while TET is of the hash-ECB-hash type. We do not include HCTR in the comparison table since it has a cubic security bound. XCB, on the other hand, is included because it is the first hash-Ctr-hash type construction (even though it does not have a security bound). The variant HCHfp is given for comparison. The difference between these two variants is that HCH uses a single block cipher key but cannot use pre-computation to speed up the hash computations while HCHfp uses a block cipher key and an n-bit hash key and can use pre-computation to speed up the hash computations. Also, HCHfp makes one less block cipher call. The other features are the same.

The number of block cipher calls and the number of multiplications made by XCB, TET and the two HCH variants are roughly the same. If pre-computation is used, then XCB, TET and HCHfp have similar efficiencies.

Key Agility: Table 3 provides the costs for the different algorithms when a key is changed. The first row gives the computation cost, which includes block cipher invocations and finite field operations. These costs are matched with Table 2 in the sense that if less is pre-computed, then the computation costs in Table 2 will increase.

In terms of key agility, CMC, EME* and HCH are the best, since only one block cipher key schedule has to be computed on a key change. (EME* requires two additional n-bit keys.) Next comes HCHfp, which requires computation of one block cipher key schedule and one multiplication table. From the viewpoint of key agility, XCB and TET are significantly slower compared to the other four with XCB requiring the maximum amount of secure storage space. TET requires a finite field inversion for every key change. This necessitates an inversion circuit for hardware implementation, which is not required by the other five modes.

5 Security of HCH and the Variants

5.1 Definitions and Notation

The discussion in this section is based on [6]. An n-bit block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, where $\mathcal{K} \neq \emptyset$ is the key space and for any $K \in \mathcal{K}$, $E(K,.)$ is a permutation. We write $E_K(\,)$ instead of $E(K,.)$.

An adversary A is a probabilistic algorithm which has access to some oracles and which outputs either 0 or 1. Oracles are written as superscripts. The notation $A^{\mathcal{O}_1,\mathcal{O}_2} \Rightarrow 1$ denotes the event that the adversary A, interacts with the oracles $\mathcal{O}_1, \mathcal{O}_2$, and finally outputs the bit 1. In what follows, by the notation $X \xleftarrow{\$} \mathcal{S}$, we will denote the event of choosing X uniformly at random from the set $\mathcal{S}$.

Let Perm(n) denote the set of all permutations on $\{0,1\}^n$. We require $E(,)$ to be a strong pseudorandom permutation. The advantage of an adversary in breaking the strong pseudorandomness of $E(,)$ is defined in the following manner.

$$\mathbf{Adv}_E^{\pm\mathrm{prp}}(A) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{E_K(\,),E_K^{-1}(\,)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\pi(\,),\pi^{-1}(\,)} \Rightarrow 1\right]\right|.$$

Formally, a tweakable enciphering scheme is a function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\mathcal{K} \neq \emptyset$ and $\mathcal{T} \neq \emptyset$ are the key space and the tweak space respectively. The message and the cipher spaces are $\mathcal{M}$. For HCH we have $\mathcal{M} = \cup_{i>n}\{0,1\}^i$. We shall write $\mathbf{E}_K^T(.)$ instead of $\mathbf{E}(K,T,.)$. The inverse of an enciphering scheme is $\mathbf{D} = \mathbf{E}^{-1}$ where $X = \mathbf{D}_K^T(Y)$ if and only if $\mathbf{E}_K^T(X) = Y$.

Let Perm$^{\mathcal{T}}(\mathcal{M})$ denote the set of all functions $\boldsymbol{\pi} : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ where $\boldsymbol{\pi}(\mathcal{T},.)$ is a length preserving permutation. Such a $\boldsymbol{\pi} \in$ Perm$^{\mathcal{T}}(\mathcal{M})$ is called a tweak indexed permutation. For a tweakable enciphering scheme $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, we define the advantage an adversary A has in distinguishing $\mathbf{E}$ and its inverse from a random tweak indexed permutation and its inverse in the following manner.

$$\mathbf{Adv}_{\mathbf{E}}^{\pm\widetilde{\mathrm{prp}}}(A) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right|.$$

Pointless queries: We assume that an adversary never repeats a query, i.e., it does not ask the encryption oracle with a particular value of (T, P) more than once and neither does it ask the decryption oracle with a particular value of (T, C) more than once. Furthermore, an adversary never queries its deciphering oracle with (T, C) if it got C in response to an encipher query (T, P) for some P. Similarly, the adversary never queries its enciphering oracle with (T, P) if it got P as a response to a decipher query of (T, C) for some C. These queries are called *pointless* as the adversary knows what it would get as responses for such queries.

Following [6], we define the query complexity σ_n of an adversary as follows. A string X contributes $\max(|X|/n, 1)$ to the query complexity. A tuple of strings $(X_1, X_2, \ldots)$ contributes the sum of the contributions from all oracle queries plus the contribution from the adversary's output. Suppose an adversary makes q queries where the number of n-bit blocks in the ith query is ℓ_i. Then, $\sigma_n = 1 + \sum_{i=1}^{q}(1 + \ell_i) \geq 2q$. Let ρ be a list of resources used by the adversary A and suppose $\mathbf{Adv}_{\mathcal{E}}^{\pm\mathrm{xxx}}(A)$ has been defined where $\mathcal{E}$ is either a block cipher or a tweakable enciphering scheme. $\mathbf{Adv}_{\mathcal{E}}^{\pm\mathrm{xxx}}(\rho)$ denotes the maximal value of $\mathbf{Adv}_{\mathcal{E}}^{\pm\mathrm{xxx}}(A)$ over all adversaries A using resources at most ρ. Usual resources of interest are the running time t of the adversary, the number of oracle queries q made by the adversary and the query complexity σ_n $(n \geq 1)$.

The notation HCH$[E]$ denotes a tweakable enciphering scheme, where the n-bit block cipher E is used in the manner specified by HCH. Our purpose is to show that HCH$[E]$ is secure if E is secure. The notation HCH[Perm(n)] denotes a tweakable enciphering scheme obtained by plugging in a random permutation from Perm(n) into the structure of HCH. For an adversary attacking HCH[Perm(n)], we do not put any bound on the running time of the adversary, though we still put a bound on the query complexity σ_n. We show the information theoretic security of HCH[Perm(n)] by obtaining an upper bound on $\mathbf{Adv}_{\mathrm{HCH[Perm}(n)]}^{\pm\widetilde{\mathrm{prp}}}(\sigma_n)$. The upper bound is obtained in terms of n and σ_n. For a fixed block cipher E, we bound $\mathbf{Adv}_{\mathrm{HCH}[E]}^{\pm\widetilde{\mathrm{prp}}}(\sigma_n, t)$ in terms of $\mathbf{Adv}_{E}^{\pm\mathrm{prp}}(\sigma_n, t')$, where $t' = t + O(\sigma_n)$. We will use the notation $\mathbf{E}_\pi$ as a shorthand for HCH[Perm(n)] and $\mathbf{D}_\pi$ will denote the inverse of $\mathbf{E}_\pi$. Thus, the notation $A^{\mathbf{E}_\pi,\mathbf{D}_\pi}$ will denote an adversary interacting with the oracles $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$.

5.2 Statement of Results

The following theorem specifies the security of HCH.

Theorem 1. *Fix n, q and $\sigma_n \geq 2q$ to be positive integers and an n-bit block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Then*

$$\mathbf{Adv}_{\mathrm{HCH[Perm}(n)]}^{\pm\widetilde{\mathrm{prp}}}(\sigma_n) \leq \frac{7\sigma_n^2}{2^n}. \tag{5}$$

$$\mathbf{Adv}_{\mathrm{HCH}[E]}^{\pm\widetilde{\mathrm{prp}}}(\sigma_n, t) \leq \frac{7\sigma_n^2}{2^n} + \mathbf{Adv}_{E}^{\pm\mathrm{prp}}(\sigma_n, t') \tag{6}$$

where $t' = t + O(\sigma_n)$. Equations (5) and (6) also hold when HCH *is replaced by its variants* HCHp *and* HCHfp.

The above result and its proof is similar to previous work (see for example [6, 7, 3]). As mentioned in [6], Equation (6) embodies a standard way to pass from the information theoretic setting to the complexity theoretic setting. We briefly provide the argument. For any adversary A, we have the following.

$$\begin{aligned}
\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{HCH}[E]}(A) &= \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right] \\
&= \left(\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right) \\
&\quad + \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right. \\
&\quad \left. - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&= X + \mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A)
\end{aligned}$$

where $X = \left(\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right)$. The quantity X represents an adversary's advantage in distinguishing HCH[E] from HCH[π], where π is a randomly chosen permutation from Perm(n). Clearly, such an adversary A can also distinguish E from a random permutation and hence $X \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{E}(A)$. This argument shows how (6) is obtained from (5).

For proving (5), we need to consider an adversary's advantage in distinguishing a tweakable enciphering scheme $\mathbf{E}$ from an oracle which simply returns random bit strings. This advantage is defined in the following manner.

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A) = \left|\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right] - \Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right]\right|$$

where $\$(., M)$ returns random bits of length $|M|$.

The basic idea of proving (5) is as follows.

$$\begin{aligned}
\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A) &= \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right. \\
&\quad \left. - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&= \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right] - \Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right]\right) \\
&\quad + \left(\Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&\leq \mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A) + \binom{q}{2}\frac{1}{2^n} \qquad (7)
\end{aligned}$$

where q is the number of queries made by the adversary. For a proof of the last inequality see [6].

The main task of the proof now reduces to obtaining an upper bound on $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(\sigma_n)$. This proof is provided in Section 6, where we show (see (20)) that for any adversary having query complexity σ_n, we have

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(\sigma_n) \leq \frac{6\sigma_n^2}{2^n}. \qquad (8)$$

Using this and (7), we obtain

$$\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(\sigma_n) \le \frac{7\sigma_n^2}{2^n}.$$

The proof of (8) uses the standard technique of sequence of games between an adversary and the mode of operation HCH. The proof is similar to the corresponding proofs of CMC [6] and EME [7]. By a sequence of games we show that if in response to any valid query of the adversary, random strings of appropriate lengths are returned then the internal computations of HCH can be performed consistently under the assumption that the block cipher and its inverse are random permutations. The crux of the proof lies in showing that there would seldom be any collisions in the range and domain sets of the block cipher if the adversary queries HCH with valid queries and HCH responds to them by producing random strings. In a later part we remove the randomness associated with the adversary and the game runs on a fixed transcript consisting of the queries and their responses. We show that in such a situation also the internal computations of HCH can be performed consistently.

6 Proof of Theorem 1

We describe the detailed proof for Theorem 1 for HCH. The proofs for the variants are similar. At the end of the proof for HCH, we provide the details of the variables which arise for each of the variants and among which the collision analysis is to be made.

We prove the upper bound on $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(q, \sigma_n)$. Recall that this is the difference in the probabilities of the adversary producing one when interacting with HCH instantiated by a random permutation ($\mathbf{E}_\pi$ and $\mathbf{D}_\pi$) and when interacting with an oracle which returns random strings on any input.

We model the adversary's interaction with the oracles $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ as a game. In the usual game, which we call HCH1, the adversary submits queries to $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ and gets appropriate answers. Starting from this game, we modify it in successive steps to obtain games where the adversary is provided random bit strings of appropriate lengths. This results in a sequence of games: HCH1, RAND1, RAND2, RAND3 and NON. In the final game, both the plaintext and the ciphertext are allowed to be fixed by the adversary. The point is then to show that even in this case, with high probability it is possible to adjust the internal variables so as to ensure that these plaintext-ciphertext pairs are consistent with the HCH mode of operation.

6.1 Overview

Before presenting the details of the games, we describe the idea behind the games. This may help in understanding the details of each game.

Game HCH1: In this game, the adversary interacts with HCH where the block cipher is instantiated by a random permutation π. For every invocation of π (resp. π^{-1}), we generate a random string; if this string is already in the domain (resp. range) of π, then we set a flag `bad` to true and randomly choose a string which is not currently in the domain (resp. range) of π.

Game RAND1: In this game, we do away with the explicit domain and range checks (and as a result the adversary obtains random strings in response to any query). So, if `bad` is not set to true, then Games HCH1 and RAND1 are same. The setting of `bad` to true means that there is either a domain or a range collision. The whole point of the rest of the proof is to upper bound the probability of such collisions.

Game RAND2: In this game, in response to any query, we explicitly return random strings to the adversary. Then, the internal variables are adjusted to ensure that whatever is returned to the adversary is consistent with the mode of operation. If this adjustment cannot be done, then bad is set to true. The probabilities of bad being set to true is the same for both games RAND1 and RAND2. Also, in this game the adversary is effectively interacting with an oracle which provides random strings on any input.

Game RAND3: In this game, we first return the random strings to the adversary and then try to adjust the internal variables. Also, in this game, we maintain the current domain and range of π as multisets. If a value occurs twice in this multiset, then there is a collision. The changes from RAND2 to RAND3 are technical in nature and does not affect the adversary's view in any manner.

Game NON: We first upper bound the probability that there is a collision in the output of the counter mode over all the queries. (This is also the probability that the quantities $P_i \oplus C_i$ are all distinct.) This probability is $\sigma_n^2/2^n$. The set of queries by the adversary and the responses to them constitute a transcript of the game. We now consider all transcripts such that outputs of the counter mode are distinct over all the queries. Such transcripts are called allowed transcripts. To obtain the game NON (for non-interactive) we fix a transcript which maximizes the advantage of the adversary. We then show that for such a transcript the probability of bad being set to true is small. Structurally, games RAND3 and NON are exactly the same though the interpretation of the variables are somewhat different.

Combinatorial analysis: The task of showing the probability of bad being set to true boils down to a combinatorial analysis. We carefully identify the quantities in the domain and range sets and then upper bound the probability that two elements in the domain or two elements in the range are equal. Most of this is routine except for the polynomial hash output. We discuss this in more details.

Polynomial hash and the effect of the extra block cipher invocation: In HCTR, the counter is initialized with the XOR of the input and the output of the block cipher invocation for the first block. In the game NON, where both the plaintext and the ciphertext are fixed, this results in the counter being initialized with the XOR of two polynomial hashes. Consequently, for each query of length m, there are m quantities in the domain which are variants of these polynomial hashes. As a result, almost all the quantities in the domain are some kind of polynomial hashes. A collision analysis of so many polynomial hash quantities makes the bound cubic.

Let us now consider the situation of HCH. For each query, the counter is initialized by the output of an encryption. In Game NON, the XOR of the polynomial hashes of the plaintext and ciphertext enters the domain once as input to the extra block cipher invocation. Thus, in HCH, for each query, we have only one polynomial hash quantity to take care of. This makes it possible to obtain a quadratic security bound.

6.2 The Games

By an abuse of notation, we will use A^{HCH1} to denote an adversary A's interaction with the oracles while playing game HCH1. We will use similar notations for the other games. The detailed pseudocode for the different games are given in Section A.

Game HCH1: We describe the attack scenario of the adversary A through a probabilistic game which we call HCH1. In HCH1, the adversary interacts with $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ where π is a randomly chosen permutation from Perm(n). Instead of initially choosing π, we build up π in the following manner.

Initially π is assumed to be undefined everywhere. When $\pi(X)$ is needed, but the value of π is not yet defined at X, then a random value is chosen among the available range values. Similarly when $\pi^{-1}(Y)$ is required and there is no X yet defined for which $\pi(X) = Y$, we choose a random value for $\pi^{-1}(Y)$ from the available domain values.

The domain and range of π are maintained in two sets *Domain* and *Range*, and $\overline{Domain}$ and $\overline{Range}$ are the complements of *Domain* and *Range* relative to $\{0,1\}^n$. The game HCH1 is shown in Figure 7. The figure shows the subroutines Ch-π, Ch-π^{-1}, the initialization steps and how the game responds to a encipher/decipher query of the adversary. The i^{th} query of the adversary depends on its previous queries, the responses to those queries and on some coins of the adversary.

The game HCH1 accurately represents the attack scenario, and by our choice of notation, we can write

$$\Pr[A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1] = \Pr[A^{\mathrm{HCH1}} \Rightarrow 1]. \tag{9}$$

Game RAND1: We modify HCH1 by deleting the boxed entries in HCH1 and call the modified game as RAND1. By deleting the boxed entries it cannot be guaranteed that π is a permutation as though we do the consistency checks but we do not reset the values of Y (in Ch-π) and X (in Ch-π^{-1}). Thus, the games HCH1 and RAND1 are identical apart from what happens when the bad flag is set. So,

$$|\Pr[A^{\mathrm{HCH1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND1}} \Rightarrow 1]| \leq \Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}] \tag{10}$$

Game RAND2: We make certain changes to the game RAND1 which are invisible to the adversary. In RAND1 as the permutation π is not maintained, thus the subroutine Ch-π and Ch-π^{-1} are no more needed. Instead we add a new subroutine called Check-Domain-Range(X, Y). In Check-Domain-Range(X, Y), X is inserted into *Domain* and Y is inserted into *Range*; also, if $X \in$ *Domain* or $Y \in$ *Range* then the bad flag is set.

In this game, for an encryption query, we choose the ciphertext blocks to be random n-bit strings and return to the adversary. Then we adjust the internal variables so as to ensure that the particular choice of ciphertext blocks is consistent as per the protocol. Similarly, for a decryption query, we choose the plaintext blocks to be random n-bit strings and return to the adversary and then adjust the internal variables. This does not alter the adversary's view of the game since for each such change the adversary obtains a random n-bit string both before and after the change. Thus,

$$\Pr[A^{\mathrm{RAND1}} \Rightarrow 1] = \Pr[A^{\mathrm{RAND2}} \Rightarrow 1] \tag{11}$$

also,

$$\Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}] = \Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] \tag{12}$$

In RAND2 the adversary is supplied with random bits as response to queries to both the encrypt and the decrypt oracles. Hence,

$$\Pr[A^{\mathrm{RAND2}} \Rightarrow 1] = \Pr[A^{\$(.,.),\$(.,.)} \Rightarrow 1] \tag{13}$$

Now, from Equation (9), (10), (11), (12) and (13) we get

$$\begin{aligned}\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A) &= |\Pr[A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1] - \Pr[A^{\$(.,.),\$(.,.)} \Rightarrow 1]| \qquad (14)\\ &= |\Pr[A^{\mathrm{HCH1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND2}} \Rightarrow 1]|\\ &= |\Pr[A^{\mathrm{HCH1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND1}} \Rightarrow 1]|\\ &\le \Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}]\\ &= \Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] \qquad (15)\end{aligned}$$

Our task is thus to bound $\Pr[A^{\mathrm{RAND2}}$ sets $\mathsf{bad}]$.

Game RAND3: Here we make two subtle changes to the game RAND2. Here instead of the *Domain* and *Range* sets we use multisets $\mathcal{D}$ and $\mathcal{R}$ respectively. In the game RAND3 on either an encryption or a decryption query by the adversary a random string is given as output. Next, the internal variables are adjusted in the first phase of the finalization step. The bad flag is set at the second phase of the finalization step by checking whether a value occurs in either $\mathcal{R}$ or $\mathcal{D}$ more than once. The game RAND3 is shown in Figure 9. RAND3 sets bad in exactly the same conditions in which RAND2 sets bad, hence

$$\Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] = \Pr[A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}]. \qquad (16)$$

Game NON: In Game RAND3 consider the variable Y_i^s which is defined as

$$\begin{aligned}Y_i^s &= C_i^s \oplus P_i^s, \text{ when } 1 \le i \le m^s - 1;\\ &= D^s_{m^s} \oplus M^s_{m^s}, \text{ when } i = m^s \text{ and } ty = \mathsf{Enc};\\ &= U^s_{m^s} \oplus V^s_{m^s}, \text{ when } i = m^s \text{ and } ty = \mathsf{Dec}.\end{aligned}$$

For $2 \le i \le m^s$, the variable Y_i^s enters the range set and it is always an n-bit random quantity. As, when $2 \le i \le m^s - 1$, then for a encryption (resp. decryption) query C_i^s (resp. P_i^s) is a randomly chosen n-bit string. When $i = m^s$, then for an encryption (resp. decryption) query $D^s_{m^s}$ (resp. $V^s_{m^s}$) is a randomly chosen n-bit string. Thus, for $(s,i) \neq (t,j)$, $\mathsf{Pr}[Y_i^s = Y_j^t] = \frac{1}{2^n}$. The condition $Y_i^s = Y_j^t$ for some $(s,i) \neq (t,j)$ leads to a collision in the range and results in bad being set to true. The total probability of bad being set to true due to collisions of this kind is at most $\binom{|\mathcal{R}|}{2}/2^n \le \sigma_n^2/2^n$. (Note that $|\mathcal{R}| \le \sigma_n$, where σ_n is the query complexity.) Let $\mathbf{X}$ be the event that bad is set to true in Game RAND3 due to collisions of this kind. Then we have

$$\begin{aligned}\Pr[A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}] &= \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}) \wedge (\mathbf{X} \vee \overline{\mathbf{X}})]\\ &= \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}) \wedge \mathbf{X}] + \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}) \wedge \overline{\mathbf{X}}]\\ &= \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad})|\mathbf{X}]\Pr[\mathbf{X}] + \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad})|\overline{\mathbf{X}}]\Pr[\overline{\mathbf{X}}]\\ &\le \Pr[\mathbf{X}] + \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad})|\overline{\mathbf{X}}]\\ &\le \frac{\sigma_n^2}{2^n} + \Pr[(A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad})|\overline{\mathbf{X}}].\end{aligned}$$

Our next task is to upper bound $\Pr[(A^{\mathrm{RAND3}}$ sets $\mathsf{bad})|\overline{\mathbf{X}}]$. The condition $\overline{\mathbf{X}}$ translates into the fact that we can assume all the Y_i^s's to be distinct. We consider the adversarial behaviour under this condition.

In the previous games, for an encipher query, the adversary specified the tweak and the plaintext; and for a decipher query, he specified the tweak and the ciphertext. We now consider the stronger condition, whereby the adversary specifies the tweak, the plaintext and the ciphertext in both the encryption and the decryption queries subject to the condition that the Y_i^s's are all distinct. For $2 \leq i \leq m^s - 1$, $Y_i^s = P_i^s \oplus C_i^s$ and hence is determined entirely by the transcript. On the other hand, the last block can be partial and hence in this case $Y_{m^s}^s$ is not entirely determined by $P_{m^s}^s \oplus C_{m_s}^s$. There are two ways to tackle this situation. In the first way, we allow the adversary to specify an additional $(n - r^s)$-bit string, which when appended to $(P_{m^s}^s \oplus C_{m_s}^s)$ forms $Y_{m^s}^s$. In the second way, we can generate this $(n - r^s)$-bit string within the game itself. We prefer the first way, since this is notationally simpler. The effect of both the methods are same since we require that Y_i^s's are distinct for all s and $i = 2, \ldots, m^s$.

We do this by modifying the game RAND3 into a new game NON (non-interactive). NON depends on a fixed transcript $\mathsf{tr} = (\mathbf{ty}, \mathbf{T}, \mathbf{P}, \mathbf{C}, \mathbf{E})$ with $\mathbf{ty} = (ty^1, ty^2, \ldots, ty^q)$, $\mathbf{T} = (\mathsf{T}^1, \mathsf{T}^2, \ldots \mathsf{T}^q)$, $\mathbf{P} = (\mathsf{P}^1, \mathsf{P}^2, \ldots \mathsf{P}^q)$, $\mathbf{C} = (\mathsf{C}^1, \mathsf{C}^2, \ldots \mathsf{C}^q)$ and $\mathbf{E} = (\mathsf{E}^1, \mathsf{E}^2, \ldots, \mathsf{E}^q)$ where $ty^s = \{\mathsf{Enc}, \mathsf{Dec}\}$, $\mathsf{T}^s \in \{0,1\}^n$, $\mathsf{P}^s = \mathsf{P}_1^s, \mathsf{P}_2^s, \ldots, \mathsf{P}_{m_s}^s$, $\mathsf{C}^s = \mathsf{C}_1^s, \mathsf{C}_2^s, \ldots, \mathsf{C}_{m_s}^s$ and each E^s is a string of length $(n - r^s)$ such that $\mathsf{Y}_{m^s}^s = (\mathsf{P}_{m_s}^s \oplus \mathsf{C}_{m_s}^s) || \mathsf{E}^s$. If this fixed transcript does not contain pointless queries and satisfies the condition that the Y_i^s's are all distinct, then the transcript is called *allowed*.

Now fix an allowed transcript tr which maximizes the probability of bad being set. This transcript tr is hardwired into the game NON. The syntax of NON is the same as the syntax of RAND3, except that the part before the finalization step is not present in NON. The main difference between NON and RAND3 is in the interpretation of the variables. The tweaks, plaintext and ciphertext blocks in RAND3 are given by the adversary while in NON they are part of the transcript tr which is hardwired into the game. We denote this difference by using the symbols $\mathsf{T}^s, \mathsf{P}_i^s$, C_i^s and Y_i^s to denote the tweaks, plaintext, ciphertext and the XOR blocks respectively in game NON. We have

$$\Pr[A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}] \leq \Pr[A^{\mathrm{NON}} \text{ sets } \mathsf{bad}] + \frac{\sigma_n^2}{2^n}. \tag{17}$$

6.3 A Useful Result

The analysis of NON requires a result, which we state and prove here. Let $\mathcal{L} = (\mathcal{L}^1, \ldots, \mathcal{L}^q)$ be a list of vectors, with $\mathcal{L}^{i_1} \neq \mathcal{L}^{i_2}$ for $i_1 \neq i_2$ and where $\mathcal{L}^i = (L_1^i, \ldots, L_{m^i}^i)$ with each L_j^i is an n-bit string considered to be an element of $GF(2^n)$. Let $\mathcal{S}$ be the set of all i, j such that $m^i = m^j$. In other words, if $i, j \in \mathcal{S}$, then $\mathcal{L}^i$ and $\mathcal{L}^j$ have the same number of components. For $\{i, j\} \in \mathcal{S}$, define a polynomial $P_{i,j}(X) \in GF(2^n)[X]$ as

$$P_{i,j}(X) = (L_1^i \oplus L_1^j) \oplus (L_2^i \oplus L_2^j)X \oplus (L_3^i \oplus L_3^j)X^2 \oplus \cdots \oplus (L_{m^j}^i \oplus L_{m^i}^j)X^{m^i - 1}. \tag{18}$$

We do not distinguish between $P_{i,j}(X)$ and $P_{j,i}(X)$. The coefficients of this polynomial are elements of $GF(2^n)$. Let $\mathbf{e}_{i,j}$ be the following event. Choose a random element R from $GF(2^n)$ and evaluate $P_{i,j}(R)$: $\mathbf{e}_{i,j}$ is the event $P_{i,j}(R) = 0$. Define

$$\mathbf{e} = \bigvee_{\{i,j\} \in \mathcal{S}} \mathbf{e}_{i,j}. \tag{19}$$

The probability of $\mathbf{e}$ is given by the following proposition.

Proposition 1. $\Pr[\mathbf{e}] \leq \frac{1}{2^n} \left(\sum_{j=1}^{q} m^j \right)^2$.

Proof : Suppose the different possible distinct lengths of the $\mathcal{L}_i$'s are $l_1, \ldots, l_p$, for some $p \leq q$. Further, let the number of $\mathcal{L}_i$'s of length l_k be t_k. Then we have $\sum_{k=1}^{p} l_k t_k = \sum_{j=1}^{q} m^j$. All the $\mathcal{L}_i$'s of length l_k contribute $\binom{t_k}{2}$ pairs to $\mathcal{S}$.

Consider the polynomial $P_{i,j}(X)$. This is a non-zero polynomial, since for $i \neq j$, we have $\mathcal{L}^i \neq \mathcal{L}^j$. The degree is at most $m^i - 1$. Let $m^i = l_k$ for some k in $\{1, \ldots, p\}$. Thus $P_{i,j}(X)$ has at most $l_k - 1$ distinct roots over $GF(2^n)$. We have $P_{i,j}(R) = 0$, if the randomly chosen R is equal to one of the roots. The probability of this happening is at most $(l_k - 1)/2^n$. Since there are $\binom{t_k}{2}$ pairs in $\mathcal{S}$ corresponding to length l_k, the contribution of all tuples of length l_k to $\mathbf{e}$ is at most $\binom{t_k}{2}(l_k - 1)/2^n$. Hence,

$$
\begin{aligned}
\mathsf{Pr}[\mathbf{e}] &\leq \binom{t_1}{2}\frac{(l_1-1)}{2^n} + \cdots + \binom{t_p}{2}\frac{(l_p-1)}{2^n} \\
&\leq \frac{1}{2^n}\sum_{k=1}^{p} t_k^2 l_k \leq \frac{1}{2^n}\sum_{k=1}^{p} t_k^2 l_k^2 \leq \frac{1}{2^n}\left(\sum_{k=1}^{p} t_k l_k\right)^2 = \frac{1}{2^n}\left(\sum_{j=1}^{q} m^j\right)^2 .
\end{aligned}
$$

The above bound can be improved by using Cauchy-Schwartz inequality in a different manner. (This was pointed out to us by Mridul Nandi.) However, this improvement only improves the constant in the bound of Theorem 1.

□

6.4 Analysis of NON

In the analysis we consider the sets $\mathcal{D}$ and $\mathcal{R}$ to consist of the formal variables instead of their values. For example, whenever we set $\mathcal{D} \leftarrow \mathcal{D} \cup \{X\}$ for some variable X we think of it as setting $\mathcal{D} \leftarrow \mathcal{D} \cup \{\text{“}X\text{”}\}$ where “X” is the name of that formal variable. This is the same technique as used in [6]. Our goal is to bound the probability that two formal variables in the sets $\mathcal{D}$ and $\mathcal{R}$ take the same value. The formal variables which enter $\mathcal{D}$ and $\mathcal{R}$ are the following:

Elements in $\mathcal{D}$:
T^s, $\mathsf{R}^s \oplus \mathsf{bin}(l^s)$,
$\mathsf{M}_1^s = \mathsf{Q}^s \oplus \mathsf{P}_1^s \oplus \mathsf{P}_2^s(\mathsf{R}^s)^{m^s-1} \oplus \cdots \oplus \mathsf{P}_{m^s-1}^s(\mathsf{R}^s)^2 \oplus \mathsf{M}_{m^s}^s\mathsf{R}^s$,
$\mathsf{I}^s = \mathsf{M}_1^s \oplus \mathsf{U}_1^s = (x+1)\mathsf{Q}^s \oplus \mathsf{Y}_1^s \oplus \mathsf{Y}_2^s(\mathsf{R}^s)^{m^s-1} \oplus \cdots \oplus \mathsf{Y}_{m^s}^s\mathsf{R}^s$,
$\mathsf{S}_1^s, \mathsf{S}_2^s, \ldots, \mathsf{S}_{m^s-1}^s$.

Elements in $\mathcal{R}$:
R^s, Q^s,
$\mathsf{U}_1^s = x\mathsf{Q}^s \oplus \mathsf{C}_1^s \oplus \mathsf{C}_2^s(\mathsf{R}^s)^{m^s-1} \oplus \cdots \oplus \mathsf{C}_{m^s-1}^s(\mathsf{R}^s)^2 \oplus \mathsf{U}_{m^s}^s\mathsf{R}^s$,
S^s,
$(\mathsf{P}_2^s \oplus \mathsf{C}_2^s), \ldots, (\mathsf{P}_{m^s-1}^s \oplus \mathsf{C}_{m^s-1}^s), (\mathsf{M}_{m^s}^s \oplus \mathsf{U}_{m^s}^s)$.

Now let us consider the probability of bad being set under the particular (allowed) transcript hardwired into NON. The variable bad can be set either as a result of a collision in the domain or as a result of a collision in the range. We consider these two separately. The number of blocks in a particular adversarial query is the length of the query. Suppose the distinct lengths of the queries made by the adversary are $l_1, \ldots, l_p$ and the adversary makes t_k queries of length l_k. Then $\sum_{k=1}^{p} l_k t_k = \sum_{s=1}^{q} m^s$.

There are $\binom{|\mathcal{D}|}{2}$ (unordered) pairs of distinct variables in $\mathcal{D}$. We have to consider the probability that such a pair collides, i.e., both the variables of the pair get the same value. We identify the following two types of pairs of variables and call them special pairs.

1. $(\mathsf{M}_1^s, \mathsf{M}_1^t)$, such that $s \neq t$ and $(\mathsf{T}^s, l^s) = (\mathsf{T}^t, l^t)$.
2. $(\mathsf{I}^s, \mathsf{I}^t)$, such that $s \neq t$ and $(\mathsf{T}^s, l^s) = (\mathsf{T}^t, l^t)$.

The number of pairs of each kind is at most $\sum_{k=1}^{p} \binom{l_k}{2}$. The total number of special pairs is at most $2\sum_{k=1}^{p} \binom{l_k}{2}$.

First let us consider the collision probability of the special pairs. The total (over all s and t) probability of the pairs of either the first or the second kind giving rise to a collision is given by Proposition 1 in Section 6.3 to be at most $\sigma_n^2/2^n$. Note that since queries are not pointless, we will have $(\mathsf{P}_1^s, \mathsf{P}_2^s, \ldots, \mathsf{P}_{m^s-1}^s, \mathsf{P}_{m^s}^s) \neq (\mathsf{P}_1^t, \mathsf{P}_2^t, \ldots, \mathsf{P}_{m^t-1}^t, \mathsf{P}_{m^t}^t)$ when $(\mathsf{T}^s, l^s) = (\mathsf{T}^t, l^t)$. Also, in an allowed transcript, all the Y_i^s's (for $1 \le i \le m^s - 1$) are distinct. These ensure that we can apply Proposition 1 to the above two cases.

For any non-special pair, the probability of collision is either 0 or equal to $1/2^n$. The actual proof is a tedious case analysis, but is based on a few observations.

- The value of T^s is never repeated in $\mathcal{D}$. According to the game, if a tweak is repeated, then it does not enter the domain again. As a consequence, the earlier value of R is used.
- If a tweak is repeated *and* length is kept same, then the earlier value of Q is used. These two points ensure that there is no collision due to repetition of the tweak and/or length.
- The variables R, Q and S are always generated randomly. Hence, when these appear as additive terms in some expression, they ensure that the corresponding quantity is a random n-bit string.
- By definition, the values S_i^s's are all distinct, and therefore $\Pr[\mathsf{S}_i^s = \mathsf{S}_j^s] = 0$ for $i \neq j$. As S^s and S^t are independent and random n-bit strings, so $\Pr[\mathsf{S}_i^s = \mathsf{S}_j^t] = 1/2^n$, for $s \neq t$. See Equation (4).

The total probability of a domain collision is the sum of the probabilities of collision between special pairs and non-special pairs. The total number of non-special pairs is at most $\binom{|\mathcal{D}|}{2})|$. Thus, the total probability of a domain collision is at most

$$\binom{|\mathcal{D}|}{2}\frac{1}{2^n} + \frac{2\sigma_n^2}{2^n} \le \frac{|\mathcal{D}|^2}{2^n} + \frac{2\sigma_n^2}{2^n} \le \frac{3\sigma_n^2}{2^n}.$$

Now consider pairs of elements from $\mathcal{R}$. First, leave out the pairs $(\mathsf{U}_1^s, \mathsf{U}_1^t)$, such that $s \neq t$ and $(\mathsf{T}^s, l^s) = (\mathsf{T}^t, l^t)$. These are now the special pairs and there are a total of $\sum_{k=1}^{p} \binom{l_k}{2}$ of such pairs. The total (over all s and t) probability of such pairs giving rise to a collision is given by Proposition 1 to be at most $\sigma_n^2/2^n$. Here we use the fact that queries are not pointless to note that $(\mathsf{C}_1^s, \mathsf{C}_2^s, \ldots, \mathsf{C}_{m^s-1}^s, \mathsf{C}_{m^s}^s) \neq (\mathsf{C}_1^t, \mathsf{C}_2^t, \ldots, \mathsf{C}_{m^t-1}^t, \mathsf{C}_{m^t}^t)$. This ensures that we can apply Proposition 1.

The probability of any non-special pair of elements from $\mathcal{R}$ colliding is either 0 or is equal to $1/2^n$. This analysis is again similar to that for the domain. The additional thing to note is that since the transcript is allowed, we have $(\mathsf{P}_i^s \oplus \mathsf{C}_i^s) \neq (\mathsf{P}_j^t \oplus \mathsf{C}_j^t)$ for $(s, i) \neq (j, t)$. In other words, the elements $(\mathsf{P}_i^s \oplus \mathsf{C}_i^s)$ are all distinct and so the probability of any two such elements colliding is zero.

There are at most $\binom{|\mathcal{R}|}{2}$ non-special pairs for $\mathcal{R}$. As in the case of domain elements, we can now show that the probability of a pair of elements in $\mathcal{R}$ colliding is at most $2\sigma_n^2/2^n$. (Note that the corresponding value for $\mathcal{D}$ has $3\sigma^2$. We get 2 here because there is only one type of special pairs from $\mathcal{R}$.)

Combining the domain and range collision probabilities, we obtain the probability of bad being set to true in NON to be at most $5\sigma_n^2/2^n$. Combining (17), (16) and (15), we have

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HCH}[\mathrm{Perm}(n)]}(A) \le \frac{6\sigma_n^2}{2^n}. \tag{20}$$

We now consider the variants of HCH. The game sequence for each of the variants can be constructed in a manner similar to that of HCH. The final collision analysis is based on the elements which enter the domain and range. Below, we provide the details of domain and range for the variants. In each case, given the elements of the domain and the range, a collision analysis similar to that of HCH provides the desired bounds on the probability of domain and range collisions.

6.5 Case of HCHp

In this case, the hash is computed with a separate random key α which is independent of the block cipher key. Compared to HCH, in this case, the quantity R^s is replaced by α in the polynomial expressions. Everything else remains unchanged.

$$
\begin{aligned}
\text{Elements in } \mathcal{D}: \quad & \mathsf{T}^s,\ \mathsf{R}^s \oplus \mathsf{bin}(l^s), \\
& \mathsf{M}_1^s = Q^s \oplus \mathsf{P}_1^s \oplus \mathsf{P}_2^s \alpha^{m^s-1} \oplus \cdots \oplus \mathsf{P}_{m^s-1}^s \alpha^2 \oplus \mathsf{M}_{m^s}^s \alpha, \\
& \mathsf{I}^s = \mathsf{M}_1^s \oplus \mathsf{U}_1^s = (x+1)Q^s \oplus \mathsf{Y}_1^s \oplus \mathsf{Y}_2^s \alpha^{m^s-1} \oplus \cdots \oplus \mathsf{Y}_{m^s}^s \alpha, \\
& \mathsf{S}_1^s, \mathsf{S}_2^s, \ldots, \mathsf{S}_{m^s-1}^s.
\end{aligned}
$$

$$
\begin{aligned}
\text{Elements in } \mathcal{R}: \quad & \mathsf{R}^s,\ \mathsf{Q}^s, \\
& \mathsf{U}_1^s = x\mathsf{Q}^s \oplus \mathsf{C}_1^s \oplus \mathsf{C}_2^s \alpha^{m^s-1} \oplus \cdots \oplus \mathsf{C}_{m^s-1}^s \alpha^2 \oplus \mathsf{U}_{m^s}^s \alpha, \\
& \mathsf{S}^s, \\
& (\mathsf{P}_2^s \oplus \mathsf{C}_2^s), \ldots, (\mathsf{P}_{m^s-1}^s \oplus \mathsf{C}_{m^s-1}^s), (\mathsf{M}_{m^s}^s \oplus \mathsf{U}_{m^s}^s).
\end{aligned}
$$

6.6 Case of HCHfp

All messages have the same fixed number of blocks denoted by m. In this case, the separate hashing key α replaces R^s in the polynomial expresssion. Also, Q is no longer computed and so $R \oplus \mathsf{bin}(l)$ does not enter the domain and Q does not enter the range. Instead, the quantity R plays the role of Q. The use of α introduces an extra source of randomness in the key. Since the length is also fixed, we can eliminate the block cipher call to produce Q.

$$
\begin{aligned}
\text{Elements in } \mathcal{D}: \quad & \mathsf{T}^s, \\
& M_1^s = R^s \oplus \mathsf{P}_1^s \oplus \mathsf{P}_2^s \alpha^{m-1} \oplus \cdots \oplus \mathsf{P}_{m-1}^s \alpha^2 \oplus \mathsf{M}_m^s \alpha, \\
& \mathsf{I}^s = \mathsf{M}_1^s \oplus \mathsf{U}_1^s = (x+1)\mathsf{R}^s \oplus \mathsf{Y}_1^s \oplus \mathsf{Y}_2^s \alpha^{m-1} \oplus \cdots \oplus \mathsf{Y}_m^s \alpha, \\
& \mathsf{S}_1^s, \mathsf{S}_2^s, \ldots, \mathsf{S}_{m-1}^s.
\end{aligned}
$$

$$
\begin{aligned}
\text{Elements in } \mathcal{R}: \quad & \mathsf{R}^s, \\
& \mathsf{U}_1^s = x\mathsf{R}^s \oplus \mathsf{C}_1^s \oplus \mathsf{C}_2^s \alpha^{m-1} \oplus \cdots \oplus \mathsf{C}_{m-1}^s \alpha^2 \oplus \mathsf{U}_m^s \alpha, \\
& \mathsf{S}^s, \\
& (\mathsf{P}_2^s \oplus \mathsf{C}_2^s), \ldots, (\mathsf{P}_{m-1}^s \oplus \mathsf{C}_{m-1}^s), (\mathsf{M}_m^s \oplus \mathsf{U}_m^s).
\end{aligned}
$$

7 Conclusion

In this paper, we have presented HCH, which is a new tweakable enciphering scheme. Our approach to the construction is based on the hash-encrypt-hash approach, where the encryption layer consists of a counter mode of encryption. The important features of HCH are the use of a single key, ability to encrypt arbitrary length messages and a quadratic security bound. To the best of our knowledge, HCH is the only construction to simultaneously achieve all the above three properties. We describe two variants of HCH – HCHp which can use pre-computation to speed up multiplication; and HCHfp which works for fixed length messages and can also utilize pre-computation. An important application of tweakable enciphering scheme is disk encryption. Both HCH and HCHfp provide a designer of a practical disk encryption algorithm with attractive alternatives.

References

1. http://csrc.nist.gov/CryptoToolkit/modes/.
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, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006.
3. Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.
4. Shai Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2004.
5. Shai Halevi. Invertible universal hashing and the TET encryption mode. Cryptology ePrint Archive, Report 2007/014, 2007. http://eprint.iacr.org/, to appear in the proceedings of Crypto 2007.
6. 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.
7. 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.
8. 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.
9. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
10. 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/.
11. David A. McGrew and John Viega. Arbitrary block length mode, 2004. http://grouper.ieee.org/groups/1619/email/pdf00005.pdf.
12. Moni Naor and Omer Reingold. A pseudo-random encryption mode. Manuscript available from www.wisdom.weizmann.ac.il/̃naor.
13. Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.
14. 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.

A The Games

Fig. 7. Game HCH1

Subroutine Ch-$\pi(X)$

$Y \xleftarrow{\$} \{0,1\}^n$; **if** $Y \in \mathit{Range}$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{Y \xleftarrow{\$} \overline{\mathit{Range}}}$; **endif**;

if $X \in \mathit{Domain}$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{Y \leftarrow \pi(X)}$; **endif**

$\pi(X) \leftarrow Y$; $\mathit{Domain} \leftarrow \mathit{Domain} \cup \{X\}$; $\mathit{Range} \leftarrow \mathit{Range} \cup \{Y\}$; **return**$(Y)$;

Subroutine Ch-$\pi^{-1}(Y)$

$X \xleftarrow{\$} \{0,1\}^n$; **if** $X \in \mathit{Domain}$, $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{X \xleftarrow{\$} \overline{\mathit{Domain}}}$; **endif**;

if $Y \in \mathit{Range}$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{X \leftarrow \pi^{-1}(Y)}$; **endif**;

$\pi(X) \leftarrow Y$; $\mathit{Domain} \leftarrow \mathit{Domain} \cup \{X\}$; $\mathit{Range} \leftarrow \mathit{Range} \cup \{Y\}$; **return**$(X)$;

Initialization:

for all $X \in \{0,1\}^n$ $\pi(X) = \mathsf{undef}$ **endfor**

$\mathsf{bad} = \mathsf{false}$

Respond to the s^{th} query as follows: (Recall $l^s = n(m^s - 1) + r^s$, with $1 \le r^s \le n$.)

Encipher query: $\mathsf{Enc}(T^s; P_1^s, P_2^s, \ldots P_{m^s}^s)$	Decipher query: $\mathsf{Dec}(T^s; C_1^s, C_2^s, \ldots, C_{m^s}^s)$
if $T^s = T^t$ for some $t < s$ **then** $R^s \leftarrow R^t$;	**if** $T^s = T^t$ for some $t < s$ **then** $R^s \leftarrow R^r$;
if $l^s = l^t$ **then** $Q^s \leftarrow Q^t$;	**if** $l^s = l^t$ **then** $Q^s \leftarrow Q^t$;
else $Q^s \leftarrow$ Ch-$\pi(R^s \oplus \mathsf{bin}(l^s))$;	**else** $Q^s \leftarrow$ Ch-$\pi(R^s \oplus \mathsf{bin}(l^s))$;
endif	**endif**
else	**else**
$R^s \leftarrow$ Ch-$\pi(T^s)$;	$R^s \leftarrow$ Ch-$\pi(T^s)$;
$Q^s \leftarrow$ Ch-$\pi(R^s \oplus \mathsf{bin}(l^s))$;	$Q^s \leftarrow$ Ch-$\pi(R^s \oplus \mathsf{bin}(l^s))$;
endif	**endif**
$M_m^s \leftarrow \mathsf{pad}_{n-r}(P_m)$;	$U_{m^s}^s \leftarrow \mathsf{pad}_{n-r^s}(C_{m^s})$;
$M_1^s \leftarrow H_{R^s,Q^s}(P_1^s, P_2^s, \ldots, P_{m-1}^s, M_m^s)$;	$U_1^s \leftarrow H_{R^s,xQ^s}(C_1^s, C_2^s, \ldots, C_{m^s-1}^s, U_{m^s}^s)$;
$U_1^s \leftarrow$ Ch-$\pi(M_1^s)$;	$M_1^s \leftarrow$ Ch-$\pi^{-1}(U_1^s)$
$I^s \leftarrow M_1^s \oplus U_1^s$;	$I \leftarrow M_1 \oplus U_1$;
$S^s \leftarrow$ Ch-$\pi(I^s)$;	$S^s \leftarrow$ Ch-$\pi(I^s)$;
for $i = 1$ to $m^s - 2$,	**for** $i = 1$ to $m^s - 2$,
$A_i^s \leftarrow S_i^s$; $B_i^s \leftarrow$ Ch-$\pi(A_i^s)$;	$A_i^s \leftarrow S_i^s$; $B_i^s \leftarrow$ Ch-$\pi(A_i^s)$;
$C_{i+1}^s \leftarrow P_{i+1}^s \oplus B_i^s$;	$P_{i+1}^s \leftarrow C_{i+1}^s \oplus B_i^s$;
end for	**end for**
$D_{m^s}^s \leftarrow M_{m^s}^s \oplus$ Ch-$\pi(S_{m^s-1}^s)$;	$V_{m^s}^s \leftarrow U_{m^s}^s \oplus$ Ch-$\pi(S_{m^s-1}^s)$;
$C_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(D_{m^s}^s)$; $U_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(C_{m^s}^s)$;	$P_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(V_{m^s}^s)$; $M_{m^s}^s \leftarrow \mathsf{pad}_{n-r^s}(P_{m^s})$;
$C_1^s \leftarrow H_{R^s,xQ^s}(U_1^s, C_2^s, \ldots, C_{m-1}^s, U_{m^s}^s)$;	$P_1^s \leftarrow H_{R^s,Q^s}(M_1^s, P_2^s, \ldots, P_{m^s-1}^s, M_{m^s}^s)$;
return $C_1^s \| C_2^s$	**return** $P_1^s \| P_2^s \| \ldots \| P_{m^s}^s$

Fig. 8. Game RAND2

Subroutine Check-Domain-Range(X, Y)

  if $X \in \mathit{Domain}$ **then** $\mathsf{bad} = \mathsf{true}$; **endif**

  if $Y \in \mathit{Range}$ **then** $\mathsf{bad} = \mathsf{true}$; **endif**

  $\mathit{Domain} = \mathit{Domain} \cup \{X\}$; $\mathit{Range} = \mathit{Range} \cup \{Y\}$;

INITIALIZATION

  $\mathit{Domain} = \mathit{Range} = \emptyset$; $\mathsf{bad} = \mathsf{false}$;

ENCIPHER QUERY: $\mathsf{Enc}(T^s; P_1^s, P_2^s, \ldots, P_{m^s}^s)$

if $T^s = T^t$ for some $r < s$ **then** $R^s \leftarrow R^t$;
  if $l^s = l^t$ **then** $Q^s \leftarrow Q^t$;
  else
    $Q^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
    Check-Domain-Range$(R^s \oplus \mathsf{bin}(l^s), Q^s)$;
  endif
else
  $R^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
  Check-Domain-Range(T^s, R^s);
  $Q^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
  Check-Domain-Range$(R^s \oplus \mathsf{bin}(l^s), Q^s)$;
endif

$M_m^s \leftarrow \mathsf{pad}_{n-r}(P_{m^s}^s)$;
$M_1^s \leftarrow H_{R^s,Q^s}(P_1^s, P_2^s, \ldots, P_{m^s-1}^s, M_m^s)$;
$S^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
for $i = 2$ to $m^s - 1$,
  $C_i^s \stackrel{\$}{\leftarrow} \{0,1\}^n$
  $Y_i^s \leftarrow C_i^s \oplus P_i^s$;
  Check-Domain-Range(S_{i-1}^s, Y_i^s);
end for
$D_{m^s}^s \stackrel{\$}{\leftarrow} \{0,1\}^n$; $Y_{m^s}^s \leftarrow D_{m^s}^s \oplus M_{m^s}^s$;
Check-Domain-Range$(S_{m^s-1}^s, Y_{m^s}^s)$;
$C_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(D_{m^s}^s)$;
$U_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(C_{m^s}^s)$;
$C_1^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
$U_1^s \leftarrow H_{R^s,xQ^s}(C_1^s, C_2^s, \ldots, C_{m-1}^s, U_m^s)$;
Check-Domain-Range(M_1^s, U_1^s);
Check-Domain-Range$(M_1^s \oplus U_1^s, S^s)$;
return $C_1^s || C_2^s || \ldots || C_{m^s}^s$

DECIPHER QUERY: $\mathsf{Dec}(T^s; C_1^s, C_2^s, \ldots, C_{m^s}^s)$

if $T^s = T^t$ for some $t < s$ **then** $R^s \leftarrow R^r$;
  if $l^s = l^t$ **then** $Q^s \leftarrow Q^t$;
  else
    $Q^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
    Check-Domain-Range$(R^s \oplus \mathsf{bin}(l^s), Q^s)$;
  endif
else
  $R^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
  Check-Domain-Range(T^s, R^s);
  $Q^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
  Check-Domain-Range$(R^s \oplus \mathsf{bin}(l^s), Q^s)$;
endif

$U_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(C_{m^s}^s)$;
$U_1^s \leftarrow H_{R^s,xQ^s}(C_1^s, C_2^s, \ldots, C_{m^s-1}^s, U_{m^s}^s)$;
$S^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
for $i = 2$ to $m^s - 1$,
  $P_i^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
  $Y_i^s \leftarrow P_i^s \oplus C_i^s$;
  Check-Domain-Range(S_{i-1}^s, Y_i^s);
end for
$V_{m^s}^s \stackrel{\$}{\leftarrow} \{0,1\}^n$; $Y_{m^s}^s \leftarrow V_{m^s}^s \oplus U_{m^s}^s$;
Check-Domain-Range$(S_{m^s-1}^s, Y_{m^s}^s)$;
$P_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(V_{m^s}^s)$;
$M_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(P_{m^s}^s)$;
$P_1^s \stackrel{\$}{\leftarrow} \{0,1\}^n$;
$M_1^s \leftarrow H_{R^s,Q^s}(P_1^s, P_2^s, \ldots, P_{m^s-1}^s, M_m^s)$;
Check-Domain-Range(M_1^s, U_1^s);
Check-Domain-Range$(M_1^s \oplus U_1^s, S^s)$;
return $P_1^s || P_2^s || \ldots || P_{m^s}^s$

Fig. 9. Game RAND3

Respond to the s^{th} adversary query as follows:

ENCIPHER QUERY $\mathsf{Enc}(T^s; P_1^s, P_2^s, \ldots, P_{m^s}^s)$

$ty^s = \mathsf{Enc}$; $C_1^s||C_2^s||\ldots||C_{m^s-1}^s||D_{m^s}^s \xleftarrow{\$} \{0,1\}^{nm^s}$;

$C_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(D_{m^s})$ **return** $C_1^s||C_2^s||\ldots||C_{m^s}^s$;

DECIPHER QUERY $\mathsf{Dec}(T^s; C_1^s, C_2^s, \ldots, C_{m^s}^s)$

$ty^s = \mathsf{Dec}$; $P_1^s||P_2^s||\ldots||P_{m_s-1}^s||V_{m^s}^s \xleftarrow{\$} \{0,1\}^{nm^s}$;

$P_{m^s}^s \leftarrow \mathsf{drop}_{n-r}(V_{m^s})$ **return** $P_1^s||P_2^s||\ldots||P_{m^s}^s$;

Finalization:

FIRST PHASE

if $T^s = T^t$ for some $t < s$ **then**
  $R^s \leftarrow R^t$;
  if $l^s = l^t$ **then**
    $Q^s \leftarrow Q^t$;
  else
    $Q^s \xleftarrow{\$} \{0,1\}^n$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{R^s \oplus \mathsf{bin}(l^s)\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Q^s\}$;
  endif
else
  $R^s \xleftarrow{\$} \{0,1\}^n$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{T^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{R^s\}$;
  $Q^s \xleftarrow{\$} \{0,1\}^n$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{R^s \oplus \mathsf{bin}(l^s)\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Q^s\}$;
endif

Case $ty^s = \mathsf{Enc}$:

$M_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(P_{m^s}^s)$;
$M_1^s \leftarrow H_{R^s,Q^s}(P_1^s, P_2^s, \ldots, P_{m^s-1}^s, M_m^s)$;
$S^s \xleftarrow{\$} \{0,1\}^n$;
for $i = 2$ to $m^s - 1$,
  $Y_i^s \leftarrow C_i^s \oplus P_i^s$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{i-1}^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Y_i^s\}$;
end for
$Y_{m^s}^s \leftarrow D_{m^s}^s \oplus M_{m^s}^s$
$\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{m^s-1}^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Y_{m^s}^s\}$;
$U_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(C_{m^s}^s)$;
$U_1^s \leftarrow H_{R^s,xQ^s}(C_1^s, C_2^s, \ldots, C_{m-1}^s, U_{m^s}^s)$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{U_1^s\}$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s \oplus U_1^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{S^s\}$;

Case $ty^s = \mathsf{Dec}$:

$U_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(C_{m^s}^s)$;
$U_1^s \leftarrow H_{R^s,xQ^s}(C_1^s, C_2^s, \ldots, C_{m^s-1}^s, U_{m^s}^s)$;
$S^s \xleftarrow{\$} \{0,1\}^n$;
for $i = 2$ to $m^s - 1$,
  $Y_i^s \leftarrow C_i^s \oplus P_i^s$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{i-1}^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Y_i^s\}$;
end for
$Y_{m^s}^s \leftarrow V_{m^s}^s \oplus U_{m^s}^s$
$\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{m^s-1}^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{Y_{m^s}^s\}$;
$M_{m^s}^s \leftarrow \mathsf{pad}_{n-r}(P_{m^s}^s)$;
$M_1^s \leftarrow H_{R^s,Q^s}(P_1^s, P_2^s, \ldots, P_{m^s-1}^s, M_m^s)$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{U_1^s\}$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s \oplus U_1^s\}$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{S^s\}$;

SECOND PHASE

if (some value occurs more than once in $\mathcal{D}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**;
if (some value occurs more than once in $\mathcal{R}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**.

B Patent and Other Issues

There are pending US patent applications on CMC, EME and XCB. The patent numbers are given in [5]. Below we put down some of our observations on the confusing issue of patents.

1. First, we would like to emphasize that we have not applied for any patent for HCH anywhere in the world and neither do we know of anyone who has done so.
2. The main components of HCH – block cipher; polynomial evaluation based universal hash function; and counter mode of operation – are old concepts. HCH puts these together in a particular fashion to construct a mode of operation. XCB, which is earlier to HCH and has a pending patent claim, also puts these components together in a different way to create a different mode of operation. However, as discussed in the main body of the paper, HCH offers several advantages over XCB which underlines the fact that the two modes of operations have *significant* differences.
3. On the other hand, depending on the nature of the patent applications, there is a legal possibility that the claims on CMC, EME and XCB cover aspects of the other constructions including PEP, HCTR, HCH and TET. Being a legal question, this can be settled only through legal procedures.