

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

Debrup Chakraborty and Palash Sarkar

Abstract—The notion of tweakable block ciphers was formally introduced by Liskov–Rivest–Wagner at Crypto 2002 (the 2002 Annual International Cryptology Conference). 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. 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 precomputation and makes one less block cipher call. This compares favorably to other hash-encrypt-hash-type constructions, supports better key agility and requires less key material.

Index Terms—Disk encryption, modes of operations, strong pseudorandom permutation, tweakable encryption.

I. 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, National Institute of Standards and Technology (NIST) of USA [1] has been running an open domain process to standardize modes of operations for achieving various functionalities. Currently, there are around 20 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.

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 precomputed 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. A Brief History of Known Constructions.

The first work to present a strong pseudorandom permutation is by Naor–Reingold [13]. They did not provide tweakable SPRPs because their work predates this notion.

The first construction of variable length 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

Manuscript received January 27, 2007; revised November 28, 2007. An abridged version of this paper appeared in *Lecture Notes in Computer Science* (Berlin, Germany: Springer-Verlag, 2006, vol. 4329, pp. 287–302).

D. Chakraborty is with the Computer Science Department, CINVESTAV-IPN, Mexico, D.F. 07360, Mexico (e-mail: debrup@cs.cinvestav.mx).

P. Sarkar is with the Applied Statistics Unit, Indian Statistical Institute, Kolkata 700 108, India (e-mail: palash@isical.ac.in).

Communicated by E. Okamoto, Associate Editor for Complexity and Cryptography.

Digital Object Identifier 10.1109/TIT.2008.917623

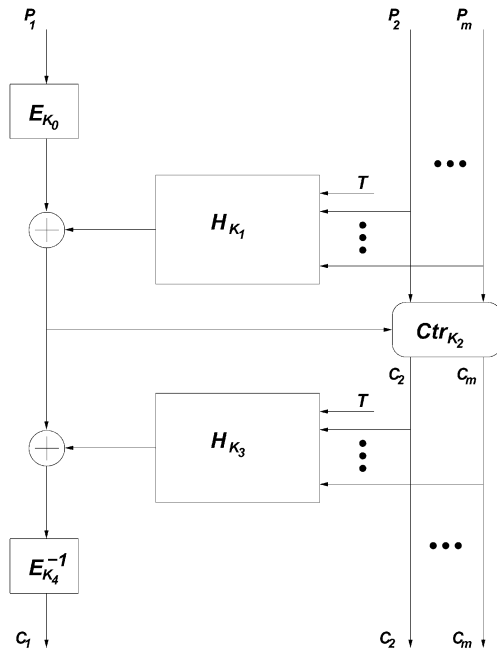


Fig. 1. Encryption using XCB. Here, the five keys K_0, \dots, K_4 are derived from a single key K using five block cipher invocations in the following manner: $K_0 = E_K(0^n)$, $K_1 = E_K(0^{n-1}||1)$, $K_2 = E_K(0^{n-2}||1||0)$, $K_3 = E_K(0^{n-2}||1^2)$, $K_4 = E_K(0^{n-3}||1||0^2)$. The keys K_0 ,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

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 (noninvertible) 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 was HCTR [15]. Another construction which has a structure similar to XCB was ABL [12]. 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 noninvertible and this can be considered to be closer to the Luby–Rackoff [9] approach, where noninvertible pseudorandom 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, because it is significantly slower than XCB.) The schematic diagrams for encryption using XCB and HCTR are given in Figs. 1 and 2. For the exact details of XCB and HCTR, we refer the reader to the respective papers.

XCB: From the single key K for the block cipher, XCB derives five keys K_0, \dots, 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

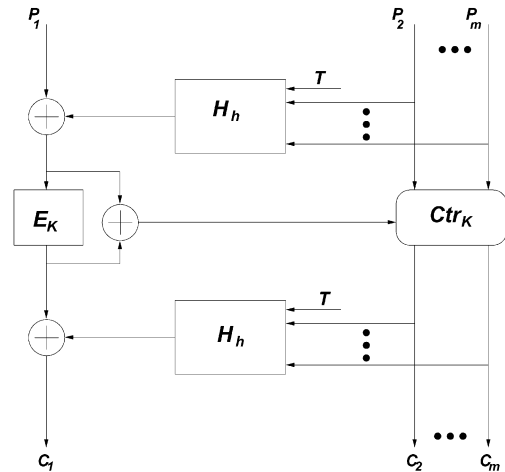


Fig. 2. Encryption using HCTR. Here, K is the key for the block cipher $E_K(\cdot)$ and h is the key for the universal hash function $H_h(\cdot)$.

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, \dots, K_4 . Then, this problem of key-length block-length mismatch 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.

In the original proposal [10], a security proof for XCB had been sketched but no concrete security bound was provided. In a recent work [11], appearing subsequently to the conference version of this paper, a quadratic security bound for XCB has been obtained.

HCTR: This work took place later than XCB and there are a few differences. It uses a single key for the block cipher and a single key for the both 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 $\sigma_n^2/2^n$.

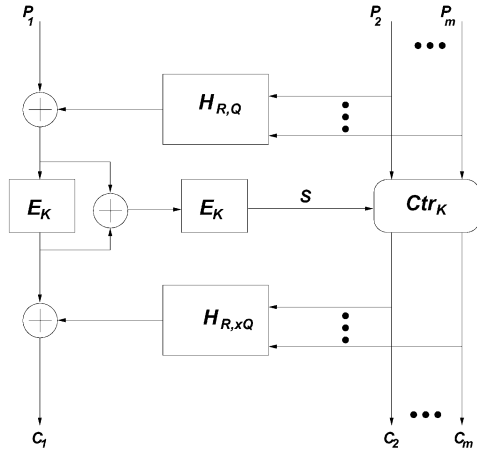


Fig. 3. Encryption using HCH. Here, $R = E_K(T)$ and $Q = E_K(R \oplus \text{bin}_n(l))$. In the figure, we show only full blocks and so M_m computed in Step 2 of the encryption algorithm given in Table II is equal to P_m .

B. 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 Fig. 3.

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 XOR of the input and the output of a block cipher invocation. In the combinatorial analysis, this turns out to be 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. 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 III. From the viewpoint of efficiency, the encrypt-mix-encrypt constructions (CMC, EME, and 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 III).

One drawback of HCH compared to XCB and TET is that it is not possible to use precomputation to speed up the polynomial hash computation. We note, however, that secure storage for precomputed tables can be a problem for multikey environments and hardware implementations. Nevertheless, we describe a simple variant of HCH called HCHp for which it is possible to utilize precomputation. 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 precomputation, 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.

II. SPECIFICATION OF HCH

We construct the tweakable enciphering scheme HCH from a block cipher $E : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ and call it $\text{HCH}[E]$. The key space of $\text{HCH}[E]$ is the 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 or equal to 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 $\text{pad}_t(X)$, we denote the string $X||0^t$ and by $\text{drop}_t(X)$, we denote the prefix of X obtained by dropping the last t bits of X . For $0 \leq i \leq 2^t - 1$, by $\text{bin}_t(i)$, we denote the t -bit binary representation of the integer i .

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

$$H_{R,Q}(A_1, \dots, A_m) = Q \oplus A_1 \oplus A_2 R^{m-1} \oplus \dots \oplus A_{m-1} R^2 \oplus A_m R. \quad (1)$$

The previous operations are over $GF(2^n)$, i.e., \oplus denotes addition over $GF(2^n)$ and terms of the form $A_i R^{m-i+1}$ denote the product $A_i(x)R^{m-i+1}(x) \bmod \tau(x)$. The final value of $H_{R,Q}(A_1, \dots, A_m)$ 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

$$\begin{aligned} &\text{If } B_1 = H_{R,Q}(A_1, A_2, \dots, A_m) \\ &\text{then } A_1 = H_{R,Q}(B_1, A_2, \dots, A_m). \end{aligned} \quad (2)$$

TABLE I
FORMATTING OF THE MESSAGE

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, \dots, P_m , with $|P_1| = \dots = |P_{m-1}| = n$ 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, \dots, C_m , with $|C_i| = |P_i|$ for $1 \leq i \leq m$.

TABLE II
ENCRYPTION AND DECRYPTION USING HCH. THE TWEAK IS T AND THE KEY IS K

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

In the conference version [2] of this paper, the function $H_{R,Q}$ was defined as $Q \oplus A_1 \oplus A_2 R \oplus \dots \oplus A_m R^{m-1}$. 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 orderings 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, \dots, S_m , where each S_i depends on S . Given such a sequence and a key K , we define the counter mode as follows:

$$\text{Ctr}_{K,S}(A_1, \dots, A_m) = (A_1 \oplus E_K(S_1), \dots, A_m \oplus E_K(S_m)). \quad (3)$$

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 nonnegative 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: for each S and for $i \neq j$ and for random and independent S^1 and S^2

$$\Pr[f_i(S^1) = f_j(S^2)] = \frac{1}{2^n} \left. \vphantom{\Pr} \right\}. \quad (4)$$

One simple way of defining f_i so as to satisfy (4) is to set $f_i(S) = S \oplus \text{bin}_n(i)$ as has been done for HCTR. Both 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 i th state of a maximal length linear feedback shift register (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_i 's from S such that the S_i 's satisfy (4) will serve our purpose.

Details on message parsing are given in Table I.

The complete encryption and decryption algorithm of HCH is given in Table II. A schematic diagram of encryption is given in Fig. 3.

Note: The two hashing layers are called with slightly different hashing keys. For the first hashing layer (in encryption), the key is (R, Q) , while for the second hashing layer, the key is (R, xQ) . The reason for this choice is to avoid certain kinds of collisions that arise in the security proof. To explain this, we consider the internal variables M_1, U_1 , and I in the encryption algorithm given in Table II (see also Table III). These variables can be written in terms of the plaintext and the ciphertext blocks in the following manner:

$$\begin{aligned} M_1 &= Q \oplus P_1 \oplus P_2 R^{m-1} \oplus \dots \oplus P_{m-1} R^2 \oplus M_m R \\ U_1 &= xQ \oplus C_1 \oplus C_2 R^{m-1} \oplus \dots \oplus C_{m-1} R^2 \oplus U_m R \\ I &= M_1 \oplus U_1 \\ &= (x \oplus 1)Q \oplus Y_1 \oplus Y_2 R^{m-1} \oplus \dots \oplus Y_{m-1} R^2 \oplus Y_m R \end{aligned}$$

where $Y_i = P_i \oplus C_i$, $1 \leq i \leq m - 1$ and $Y_m = M_m \oplus U_m$. If the same hashing key (R, Q) were used for both layers, then I would not depend on Q and consequently also would not depend on the length l of the message. The final analysis of the security proof allows the adversary to control both the plaintext and the ciphertext. Such an adversary can easily find two plaintext-ciphertext pairs of different lengths for which the I -values are equal. This would invalidate the collision analysis in the security proof. It turns out that using the slightly different hashing key (R, xQ) for the second hashing layer avoids this problem.

A. Messages of Length n

We have excluded the case of $l = n$ from the previous 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.

TABLE III

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 II-A. FOR PEP, WE ASSUME $m \geq 3$. FOR TET, m' IS THE NUMBER OF FULL BLOCKS AND i 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: AUXILIARY 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	$\sigma_n^2/2^n$	$u((m - 1)[M]+2[BC])$ $+(m + 2)[BC]$ $+2m'[M]+1[I]$	$3m[R]+2m[W]$	2[BCK]	$\geq n$	3	1	partial
XCB	III	$\sigma_n^2/2^n$	$(m + 6)[BC]$ $+2(m + 1)[M]$	$2m[R]+m[W]$	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

(The quantity Q is still required.) Thus, this case needs to be tackled separately. The extension is the following:

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 previously. Decryption is simple. This requires a total of three block cipher calls (two to produce Q and one for producing C_1).

The security of the previous 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 Q 's are different and so the probability of a collision is $1/2^n$. The values of Q will be the same only for those blocks which have the same tweak. However, in this case, the corresponding plaintext must be different (as otherwise the adversary has provided the same query twice) and hence the probability of a collision is zero.

III. DISCUSSION AND COMPARISON

The differences between HCH, XCB, and HCTR were mentioned earlier. All three can handle arbitrary length strings.

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 multikey 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 $\tau = \text{PRF}_{K_1}(0, i)$ (where $\text{PRF}_{K_1}()$ is a pseudorandom function with key K_1) and $\sigma = 1 \oplus \tau \oplus \dots \oplus \tau^m$ successively for $i = 0, 1, \dots$ until a nonzero 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 III, we present a comparison of HCH with the previous algorithms. We compare it with CMC, EME, and EME*, which are encrypt-mix-encrypt-type constructions. XCB and HCTR are also included, because they are of the hash-counter-hash type. We do not include ABL, because 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 that can tackle arbitrary length strings. (HCH can be modified to also tackle strings of length n ; see Section II-A.) As discussed earlier, TET is not interesting for encrypting variable length strings. EME* has a quadratic security bound, and uses one block cipher key plus two n -bit strings as auxiliary key material. XCB uses a single key with some restrictions. 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 holds and using the hash-counter-hash approach will improve.

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 online, i.e., the ciphertext cannot be produced without reading (or processing) the entire message. This property is natural to be expected from a *strong* pseudorandom permutation, because such a cryptographic 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 behavior of a multipass 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 $2m$ 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 III. From the previous 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 may vary quite a lot.

Parallelism: Some of the constructions in Table III 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 $GF(2^n)$ multiplications by roughly a factor of two.

Arbitrary Length Tweaks: CMC, EME, EME*, and HCH support n -bit tweaks. On the other hand, XCB and 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 pseudo-random 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, because in this case, sector addresses are used as tweaks.

A. Precomputation

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

- **Block cipher key schedule.** If the block cipher key is fixed, then it is possible to precompute 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 precomputed key schedules. Thus, a construction that uses more block cipher keys requires more storage space. Let us consider XCB as an example. The five keys that it requires do not depend on the message or the tweak and hence these keys can be precomputed. Out of these keys, three are block cipher keys and to obtain good speed one may need to precompute and store the key schedule for all three of these keys. Similarly, TET uses two block cipher keys and will require to store the key schedule for both keys. In contrast to this, EME*, HCTR, and HCH use a single block cipher key and hence the storage space for the precomputed key schedule will be smaller.
- **Table for computing polynomial hash.** If one of the operands is fixed, then precomputation can be used to speed up finite field multiplication [14]. XCB, HCTR, and TET can use such precomputation to speed up the computation of the polynomial hash. On the other hand,

TABLE IV
 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 $E_{K,\alpha}^T(P_1, \dots, P_m)$	Algorithm $D_{K,\alpha}^T(C_1, \dots, C_m)$
1. $R = E_K(T)$; $Q = E_K(R \oplus \text{bin}_n(t))$;	1. $R = E_K(T)$; $Q = E_K(R \oplus \text{bin}_n(t))$;
2. $M_m = \text{pad}_{n-r}(P_m)$;	2. $U_m = \text{pad}_{n-r}(C_m)$;
3. $M_1 = H_{\alpha,Q}(P_1, \dots, P_{m-1}, M_m)$;	3. $U_1 = H_{\alpha,Q}(C_1, \dots, C_{m-1}, U_m)$;
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. $(C_2, \dots, C_{m-1}, D_m)$ $= \text{Ctr}_{K,S}(P_2, \dots, P_{m-1}, M_m)$;	5. $(P_2, \dots, P_{m-1}, V_m)$ $= \text{Ctr}_{K,S}(C_2, \dots, C_{m-1}, U_m)$;
6. $C_m = \text{drop}_{n-r}(D_m)$; $U_m = \text{pad}_{n-r}(C_m)$;	6. $P_m = \text{drop}_{n-r}(V_m)$; $M_m = \text{pad}_{n-r}(P_m)$;
7. $C_1 = H_{\alpha,Q}(U_1, C_2, \dots, C_{m-1}, U_m)$;	7. $P_1 = H_{\alpha,Q}(M_1, P_2, \dots, P_{m-1}, M_m)$;
8. return (C_1, \dots, C_m) .	8. return (P_1, \dots, P_m) .

because HCH evaluates the polynomial at a value that depends on the tweak and the length, it is not possible to use precomputation for speeding up polynomial arithmetic.

We note, however, that the general issue of precomputation 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 multikey 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.

B. HCHp: A Variant Supporting Precomputation

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 precomputation 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 Table IV.

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 . Because α is a fixed quantity, a table can be precomputed 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 are provided in Section VI-E.

IV. 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 precomputed. (After precomputing the keys, depending on the available secure storage space, one may or may not choose to precompute 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, because the value of m is now fixed, the values of τ and σ^{-1} can be precomputed and stored. Also, TET requires the encryption of the message length which can be precomputed and stored. This reduces the number of block cipher calls by one and increases the storage requirement by three n -bit strings.

A. 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. Next, we describe a fixed length variant HCHfp, which is a variant for fixed length messages and it supports precomputation. This is given in Table V

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 precomputed 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 VI-F.

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 precomputed if required. This may be desirable for some applications. The collision analysis of this variant is almost the same as that of HCHp and hence is not provided.

Note: The collision analysis for HCHfp will fail if variable length messages are allowed. To see this, consider the internal variable M_1 in Table V. This can be written in terms of P_1, \dots, P_{m-1}, M_m as follows:

$$M_1 = R \oplus P_1 \oplus P_2 \alpha^{m-1} \oplus \dots \oplus P_{m-1} \alpha^2 \oplus M_m \alpha.$$

The quantity R is independent of the length of the message. Now consider two messages $(P_1, 0^n)$ and $(P_1, 0^n, 0^n)$ with the same tweak T . Here, P_1 is any n -bit block. Let us denote the quantities corresponding to the two messages by superscripts 1 and 2. For example, the corresponding values of M_1 will be denoted by M_1^1 and M_1^2 . Because the tweak is the same for both messages, we have $R^1 = R^2$ which we simply denote by R .

TABLE V
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 LENGTH OF THE MESSAGE AND HENCE THE NUMBER OF BLOCKS m IS FIXED

Algorithm $E_{K,\alpha}^T(P_1, \dots, P_m)$	Algorithm $D_{K,\alpha}^T(C_1, \dots, C_m)$
1. $R = E_K(T)$;	1. $R = E_K(T)$;
2. $M_m = \text{pad}_{n-r}(P_m)$;	2. $U_m = \text{pad}_{n-r}(C_m)$;
3. $M_1 = H_{\alpha,R}(P_1, \dots, P_{m-1}, M_m)$;	3. $U_1 = H_{\alpha,xR}(C_1, \dots, C_{m-1}, U_m)$;
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. $(C_2, \dots, C_{m-1}, D_m)$ $= \text{Ctr}_{K,S}(P_2, \dots, P_{m-1}, M_m)$;	5. $(P_2, \dots, P_{m-1}, V_m)$ $= \text{Ctr}_{K,S}(C_2, \dots, C_{m-1}, U_m)$;
6. $C_m = \text{drop}_{n-r}(D_m)$; $U_m = \text{pad}_{n-r}(C_m)$;	6. $P_m = \text{drop}_{n-r}(V_m)$; $M_m = \text{pad}_{n-r}(P_m)$;
7. $C_1 = H_{\alpha,xR}(U_1, C_2, \dots, C_{m-1}, U_m)$;	7. $P_1 = H_{\alpha,R}(M_1, P_2, \dots, P_{m-1}, M_m)$;
8. return (C_1, \dots, C_m) .	8. return (P_1, \dots, P_m) .

TABLE VI
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; BCK: BLOCK CIPHER KEY; AK: AUXILIARY n -BIT STRING (INCLUDING POLYNOMIAL HASH KEYS)

Mode	CMC	EME*	XCB	TET	HCH	HCHfp
[BC]	$2m + 1$	$2m + 1 + m/n$	$m + 1$	$m + 1$	$m + 3$	$m + 2$
[M]	–	–	$2(m + 3)$	$2m$	$2(m - 1)$	$2(m - 1)$
[BCK]	1	1	3	2	1	1
[AK]	–	2	2	3	–	1
precomp (mult table)	–	–	yes	yes	no	yes

TABLE VII
EFFICIENCY OF KEY CHANGE. FOR TET, THE VALUE OF i DEPENDS ONLY ON K (BECAUSE THE NUMBER OF BLOCKS m IS FIXED). I: INVERSION

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

It is easy to see that $M_1^1 = R \oplus P_1 = M_1^2$. Consequently, $U_1^1 = U_1^2$, $I^1 = I^2$ and $S^1 = S^2$. This means that the counter gets initialized to the same value for both messages. As a result, $C_2^1 = 0^n \oplus E_K(S^1) = 0^n \oplus E_K(S^2) = C_2^2$. In other words, the second ciphertext blocks corresponding to the two messages are equal. Consequently, HCHfp is not a pseudorandom permutation for variable length messages. Basically, to tackle variable length messages, we need to incorporate the length directly into the encryption process. This is done in HCH and HCHfp.

B. Comparison Among Fixed Length Variants

The comparison for fixed length inputs is given in Table VI. 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.

Let us now consider the constructions in the hash-encrypt-hash approaches. Table VI 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, because it has a cubic 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 precomputation to speed up the hash computations, while HCHfp uses a block cipher key and an n -bit hash key and can use precomputation 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 precomputation is used, then XCB, TET, and HCHfp have similar efficiencies.

Key Agility: Table VII 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 VI in the sense that if less is precomputed, then the computation costs in Table VI will increase.

In terms of key agility, CMC, EME*, and HCH are the best, because 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.

V. SECURITY OF HCH AND THE VARIANTS

A. 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 $K \in \mathcal{K}$, $E(K, \cdot)$ is a permutation. We write $E_K(\cdot)$ instead of $E(K, \cdot)$.

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 $X \stackrel{\$}{\leftarrow} \mathcal{S}$, we will denote the event of choosing X uniformly at random from the set \mathcal{S} .

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

$$\text{Adv}_E^{\pm\text{PRP}}(A) = \left| \Pr \left[K \stackrel{\$}{\leftarrow} \mathcal{K} : A^{E_K(\cdot), E_K^{-1}(\cdot)} \Rightarrow 1 \right] - \Pr \left[\pi \stackrel{\$}{\leftarrow} \text{Perm}(n) : A^{\pi(\cdot), \pi^{-1}(\cdot)} \Rightarrow 1 \right] \right|.$$

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 will write $\mathbf{E}_K^T(\cdot)$ instead of $\mathbf{E}(K, T, \cdot)$. 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 $\text{Perm}^T(\mathcal{M})$ denote the set of all functions $\pi : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\pi(\mathcal{T}, \cdot)$ is a length preserving permutation. Such a $\pi \in \text{Perm}^T(\mathcal{M})$ 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} , its inverse from a random tweak indexed permutation, and its inverse in the following manner:

$$\text{Adv}_E^{\pm\text{PRP}}(A) = \left| \Pr \left[K \stackrel{\$}{\leftarrow} \mathcal{K} : A^{\mathbf{E}_K(\cdot, \cdot), \mathbf{E}_K^{-1}(\cdot, \cdot)} \Rightarrow 1 \right] - \Pr \left[\pi \stackrel{\$}{\leftarrow} \text{Perm}^T(\mathcal{M}) : A^{\pi(\cdot, \cdot), \pi^{-1}(\cdot, \cdot)} \Rightarrow 1 \right] \right|.$$

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 the 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, \dots) 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 i th 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 $\text{Adv}_{\mathcal{E}}^{\pm\text{xxx}}(A)$ has been defined, where \mathcal{E} is either a block cipher or a tweakable enciphering scheme. $\text{Adv}_{\mathcal{E}}^{\pm\text{xxx}}(\rho)$ denotes the maximal value of $\text{Adv}_{\mathcal{E}}^{\pm\text{xxx}}(A)$ 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 $\sigma_n (n \geq 1)$.

The notation $\text{HCH}[E]$ 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 $\text{HCH}[E]$ is secure if E is secure. The notation $\text{HCH}[\text{Perm}(n)]$ denotes a tweakable enciphering scheme obtained by plugging in a random permutation from $\text{Perm}(n)$ into the structure of HCH. For an adversary attacking $\text{HCH}[\text{Perm}(n)]$, 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 $\text{HCH}[\text{Perm}(n)]$ by obtaining an upper bound on $\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{PRP}}(\sigma_n)$. The upper bound is obtained in terms of n and σ_n . For a fixed block cipher E , we bound $\text{Adv}_{\text{HCH}[E]}^{\pm\text{PRP}}(\sigma_n, t)$ in terms of $\text{Adv}_E^{\pm\text{PRP}}(\sigma_n, t')$, where $t' = t + O(\sigma_n)$. We will use the notation \mathbf{E}_π as a shorthand for $\text{HCH}[\text{Perm}(n)]$ and \mathbf{D}_π will denote the inverse of \mathbf{E}_π . Thus, the notation $A^{\mathbf{E}_\pi, \mathbf{D}_\pi}$ will denote an adversary interacting with the oracles \mathbf{E}_π and \mathbf{D}_π .

B. Statement of the Results

The following theorem specifies the security of HCH.

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

$$\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{PRP}}(\sigma_n) \leq \frac{7\sigma_n^2}{2^n}. \quad (5)$$

$$\text{Adv}_{[E]}^{\pm\text{PRP}}(\sigma_n, t) \leq \frac{7\sigma_n^2}{2^n} + \text{Adv}_E^{\pm\text{PRP}}(\sigma_n, t') \quad (6)$$

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

The previous result and its proof are similar to the previous work (see, for example, [6], [7], and [3]). As mentioned in [6], (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:

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

where

$$X = \left(\Pr \left[K \stackrel{\$}{\leftarrow} \mathcal{K} : A^{\mathbf{E}_K(\cdot, \cdot), \mathbf{E}_K^{-1}(\cdot, \cdot)} \Rightarrow 1 \right] - \Pr \left[\pi \stackrel{\$}{\leftarrow} \text{Perm}(n) : A^{\mathbf{E}_\pi, \mathbf{D}_\pi} \Rightarrow 1 \right] \right).$$

The quantity X represents an adversary's advantage in distinguishing $\text{HCH}[E]$ from $\text{HCH}[\pi]$, where π is a randomly chosen permutation from $\text{Perm}(n)$. Clearly, such an adversary A can also distinguish E from a random permutation and hence $X \leq \text{Adv}_E^{\pm\text{prp}}(A)$. 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:

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

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

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

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

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 $\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{rnd}}(\sigma_n)$. This proof is provided in Section VI, where we show [see (17)] that for any adversary having query complexity σ_n , we have

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

Using this and (7), we obtain

$$\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{prp}}(\sigma_n) \leq \frac{7\sigma_n^2}{2^n}.$$

The upper bound on $\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{prp}}(\sigma_n)$ is obtained by 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.

VI. 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 $\text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm\text{rnd}}(q, \sigma_n)$. 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}_π and \mathbf{D}_π) and when interacting with an oracle which returns random strings on any input.

We model the adversary's interaction with the oracles \mathbf{E}_π and \mathbf{D}_π as a game. In the usual game, which we call HCH1, the adversary submits queries to \mathbf{E}_π and \mathbf{D}_π and gets appropriate answers. Starting from this game, we modify it in successive steps to obtain games where the adversary is provided with 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.

A. 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 the 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 probability 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 do not affect the adversary's view in any manner.

TABLE VIII
ELEMENTS OF \mathcal{D} AND \mathcal{R} FOR HCH

Elements in \mathcal{D} :	$T^s, R^s \oplus \text{bin}(l^s),$ $M_1^s = Q^s \oplus P_1^s \oplus P_2^s (R^s)^{m^s-1} \oplus \dots \oplus P_{m^s-1}^s (R^s)^2 \oplus M_{m^s}^s R^s,$ $I^s = M_1^s \oplus U_1^s = (x+1)Q^s \oplus Y_1^s \oplus Y_2^s (R^s)^{m^s-1} \oplus \dots \oplus Y_{m^s}^s R^s,$ $S_1^s, S_2^s, \dots, S_{m^s-1}^s.$
Elements in \mathcal{R} :	$R^s, Q^s,$ $U_1^s = xQ^s \oplus C_1^s \oplus C_2^s (R^s)^{m^s-1} \oplus \dots \oplus C_{m^s-1}^s (R^s)^2 \oplus U_{m^s}^s R^s,$ $S^s,$ $(P_2^s \oplus C_2^s), \dots, (P_{m^s-1}^s \oplus C_{m^s-1}^s), (M_{m^s}^s \oplus U_{m^s}^s).$

TABLE IX
ELEMENTS OF \mathcal{D} AND \mathcal{R} FOR HCHP

Elements in \mathcal{D} :	$T^s, R^s \oplus \text{bin}(l^s),$ $M_1^s = Q^s \oplus P_1^s \oplus P_2^s \alpha^{m^s-1} \oplus \dots \oplus P_{m^s-1}^s \alpha^2 \oplus M_{m^s}^s \alpha,$ $I^s = M_1^s \oplus U_1^s = (x+1)Q^s \oplus Y_1^s \oplus Y_2^s \alpha^{m^s-1} \oplus \dots \oplus Y_{m^s}^s \alpha,$ $S_1^s, S_2^s, \dots, S_{m^s-1}^s.$
Elements in \mathcal{R} :	$R^s, Q^s,$ $U_1^s = xQ^s \oplus C_1^s \oplus C_2^s \alpha^{m^s-1} \oplus \dots \oplus C_{m^s-1}^s \alpha^2 \oplus U_{m^s}^s \alpha,$ $S^s,$ $(P_2^s \oplus C_2^s), \dots, (P_{m^s-1}^s \oplus C_{m^s-1}^s), (M_{m^s}^s \oplus U_{m^s}^s).$

TABLE X
ELEMENTS OF \mathcal{D} AND \mathcal{R} FOR HCHFP

Elements in \mathcal{D} :	$T^s,$ $M_1^s = R^s \oplus P_1^s \oplus P_2^s \alpha^{m-1} \oplus \dots \oplus P_{m-1}^s \alpha^2 \oplus M_m^s \alpha,$ $I^s = M_1^s \oplus U_1^s = (x+1)R^s \oplus Y_1^s \oplus Y_2^s \alpha^{m-1} \oplus \dots \oplus Y_m^s \alpha,$ $S_1^s, S_2^s, \dots, S_{m-1}^s.$
Elements in \mathcal{R} :	$R^s,$ $U_1^s = xR^s \oplus C_1^s \oplus C_2^s \alpha^{m-1} \oplus \dots \oplus C_{m-1}^s \alpha^2 \oplus U_m^s \alpha,$ $S^s,$ $(P_2^s \oplus C_2^s), \dots, (P_{m-1}^s \oplus C_{m-1}^s), (M_m^s \oplus U_m^s).$

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 the outputs of the counter mode are distinct over all the queries. Such transcripts are called allowed transcripts. To obtain the game NON (for noninteractive), 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.

B. The Games

By the 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 pseudocodes for the different games are given in Tables XI–XIII.

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}_π and \mathbf{D}_π , where π is a randomly chosen permutation from $\text{Perm}(n)$. 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

TABLE XI
GAME HCH1

Subroutine $\text{Ch-}\pi(X)$ $Y \stackrel{\$}{\leftarrow} \{0, 1\}^n$; if $Y \in \text{Range}$ then $\text{bad} \leftarrow \text{true}$; $Y \stackrel{\\$}{\leftarrow} \overline{\text{Range}}$; endif ; if $X \in \text{Domain}$ then $\text{bad} \leftarrow \text{true}$; $Y \leftarrow \pi(X)$; endif $\pi(X) \leftarrow Y$; $\text{Domain} \leftarrow \text{Domain} \cup \{X\}$; $\text{Range} \leftarrow \text{Range} \cup \{Y\}$; return (Y); Subroutine $\text{Ch-}\pi^{-1}(Y)$ $X \stackrel{\$}{\leftarrow} \{0, 1\}^n$; if $X \in \text{Domain}$, $\text{bad} \leftarrow \text{true}$; $X \stackrel{\\$}{\leftarrow} \overline{\text{Domain}}$; endif ; if $Y \in \text{Range}$ then $\text{bad} \leftarrow \text{true}$; $X \leftarrow \pi^{-1}(Y)$; endif ; $\pi(X) \leftarrow Y$; $\text{Domain} \leftarrow \text{Domain} \cup \{X\}$; $\text{Range} \leftarrow \text{Range} \cup \{Y\}$; return (X); <u>Initialization:</u> for all $X \in \{0, 1\}^n$ $\pi(X) = \text{undef}$ endfor $\text{bad} = \text{false}$; $\text{Domain} = \text{Range} = \emptyset$;	
Respond to the s^{th} query as follows: (Recall $l^s = n(m^s - 1) + r^s$, with $1 \leq r^s \leq n$.)	
<u>Encipher query:</u> $\text{Enc}(T^s; P_1^s, P_2^s, \dots, P_{m^s}^s)$ 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 $Q^s \leftarrow \text{Ch-}\pi(R^s \oplus \text{bin}(l^s))$; endif else $R^s \leftarrow \text{Ch-}\pi(T^s)$; $Q^s \leftarrow \text{Ch-}\pi(R^s \oplus \text{bin}(l^s))$; endif $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s)$; $M_1^s \leftarrow H_{R^s, Q^s}(P_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s)$; $U_1^s \leftarrow \text{Ch-}\pi(M_1^s)$; $I^s \leftarrow M_1^s \oplus U_1^s$; $S^s \leftarrow \text{Ch-}\pi(I^s)$; for $i = 1$ to $m^s - 2$, $A_i^s \leftarrow S_i^s$; $B_i^s \leftarrow \text{Ch-}\pi(A_i^s)$; $C_{i+1}^s \leftarrow P_{i+1}^s \oplus B_i^s$; end for $D_{m^s}^s \leftarrow M_{m^s}^s \oplus \text{Ch-}\pi(S_{m^s-1}^s)$; $C_{m^s}^s \leftarrow \text{drop}_{n-r^s}(D_{m^s}^s)$; $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s)$; $C_1^s \leftarrow H_{R^s, Q^s}(U_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s)$; return $C_1^s C_2^s \dots C_{m^s}^s$	<u>Decipher query:</u> $\text{Dec}(T^s; C_1^s, C_2^s, \dots, C_{m^s}^s)$ 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 $Q^s \leftarrow \text{Ch-}\pi(R^s \oplus \text{bin}(l^s))$; endif else $R^s \leftarrow \text{Ch-}\pi(T^s)$; $Q^s \leftarrow \text{Ch-}\pi(R^s \oplus \text{bin}(l^s))$; endif $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s)$; $U_1^s \leftarrow H_{R^s, Q^s}(C_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s)$; $M_1^s \leftarrow \text{Ch-}\pi^{-1}(U_1^s)$; $I^s \leftarrow M_1^s \oplus U_1^s$; $S^s \leftarrow \text{Ch-}\pi(I^s)$; for $i = 1$ to $m^s - 2$, $A_i^s \leftarrow S_i^s$; $B_i^s \leftarrow \text{Ch-}\pi(A_i^s)$; $P_{i+1}^s \leftarrow C_{i+1}^s \oplus B_i^s$; end for $V_{m^s}^s \leftarrow U_{m^s}^s \oplus \text{Ch-}\pi(S_{m^s-1}^s)$; $P_{m^s}^s \leftarrow \text{drop}_{n-r^s}(V_{m^s}^s)$; $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s)$; $P_1^s \leftarrow H_{R^s, Q^s}(M_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s)$; return $P_1^s P_2^s \dots P_{m^s}^s$

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 $\overline{\text{Domain}}$ and $\overline{\text{Range}}$ are the complements of Domain and Range relative to $\{0, 1\}^n$. The game HCH1 is shown in Table XI. The figure shows the subroutines $\text{Ch-}\pi$ and $\text{Ch-}\pi^{-1}$, the initialization steps, and how the game responds to the encipher/decipher query of the adversary. The i^{th} query of the adversary depends on its previous queries, the responses to those queries, and some coins of the adversary.

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

$$\Pr[A^{\text{E}\pi, \text{D}\pi} \Rightarrow 1] = \Pr[A^{\text{HCH1}} \Rightarrow 1]. \quad (8)$$

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 $\text{Ch-}\pi$) and X (in $\text{Ch-}\pi^{-1}$). Thus, the games HCH1 and RAND1 are identical apart from what happens when the bad flag is set; so

$$|\Pr[A^{\text{HCH1}} \Rightarrow 1] - \Pr[A^{\text{RAND1}} \Rightarrow 1]| \leq \Pr[A^{\text{RAND1}} \text{sets bad}]. \quad (9)$$

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

TABLE XII
GAME RAND2

Subroutine Check-Domain-Range(X, Y) if $X \in \text{Domain}$ then bad = true; endif if $Y \in \text{Range}$ then bad = true; endif $\text{Domain} = \text{Domain} \cup \{X\}$; $\text{Range} = \text{Range} \cup \{Y\}$; INITIALIZATION $\text{Domain} = \text{Range} = \emptyset$; bad = false;	
ENCRYPTER QUERY: Enc($T^s; P_1^s, P_2^s, \dots, P_{m^s}^s$) 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 $Q^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range($R^s \oplus \text{bin}(l^s), Q^s$); endif else $R^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range(T^s, R^s); $Q^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range($R^s \oplus \text{bin}(l^s), Q^s$); endif $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s)$; $M_1^s \leftarrow H_{R^s, Q^s}(P_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s)$; $S^s \xleftarrow{\$} \{0, 1\}^n$; for $i = 2$ to $m^s - 1$, $C_i^s \xleftarrow{\$} \{0, 1\}^n$ $Y_i^s \leftarrow C_i^s \oplus P_i^s$; Check-Domain-Range(S_{i-1}^s, Y_i^s); end for $D_{m^s}^s \xleftarrow{\$} \{0, 1\}^n$; $Y_{m^s}^s \leftarrow D_{m^s}^s \oplus M_{m^s}^s$; Check-Domain-Range($S_{m^s-1}^s, Y_{m^s}^s$); $C_{m^s}^s \leftarrow \text{drop}_{n-r^s}(D_{m^s}^s)$; $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s)$; $C_1^s \xleftarrow{\$} \{0, 1\}^n$; $U_1^s \leftarrow H_{R^s, Q^s}(C_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s)$; Check-Domain-Range(M_1^s, U_1^s); Check-Domain-Range($M_1^s \oplus U_1^s, S^s$); return $C_1^s C_2^s \dots C_{m^s}^s$	DECRYPTER QUERY: Dec($T^s; C_1^s, C_2^s, \dots, C_{m^s}^s$) 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 $Q^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range($R^s \oplus \text{bin}(l^s), Q^s$); endif else $R^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range(T^s, R^s); $Q^s \xleftarrow{\$} \{0, 1\}^n$; Check-Domain-Range($R^s \oplus \text{bin}(l^s), Q^s$); endif $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s)$; $U_1^s \leftarrow H_{R^s, Q^s}(C_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s)$; $S^s \xleftarrow{\$} \{0, 1\}^n$; for $i = 2$ to $m^s - 1$, $P_i^s \xleftarrow{\$} \{0, 1\}^n$; $Y_i^s \leftarrow P_i^s \oplus C_i^s$; Check-Domain-Range(S_{i-1}^s, Y_i^s); end for $V_{m^s}^s \xleftarrow{\$} \{0, 1\}^n$; $Y_{m^s}^s \leftarrow V_{m^s}^s \oplus U_{m^s}^s$; Check-Domain-Range($S_{m^s-1}^s, Y_{m^s}^s$); $P_{m^s}^s \leftarrow \text{drop}_{n-r^s}(V_{m^s}^s)$; $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s)$; $P_1^s \xleftarrow{\$} \{0, 1\}^n$; $M_1^s \leftarrow H_{R^s, Q^s}(P_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s)$; Check-Domain-Range(M_1^s, U_1^s); Check-Domain-Range($M_1^s \oplus U_1^s, S^s$); return $P_1^s P_2^s \dots P_{m^s}^s$

serted into Range; also, if $X \in \text{Domain}$ or $Y \in \text{Range}$, 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, because for each such change the adversary obtains a random n -bit string both before and after the change. Thus

$$\Pr[A^{\text{RAND1}} \Rightarrow 1] = \Pr[A^{\text{RAND2}} \Rightarrow 1] \quad (10)$$

and also

$$\Pr[A^{\text{RAND1}} \text{sets bad}] = \Pr[A^{\text{RAND2}} \text{sets bad}]. \quad (11)$$

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

$$\Pr[A^{\text{RAND2}} \Rightarrow 1] = \Pr[A^{\$(\cdot, \cdot), \$(\cdot, \cdot)} \Rightarrow 1]. \quad (12)$$

Now, from (8)–(12), we get

$$\begin{aligned} & \text{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm \text{rand}}(A) \\ &= |\Pr[A^{\text{E}_\pi, \text{D}_\pi} \Rightarrow 1] - \Pr[A^{\$(\cdot, \cdot), \$(\cdot, \cdot)} \Rightarrow 1]| \\ &= |\Pr[A^{\text{HCH1}} \Rightarrow 1] - \Pr[A^{\text{RAND2}} \Rightarrow 1]| \end{aligned}$$

$$\begin{aligned}
&= |\Pr[A^{\text{HCH1}} \Rightarrow 1] - \Pr[A^{\text{RAND1}} \Rightarrow 1]| \\
&\leq \Pr[A^{\text{RAND1}}_{\text{sets bad}}] \\
&= \Pr[A^{\text{RAND2}}_{\text{sets bad}}]. \tag{13}
\end{aligned}$$

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

Game RAND3: 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 the 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 Table XIII. RAND3 sets bad in exactly the same conditions in which RAND2 sets bad, hence

$$\Pr[A^{\text{RAND2}}_{\text{sets bad}}] = \Pr[A^{\text{RAND3}}_{\text{sets bad}}]. \tag{14}$$

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

$$\begin{aligned}
Y_i^s &= C_i^s \oplus P_i^s, \text{ when } 1 \leq i \leq m^s - 1; \\
&= D_{m^s}^s \oplus M_{m^s}^s, \text{ when } i = m^s \text{ and } ty = \text{Enc}; \\
&= U_{m^s}^s \oplus V_{m^s}^s, \text{ when } i = m^s \text{ and } ty = \text{Dec}.
\end{aligned}$$

For $2 \leq i \leq m^s$, the variable Y_i^s enters the range set and it is always an n -bit random quantity. When $2 \leq i \leq m^s - 1$, then for an 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, $D_{m^s}^s$ (resp. $V_{m^s}^s$) is a randomly chosen n -bit string. Thus, for $(s, i) \neq (t, j)$, $\Pr[Y_i^s = Y_j^t] = \frac{1}{2^n}$. 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 $\binom{|\mathcal{R}|}{2} / 2^n \leq \sigma_n^2 / 2^n$. (Note that $|\mathcal{R}| \leq \sigma_n$, 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

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

Our next task is to upper bound $\Pr[(A^{\text{RAND3}}_{\text{sets bad}}) | \overline{\mathbf{X}}]$. 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 behavior under this condition.

In the previous games, for an encipher query, the adversary specified the tweak and the plaintext; and for a decipher query, it 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, because it is notationally simpler. The effects of both methods are the same, because we require that Y_i^s 's are distinct for all s and $i = 2, \dots, m^s$.

We do this by modifying the game RAND3 into a new game NON (noninteractive). NON depends on a fixed transcript $\text{tr} = (\mathbf{ty}, \mathbf{T}, \mathbf{P}, \mathbf{C}, \mathbf{E})$ with $\mathbf{ty} = (ty^1, ty^2, \dots, ty^q)$, $\mathbf{T} = (T^1, T^2, \dots, T^q)$, $\mathbf{P} = (P^1, P^2, \dots, P^q)$, $\mathbf{C} = (C^1, C^2, \dots, C^q)$, and $\mathbf{E} = (E^1, E^2, \dots, E^q)$, where $ty^s \in \{\text{Enc}, \text{Dec}\}$, $T^s \in \{0, 1\}^n$, $P^s = (P_1^s, P_2^s, \dots, P_{m^s}^s)$, $C^s = (C_1^s, C_2^s, \dots, C_{m^s}^s)$ and each E^s is a string of length $(n - r^s)$ such that $Y_{m^s}^s = (P_{m^s}^s \oplus C_{m^s}^s) || E^s$. 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 T^s , 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

$$\Pr[A^{\text{RAND3}}_{\text{sets bad}}] \leq \Pr[A^{\text{NON}}_{\text{sets bad}}] + \frac{\sigma_n^2}{2^n}. \tag{15}$$

C. Useful Result

The analysis of NON requires a result, which we state and prove here. Let $\mathcal{L} = (\mathcal{L}^1, \dots, \mathcal{L}^q)$ be a list of vectors, with $\mathcal{L}^{i_1} \neq \mathcal{L}^{i_2}$ for $i_1 \neq i_2$ and where $\mathcal{L}^i = (L_1^i, \dots, L_{m^i}^i)$ 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

$$\begin{aligned}
P_{i,j}(X) &= (L_1^i \oplus L_1^j) \oplus (L_2^i \oplus L_2^j)X \oplus (L_3^i \oplus L_3^j)X^2 \\
&\quad \oplus \dots \oplus (L_{m^i}^i \oplus L_{m^i}^j)X^{m^i-1}.
\end{aligned}$$

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

$$\mathbf{e} = \bigvee_{\{i,j\} \in \mathcal{S}} \mathbf{e}_{i,j}. \tag{16}$$

TABLE XIII
GAME RAND3

Respond to the s^{th} adversary query as follows: ENCRYPT QUERY $\text{Enc}(T^s; P_1^s, P_2^s, \dots, P_{m^s}^s)$ $ty^s = \text{Enc}; C_1^s C_2^s \dots C_{m^s-1}^s D_{m^s}^s \xleftarrow{\$} \{0, 1\}^{nm^s};$ $C_{m^s}^s \leftarrow \text{drop}_{n-r^s}(D_{m^s}^s)$ return $C_1^s C_2^s \dots C_{m^s}^s;$ DECRYPT QUERY $\text{Dec}(T^s; C_1^s, C_2^s, \dots, C_{m^s}^s)$ $ty^s = \text{Dec}; P_1^s P_2^s \dots P_{m^s-1}^s V_{m^s}^s \xleftarrow{\$} \{0, 1\}^{nm^s};$ $P_{m^s}^s \leftarrow \text{drop}_{n-r^s}(V_{m^s}^s)$ return $P_1^s P_2^s \dots P_{m^s}^s;$	
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 $Q^s \xleftarrow{\$} \{0, 1\}^n;$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{R^s \oplus \text{bin}(l^s)\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{Q^s\};$ endif else $R^s \xleftarrow{\$} \{0, 1\}^n;$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{T^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{R^s\};$ $Q^s \xleftarrow{\$} \{0, 1\}^n;$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{R^s \oplus \text{bin}(l^s)\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{Q^s\};$ endif	
Case $ty^s = \text{Enc}$: $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s);$ $M_1^s \leftarrow H_{R^s, Q^s}(P_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s);$ $S^s \xleftarrow{\$} \{0, 1\}^n;$ 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 $Y_{m^s}^s \leftarrow D_{m^s}^s \oplus M_{m^s}^s$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{m^s-1}^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{Y_{m^s}^s\};$ $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s);$ $U_1^s \leftarrow H_{R^s, xQ^s}(C_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s);$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{U_1^s\};$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s \oplus U_1^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{S^s\};$	Case $ty^s = \text{Dec}$: $U_{m^s}^s \leftarrow \text{pad}_{n-r^s}(C_{m^s}^s);$ $U_1^s \leftarrow H_{R^s, xQ^s}(C_1^s, C_2^s, \dots, C_{m^s-1}^s, U_{m^s}^s);$ $S^s \xleftarrow{\$} \{0, 1\}^n;$ 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 $Y_{m^s}^s \leftarrow V_{m^s}^s \oplus U_{m^s}^s$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_{m^s-1}^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{Y_{m^s}^s\};$ $M_{m^s}^s \leftarrow \text{pad}_{n-r^s}(P_{m^s}^s);$ $M_1^s \leftarrow H_{R^s, Q^s}(P_1^s, P_2^s, \dots, P_{m^s-1}^s, M_{m^s}^s);$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{U_1^s\};$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{M_1^s \oplus U_1^s\}; \mathcal{R} \leftarrow \mathcal{R} \cup \{S^s\};$
SECOND PHASE if (some value occurs more than once in \mathcal{D}) then bad = true endif ; if (some value occurs more than once in \mathcal{R}) then bad = true endif .	

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

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

Proof: Suppose the different possible distinct lengths of the \mathcal{L}_i 's are l_1, \dots, 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 nonzero polynomial, because 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 $\{1, \dots, p\}$. 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 $(l_k - 1)/2^n$. Because there are $\binom{t_k}{2}$ pairs in \mathcal{S} corresponding to length l_k , the contribu-

tion of all tuples of length l_k to e is at most $\binom{t_k}{2} (l_k - 1)/2^n$. Hence

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

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

D. 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 that enter \mathcal{D} and \mathcal{R} are shown in Table VIII.

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, \dots, 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 variables of the pair get the same value. We identify the following two types of pairs of variables and call them special pairs.

- 1) (M_1^s, M_1^t) , such that $s \neq t$ and $(T^s, l^s) = (T^t, l^t)$.
- 2) (l^s, l^t) , such that $s \neq t$ and $(T^s, l^s) = (T^t, l^t)$.

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 VI-C to be at most $\sigma_n^2/2^n$. Note that because queries are not pointless, we will have $(P_1^s, P_2^s, \dots, P_{m^s-1}^s, P_{m^s}^s) \neq (P_1^t, P_2^t, \dots, P_{m^t-1}^t, P_{m^t}^t)$, when $(T^s, l^s) = (T^t, l^t)$. Also, in an allowed transcript, all the Y_i^s 's (for $1 \leq i \leq m^s - 1$) are distinct. They ensure that we can apply Proposition 1 to the aforementioned two cases.

For any nonspecial 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 the 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, $\Pr[S_i^s = S_j^s] = 0$ for $i \neq j$. As S^s and S^t are independent and random n -bit strings, so $\Pr[S_i^s = S_j^t] = 1/2^n$, for $s \neq t$. See (4).

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

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

Now consider pairs of elements from \mathcal{R} . First, leave out the pairs (U_1^s, U_1^t) , such that $s \neq t$ and $(T^s, l^s) = (T^t, l^t)$. 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 $(C_1^s, C_2^s, \dots, C_{m^s-1}^s, C_{m^s}^s) \neq (C_1^t, C_2^t, \dots, C_{m^t-1}^t, C_{m^t}^t)$. This ensures that we can apply Proposition 1.

The probability of any nonspecial 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 $(P_i^s \oplus C_i^s) \neq (P_j^t \oplus C_j^t)$ for $(s, i) \neq (j, t)$. In other words, the elements $(P_i^s \oplus C_i^s)$ are all distinct and so the probability of any two such elements colliding is zero.

There are at most $\binom{|\mathcal{R}|}{2}$ nonspecial 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 $3\sigma_n^2$. 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 (13)–(15), we have

$$\mathbf{Adv}_{\text{HCH}[\text{Perm}(n)]}^{\pm \text{rnd}}(A) \leq \frac{6\sigma_n^2}{2^n}. \quad (17)$$

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. In the following, 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.

E. 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. The structure of the elements in \mathcal{D} and \mathcal{R} is shown in Table IX.

F. 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 expression. Also, Q is no longer computed and so $R \oplus \text{bin}(l)$ 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. Because the length is also fixed, we can eliminate the block cipher call to produce Q . The structure of the elements in \mathcal{D} and \mathcal{R} is shown in Table X.

VII. 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 those three properties. We describe two variants of HCH-HCHp that can use precomputation to speed up multiplication, and HCHfp that works for fixed length messages and can also utilize precomputation. 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.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their comments which helped in improving this paper.

REFERENCES

- [1] National Institute of Standards and Technology, Cryptographic Toolkit, [Online]. Available: <http://csrc.nist.gov/CryptoToolkit/modes/>
- [2] D. Chakraborty and P. Sarkar, "HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach," in *Lecture Notes in Computer Science*, R. Barua and T. Lange, Eds. Berlin, Germany: Springer-Verlag, 2006, vol. 4329, pp. 287–302.
- [3] D. Chakraborty and P. Sarkar, "A new mode of encryption providing a tweakable strong pseudo-random permutation," in *Lecture Notes in Computer Science*, J. B. Robshaw, Ed. Berlin, Germany: Springer-Verlag, 2006, vol. 4047, pp. 293–309.
- [4] S. Halevi, "Extending EME to handle arbitrary-length messages with associated data," in *Lecture Notes in Computer Science*, A. Canteaut and K. Viswanathan, Eds. Berlin, Germany: Springer-Verlag, 2004, vol. 3348, pp. 315–327.
- [5] S. Halevi, "Invertible universal hashing and the TET encryption mode," in *Lecture Notes in Computer Science*, A. Menezes, Ed. Berlin, Germany: Springer-Verlag, 2007, vol. 4622, pp. 412–429.
- [6] S. Halevi and P. Rogaway, "A tweakable enciphering mode," in *Lecture Notes in Computer Science*, D. Boneh, Ed. Berlin, Germany: Springer-Verlag, 2003, vol. 2729, pp. 482–499.
- [7] S. Halevi and P. Rogaway, "A parallelizable enciphering mode," in *Lecture Notes in Computer Science*, T. Okamoto, Ed. Berlin, Germany: Springer-Verlag, 2004, vol. 2964, pp. 292–304.
- [8] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable block ciphers," in *Lecture Notes in Computer Science*, M. Yung, Ed. Berlin, Germany: Springer-Verlag, 2002, vol. 2442, pp. 31–46.
- [9] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Comput.*, vol. 17, no. 2, pp. 373–386, 1988.
- [10] D. A. McGrew and S. R. Fluhrer, "The extended codebook (XCB) mode of operation," Rep. 2004/278, 2004 [Online]. Available: <http://eprint.iacr.org/>, Cryptology ePrint Archive
- [11] D. A. McGrew and S. R. Fluhrer, "The security of the extended codebook (XCB) mode of operation," in *Lecture Notes in Computer Science*, C. M. Adams, A. Miri, and M. J. Wiener, Eds. Berlin, Germany: Springer-Verlag, 2007, vol. 4876, Selected Areas in Cryptography, pp. 311–327.
- [12] D. A. McGrew and J. Viega, "Arbitrary block length mode," 2004 [Online]. Available: <http://grouper.ieee.org/groups/1619/email/pdf00005.pdf>
- [13] M. Naor and O. Reingold, "A pseudo-random encryption mode," [Online]. Available: www.wisdom.weizmann.ac.il/naor
- [14] V. Shoup, "On fast and provably secure message authentication based on universal hashing," in *Lecture Notes in Computer Science*, N. Kobitz, Ed. Berlin, Germany: Springer-Verlag, 1996, vol. 1109, pp. 313–328.
- [15] P. Wang, D. Feng, and W. Wu, "A variable-input-length enciphering mode," in *Lecture Notes in Computer Science*, D. Feng, D. Lin, and M. Yung, Eds. Berlin, Germany: Springer-Verlag, 2005, vol. 3822, pp. 175–188.