Efficient and Provably Secure Ciphers for Storage Device Block Level Encryption

Yuliang Zheng
SIS Department, UNC Charlotte
yzheng@uncc.edu

Yongge Wang
SIS Department, UNC Charlotte
yonwang@uncc.edu

ABSTRACT

Block ciphers generally have fixed and relatively small input length. Thus they are often used in some mode of operations (e.g., ECB, CBC, CFB, and CTR) that enables the encryption of longer messages. Unfortunately, all these modes of operation reveal some information on their inputs or on relationships between different inputs. As an example, in the CBC mode, encrypting two messages with an identical prefix will result in identical initial blocks in the ciphertexts. Due to the well-known birthday attack and the small input length, the CBC mode becomes less secure as the number of data blocks to be encrypted increases. This leads to a challenging task, namely to design schemes for storage device block or sector level data encryption that are efficient and do not have the disadvantages mentioned above. In this paper, we propose an efficient cipher whose data/cipher blocks can be specified flexibly to match the length of a block unit for current and foreseeable future storage devices. We show that our encryption scheme is provably secure under the assumption that the underlying one-way hash function is a random function.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption—*standards*

General Terms

Algorithms, performance, security, theory

Keywords

Symmetric cipher, hash function, storage device encryption

1. INTRODUCTION

Recent years have seen an increasing demand for properly designed encryption schemes that could be used to encrypt storage devices at the block (sector) level. Since data are read from/written to storage devices per storage block unit (e.g., one kilobyte), chains between ciphertext should be limited within storage blocks to maintain good performance. Thus, storage device block encryptions are generally done by using block ciphers (e.g., AES) in some mode of operation such as CBC (note that the term "block" in "block cipher" has a different meaning from the term "block" in "storage block"). CBC encryption becomes insecure once $2^{n/2}$ blocks have been encrypted, in the sense that at this point partial information about the message begins to leak, due to the well-known birthday attacks (see, e.g., [1]), where n is the cipher-block length. Furthermore, this is true for many other common modes of operations too. Thus direct use of a 128-bit block size cipher (like AES-128) usually enables one to safely encrypt no more than 2^{64} 128-bit data blocks, which is not acceptable in many other applications such as terabyte disk encryptions and global scale storage systems. Thus better ciphers for storage device block-level encryptions are desired. Based on Luby and Rackoff's [6] seminal work, we propose two provable secure encryption schemes with arbitrary cipher-block lengths which are ideal for storage device (indeed for general block device) encryptions.

Motivated by the Feistel network structure in Data Encryption Standard DES, Luby and Rackoff [6] showed a general method (LR-construction) for constructing pseudo-random permutations from pseudo-random functions. Their construction consists of four rounds (or three rounds for weaker security requirements) of Feistel permutations, each round involves an application of a (different) pseudo-random function. The elegance of LR-construction is their provable security. Recently, Naor and Reingold [8] provided a better understanding of the LR-construction and improved the constructions by replacing the first and last rounds in the LR-construction with appropriately designed pair-wise independent permutations based on universal hash functions. Note that while Naor and Reingold's constrution reduces the required number of pseudo-random functions, it does *not* reduce the required number of rounds.

In this paper, we design efficient, arbitrary input length, and provable secure ciphers based on LR-construction by replacing the pseudo-random functions in the LR-construction with random functions constructed from efficient hash functions. Naor and Reingold's improvement is also applicable to our ciphers. That is, if preferred, the first and the last rounds in our ciphers could also be replaced with properly designed permutations. Note that our constructions have several practical advantages over other constructions. Firstly, our encryption schemes are only based on fast hash functions such as SHA-1 or SHA-256 which are extensively used in industry. Secondly, the code size for our scheme is very small (the code for hash function is enough). Thirdly, our schemes are very efficient. Fourthly, since the cipher-block for our scheme is larger, our scheme is more immune against the birthday attacks. Thus our schemes are more suitable for storage device encryptions. Due to their efficiency and small code size, our schemes are also suitable for small device or embedded device storage encryptions.

Note that for Naor and Reingold's improvement, further code for universal hash functions is required, which could be expensive in some small devices.

Instead of constructing a cipher of larger block size, Liskov, Rivest, and Wagner [5] and Halevi and Rogaway [4] proposed modes of operations to address the similar problems. In particular, Halevi and Rogaway [4] proposed tweakable the following tweakable enciphering mode for storage block level encrypitons: derive the ciphertext c of a plaintext m as $c = \mathcal{E}_K^t(m)$ where t is the harddist position that m (c indeed) is stored. Specifically, they proposed a enciphering mode CBC-Mask-CBC CMC. CMC starts with a block cipher $\mathcal{E} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and turns it into an enciphering scheme CMC$[\mathcal{E}] : \mathcal{K}' \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ where $\mathcal{T} = \{0,1\}^n$ contains the position information and $\mathcal{M}$ contains strings with any number (at least two) of n-bit blocks. Note that the tweak construction in [4] is essentially based on the tweak construction in Liskov, Rivest, and Wagner [5].

2. LR-CONSTRUCTION WITH HASH FUNCTIONS

In this section, we present our first cipher which is a straightforward application of hash functions to the LR-construction. Let n be a positive integer and $H(\cdot)$ be a random function mapping arbitrary length binary strings to n-bit binary strings.

Let L and R denote the left and right half of a $2n$-bit string $L||R$, K be any fixed-length binary string, $[i]_8$ be a 8-bit block that is equal to the number i expressed using a binary representation, and etc_i be a binary string (in practice etc_i, could be some known information, for example, the block device block index together with the cipher round index in binary representations). Then, for each $i \le 3$, we can define the function $f_{i,K}(\cdot)$ (mapping from $2n$-bit binary strings to $2n$-bit binary strings) by letting

$$f_{i,K}(L||R) = R||(L \oplus H(R||[i]_8||K||etc_i)),$$

that is, the right half of the argument appears unchanged as the left half of the result and the right half of the result is equal to $L \oplus H(R||[i]_8||K||etc_i)$. This corresponds to one round of DES and is shown in Figure 1.

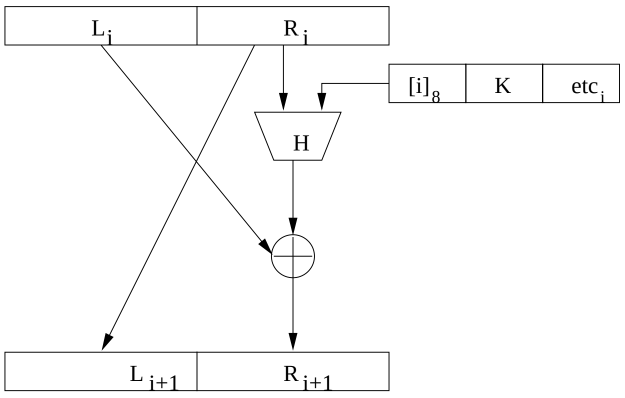

Figure 1: One round $f_{i,K}(L||R)$ of the encryption scheme $\mathcal{E}_K(\cdot)$

The encryption scheme $\mathcal{E}_K(\cdot)$ is defined as the permutation

$$\mathcal{E}_K(L||R) = f_{3,K}(f_{2,K}(f_{1,K}(f_{0,K}(L||R))))$$

where $L||R$ is a $2n$-bit binary string and K is the secret key. For a $2n$-bit string $L||R$, let

$$f_{i,K}^{-1}(L||R) = (R \oplus H(L||[i]_8||K||etc_i))||L.$$

Then for a $2n$-bit ciphertext $L_4||R_4$, the plaintext is computed by the following operation:

$$\mathcal{E}_K^{-1}(L_4||R_4) = f_{0,K}^{-1}(f_{1,K}^{-1}(f_{2,K}^{-1}(f_{3,K}^{-1}(L_4||R_4))))$$

Pseudo-random bit generators were introduced by Blum and Micali [3] and Yao [11]. We refer to section 4 of [6] for definitions of a pseudo-random number (or bit) generator (PRNG), of a pseudo-random function generator (PRFG), and of a pseudo-random permutation generator (PRPG). We will also use standard terms such as (non)adaptive chosen plaintext attacks, (non)adaptive chosen ciphertext attacks, security against (non)adaptive chosen plaintext and chosen ciphertext attacks. For these definitions, we refer to [8, 6]. In the following, we show that if the random function H is appropriately defined, then the encryption scheme $\mathcal{E}_K(\cdot)$ is provable secure against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.

LEMMA 2.1. *Let m, n and l be three integers such that $lm = n$ and $l << n$. Assume that $\mathcal{R}$ is a pseudo-random function mapping arbitrary length input binary strings x to m-bit binary strings $\mathcal{R}(x)$. Define the function $H(\cdot)$ by letting*

$$H(x) = \mathcal{R}([0]_8||x)||\mathcal{R}([1]_8||x)||\cdots||\mathcal{R}([l-1]_8||x)$$

where $[0]_8$, $[1]_8$, ..., and $[l-1]_8$ are 8-bit blocks that are equal to the numbers $0, 1, \ldots$, and $l-1$ expressed using a binary representation, respectively. Then $H(\cdot)$ is a pseudo-random generator.

PROOF. The proof for this lemma is similar to those proofs in Blum and Micali [3] and Yao [11]. The details will be given in the final version of this paper. Q.E.D. □

THEOREM 2.2. *Let $H(\cdot)$ be a pseudo-random generator constructed from a pseudo-random function $\mathcal{R}$ as in Lemma 2.1. Then the encryption scheme $\mathcal{E}_K(\cdot)$ is provable secure against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.*

PROOF. Assume that $\mathcal{R}$ is a pseudo-random function and H is a pseudo-random generator constructed from $\mathcal{R}$ as in Lemma 2.1. For $i \le 3$, let $g_i(\cdot)$ be defined by letting

$$g_i(x) = H(x||[i]_8||K||etc_i)$$

where K is the secret key. Then it can be shown that the pseudo-random function set $\{g_0, g_1, g_2, g_3\}$ is indistinguishable from a set of four pseudo-randomly chosen functions mapping arbitrary length binary strings to n-bit binary strings. Thus the theorem follows from the results in Luby and Rackoff [6]. The details of the proof will be given in the final version of this paper. Q.E.D. □

If security against non-adaptive attacks is sufficient in some scenarios, then one round in $\mathcal{E}_K(\cdot)$ could be reduced. That is, let

$$\mathcal{E}'_K(L||R) = f_{2,K}(f_{1,K}(f_{0,K}(L||R))).$$

THEOREM 2.3. *Let $H(\cdot)$ be a pseudo-random generator constructed from a pseudo-random function $\mathcal{R}$ as in Lemma 2.1. Then the encryption scheme $\mathcal{E}'_K(\cdot)$ is provable secure against non-adaptive chosen-plaintext and non-adaptive chosen-ciphertext attacks.*

PROOF. The proof is similar to the proof of Theorem 2.2 using the results from Maurer [7]. Q.E.D. □

If it is preferred, Naor and Reingold's [8] technique could be used to replace the functions $f_{3,K}$ and $f_{0,K}$ in $\mathcal{E}_K(\cdot)$ with two pair-wise independent permutations. That is, let $h_1(\cdot)$ and $h_2(\cdot)$ be two pair-wise independent permutations on the set of $2n$-bit binary strings. Then

$$\mathcal{E}''_K(L||R) = h_2^{-1}(f_{1,K}(f_{0,K}(h_1(L||R))))$$

is a provable secure encryption scheme against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.

The encryption scheme $\mathcal{E}_K(\cdot)$ has several advantages for storage device block-level data encryptions. In practice, we can replace the random function $\mathcal{R}$ in Lemma 2.1 with appropriately chosen hash functions such as SHA-1 or SHA-256 [10]. In the following, by replacing $\mathcal{R}$ with SHA-256 (that is, the pseudo-random bit generator H is constructed from SHA-256 as in Lemma 2.1), we give an exemplar analysis of using the cipher $\mathcal{E}_K(\cdot)$ to encrypt storage devices at sector (block) level.

Assume that the block unit of a storage device is 1024 bytes. Then $\mathcal{E}_K(\cdot)$ can be used with the parameter $n = 4096$ (note that 1024 bytes are $8162 = 2 \times 4096$ bits). That is, each storage device block unit is one cipher-block of $\mathcal{E}_K(\cdot)$ and is divided into two parts L and R, each of 4096 bits. Since the output of SHA-256 is 256 bits, the function H consists of $l = 4096/256 = 16$ times applications of SHA-256. The most computation intensive part of the scheme $\mathcal{E}_K(\cdot)$ is the evaluation of the functions H. Thus the computation complexity of the encryption of one block unit of the storage device is approximately $4 \times 16 = 64$ applications of SHA-256 on $(4096 + 8 + k + |etc_i|)$-bit inputs, where k is the bit-length of the secret key and $|etc_i|$ is the bit-length of the additional input etc_i. Assume that 128-bit key is used and etc_i is the storage device block number plus the cipher round index in binary representations. Then 32 bits are enough for etc_i if the volume of the storage device is about one terabyte. Thus the input to each application of SHA-256 is 4264 bits. Since the block size for SHA-256 is 512 bits, for inputs of 4264 bits, SHA-256 needs a total of 9 times of the basic four-step SHA-256 operation. In a summary, if SHA-256 is used as the underlying pseudo-random function and 128 bits secret key is used, then a total of $9 \times 64 = 576$ times of the basic four-step SHA-256 operations are needed to encrypt a 1024 bytes storage device block unit. Similar analysis shows that if SHA-512 (SHA-512 has a block size of 1024 bits) is used as the underlying pseudo-random function and 256-bit secret key is used, then a total of $5 \times 32 = 160$ times of the basic four-step SHA-512 operation are needed to encrypt a 1024 bytes storage device block unit.

Naor and Reingold's [8] method can be applied to the scheme $\mathcal{E}_K(\cdot)$ by replacing the first and last rounds with pair-wise independent permutations. The new scheme is still provable secure against adaptive chosen plaintext and adaptive chosen ciphertext attacks. The speed of the new scheme is two times faster than the speed of the original scheme $\mathcal{E}_K(\cdot)$ if we ignore the time required for the two permutation operations. However, in several applications, we may prefer not to use Naor and Reingold's method in the scheme $\mathcal{E}_K(\cdot)$ if the universal hash function-based permutations are not preferred in the application.

The above analysis shows that the encryption scheme $\mathcal{E}_K(\cdot)$ could be implemented with very good performance. In addition, the scheme $\mathcal{E}_K(\cdot)$ has a few advantages over AES or other fixed block size ciphers.

- The scheme $\mathcal{E}_K(\cdot)$ is provable secure assuming that the underlying hash function is a pseudo-random function.

- If a cipher's block size is n-bit and CBC mode is used, then the scheme becomes insecure when $2^{n/2}$ blocks have been encrypted. Since the encryption scheme $\mathcal{E}_K(\cdot)$'s cipher-block size is equal to the storage device block unit size (which is generally 1024 bytes), $\mathcal{E}_K(\cdot)$ achieves the best security bound against birthday attacks that an encryption scheme for storage device can have. Naor and Reingold [8, 9] constructed an encryption scheme for storage devices based on ciphers such as AES. As they have pointed out, however, their scheme is also vulnerable to a birthday attack on the size of the original block size.

- The scheme $\mathcal{E}_K(\cdot)$ has a very simple internal structure and the main code needed for implementation is the code for hash functions. Thus this encryption scheme may be preferred in small embedded device storage encryptions.

3. EFFICIENT HASH BASED ENCRYPTION FOR STORAGE SECTORS: HESS

In the encryption scheme $\mathcal{E}_K(\cdot)$ from last section, each hash function is applied to the entire string "$R||[i]_8||K||etc_i$". For storage devices with larger block unit, this string could be very longer. Since most hash functions such as SHS has a relatively small block size, the speed of the encryption scheme $\mathcal{E}_K(\cdot)$ could be significantly improved if the input to the hash functions are relatively short. In this section, we introduce an efficient hash based encryption scheme for storage sectors: HESS, which is an improvement of the scheme $\mathcal{E}_K(\cdot)$.

Let n, m and l be three non-negative integers such that $lm = n$ and $l \leq 64$. Let H be a pseudo-random function mapping arbitrary length binary strings to m-bit binary strings. For each $i = 0, 1, 2,$ and 3, we define a function $g_{i,K}(\cdot)$ (Figure 2) mapping n-bit binary strings to n-bit binary strings as follows:

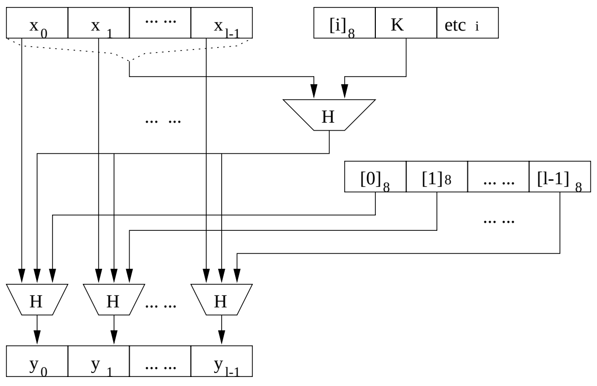

Figure 2: The function $g_{i,K}(\cdot)$

1. Let $x = x_0 \ldots x_{l-1}$ where x is an n-bit binary string, x_j's are m-bit binary strings for $j = 0, \ldots, l-1$.

2. Let $z' = H(x||[i]_8||K||etc_i)$, where $[i]_8$ is a 8-bit block that is equal to the number i expressed using a binary representation, K is any fixed-length binary string, and etc_i is a binary string (in practice, etc_i could be some known information, for example, the block device block index together with the cipher round index in binary representations), and let z be the leftmost $m - 8$ bits of z'.

3. Let $y_j = H(x_j||z||[j]_8)$ for $j = 0, \ldots, l-1$.

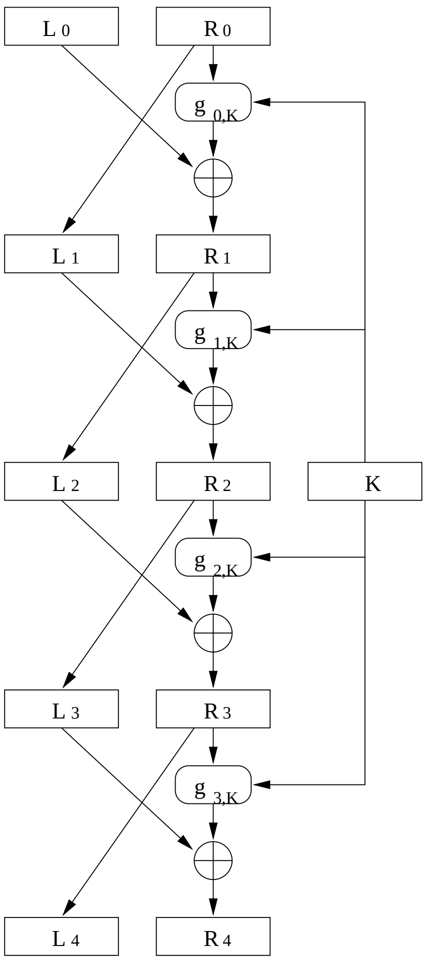

Figure 3: The encryption scheme HESS$_K(\cdot)$

4. Let $g_i(x) = y_0y_1 \ldots y_{l-1}$.

Let L and R denote the left and right half of a $2n$-bit string $L||R$, K be any fixed-length binary string. Then, for each $i \leq 3$, we can define the function $f_{i,K}(\cdot)$ (mapping from $2n$-bit binary strings to $2n$-bit binary strings) by letting

$$f_{i,K}(L||R) = R||(L \oplus g_{i,K}(R)),$$

that is, the right half of the argument appears unchanged as the left half of the result and the right half of the result is equal to $L \oplus g_{i,K}(R||K)$. The encryption scheme (see Figure 3) is defined as the permutation

$$\text{HESS}_K(L||R) = f_{3,K}(f_{2,K}(f_{1,K}(f_{0,K}(L||R))))$$

where K is the secret key. For a $2n$-bit string $L||R$, let

$$f^{-1}_{i,K}(L||R) = (R \oplus g_{i,K}(L))||L.$$

Then for a $2n$-bit ciphertext $L_4||R_4$, the plaintext is computed by the following operation:

$$\text{HESS}^{-1}_K(L_4||R_4) = f^{-1}_{0,K}(f^{-1}_{1,K}(f^{-1}_{2,K}(f^{-1}_{3,K}(L_4||R_4))))$$

In the following, we show that if H is a pseudo-random function, then the encryption scheme HESS$_K(\cdot)$ is provable secure against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.

LEMMA 3.1. *Let n, m and l be three integers such that $lm = n$ and $l << n$. Assume that H is a pseudo-random function mapping arbitrary length input binary strings x to m-bit binary strings $H(x)$. For each $i \leq 3$ and any fixed binary string K, let $g_{i,K}$ be defined as in Figure 2. Then the function set $\{g_{i,K} : i = 0, 1, 2, 3\}$ is indistinguishable from a set of four pseudo-randomly chosen functions mapping arbitrary length binary strings to n-bit binary strings.*

PROOF. The details of the proof will be given in the final version of this paper. Q.E.D. □

THEOREM 3.2. *Let H be a pseudo-random function and $\{g_{i,K} : i = 0, 1, 2, 3\}$ be a set of functions constructed from H as in Figure 2. Then the encryption scheme HESS$_K(\cdot)$ is provable secure against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.*

PROOF. This follows from Lemma 3.1 and the results in Luby and Rackoff [6]. Q.E.D. □

If it is preferred, Naor and Reingold's [8] technique could be used to replace the functions f_{3,K_3} and f_{0,K_0} in HESS$_K(\cdot)$ with two pair-wise independent permutations, which may improve the performance in some applications. That is, let $h_1(\cdot)$ and $h_2(\cdot)$ be two pair-wise independent permutations on the set of $2n$-bit binary strings. Then

$$\text{HESS}''_K(L||R) = h_2^{-1}(f_{1,K}(f_{0,K}(h_1(L||R))))$$

is a provable secure encryption scheme against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks.

In the following, we show that the encryption scheme HESS$_K(\cdot)$ is a very efficient scheme. In practice, we can replace the pseudo-random function H in Figure 2 with appropriately chosen hash functions such as slightly modified SHA-256 [10] algorithms. In the following, by replacing H with the SHA-256 algorithm without the preprocessing padding procedure [10][1] (that is, the value $H(x)$ is computed by applying SHA-256 to the string x without the padding procedure in the SHA-256 algorithm), we give an exemplar analysis of using the cipher HESS$_K(\cdot)$ to encrypt storage devices at sector (block) level.

Assume that the block unit of the storage device is 1024 bytes and SHA-256 will be used as the underlying hash function. Then HESS$_K(\cdot)$ can be used with the parameter $n = 4096$, $m = 256$, and $l = 16$ (note that 1024 bytes are $8162 = 2 \times 4096$ bits). That is, each storage device block unit is divided into two parts L and R, each of 4096 bits. The input to the first H function is $4096 + 8 + k + |etc_i|$ bits, where k is the bit-length of the secret key and $|etc_i|$ is the bit-length of the additional input etc_i. As in the analysis for the encryption scheme $\mathcal{E}$, we may assume that 128-bit key is used and 32 bits are enough for etc_i. Thus the length of the input to the first H function is 4264 bits. Since the block size of SHA-256 is 512 bits, SHA-256 (without the preprocessing padding procedure) needs a total of 9 times of the basic four-step SHA-256 operation. For inputs x of 512 bits, the evaluation of $H(x)$ is just a single application of the basic four-step SHA-256 operation (without the preprocessing padding procedure). Thus the evaluation of the function $g_{i,K}$ on a 4264-bit input needs a total of $l + 9 = 25$ times of the basic four-step SHA-256 operation. In a summary, if

[1]In the algorithm SHA-256, j-bit input strings are padded to a multiple of 512 bits as follows: append the bit "1" to the end of the input, followed by k zero bits, where k is the smallest, non-negative solution to the equation $j + 1 + k \equiv 448 \bmod 512$. Then append a 64-bit block that is equal to the number j expressed using a binary representation.

SHA-256 without the processing padding procedure is used as the pseudo-random function H and 128 bits secret key is used, then a total of $4 \times 25 = 100$ (note that $\mathcal{E}$ needs 576) times of the basic four-step SHA-256 operations are needed to encrypt a 1024-byte storage device block unit. This will certainly be much faster than CBC-AES mode used to encrypt such a 1024-byte storage device block unit. Additionally, the scheme HESS can be used to encrypt 2^n $(= 2^{4096}$ for one kilobyte storage block device unit) blocks of data securely (against birthday attacks) which is extremely larger than the blocks $(= 2^{64})$ of data that AES-128 can be used to encrypt securely.

For SHA-512, $\text{HESS}_K(\cdot)$ can be used with the parameter $n = 4096$, $m = 512$, and $l = 8$. A similar analysis shows that if SHA-512 without the preprocessing padding procedure is used as the pseudo-random function H and 256 bits secret key is used, then a total of $4 \times 13 = 52$ (note that $\mathcal{E}$ needs 160) times of the basic four-step SHA-512 operations are needed to encrypt a 1024-byte storage device block unit.

Naor and Reingold's [8] method can be applied to the scheme $\text{HESS}_K(\cdot)$ by replacing the first and last rounds with pair-wise independent permutations. The new scheme is still provable secure against adaptive chosen plaintext and chosen ciphertext attacks. The speed of the new scheme is two times faster than the speed of the original $\text{HESS}_K(\cdot)$ scheme if we ignore the time required for the two permutations.

In the final version of this paper, we will implement and compare the performances of AES-128 and HESS for storage encryption. In table 1, we list a comparison of calls to the basic operations when encrypting a kilobyte storage block. For example, the table shows that in order to encrypt a kilobyte storage block, AES-128 needs 64 calls to the encryption subroutine on 128-bit inputs, while HESS-SHA-256 with Naor and Reingold's method needs 50 calls to the SHA-256 basic operation on 512-bit inputs (without the padding procedure). In another word, if we use HESS-SHA-256 to encrypt a kilobyte storage device block, then the time needed is approximately the time needed for HESS-SHA-256 to hash a data block of 50×512 bits (which is approximately 3KB).

Table 1: AES and HESS primitive calls comparison for one kilobyte block

cipher	**primitive operation calls**	**primitive block size (bits)**
AES-128	64	128
HESS-SHA-256	100	512
HESS-SHA-256 (NR)	50	512
HESS-SHA-512	52	1024
HESS-SHA-512 (NR)	26	1024

In summary, we have proposed a new encryption scheme $\text{HESS}_K(\cdot)$ that has the following desirable properties:

- The scheme is built on a single one-way hash function which is used as a black box, without the need to tweak the program for the hash function.
- The performance of the cipher is very good.
- It is flexible. It can accommodate more rounds when necessary, and cater for different key lengthes as well as large block sizes.
- It allows applications to specify the etc_i part which may include such information as block index, time stamp, even file name etc. This provides an avenue for integrity checking by application programs.
- It is provably secure under the assumption that the one-way hash function is a pseudo-random function.
- It enjoys a compact code footprint, an especially useful property for small devices.
- It allows users to choose their favorite one-way hash function in implementing the cipher in practice.

4. REFERENCES

[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proc. of the 38th Symposium on Foundations of Computer Science*, IEEE Press, 1997.

[2] M. Bellare, T. Krovetz, and P. Rogaway. Luby-Rackoff backwards: increasing security by making block ciphers non-invertible. In: *Proc. Eurocrypt 98, Lecture Notes in Computer Science 1403*. Springer-Verlag 1998.

[3] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.* **13**:850–864, 1984.

[4] S. Halevi and P. Rogaway. A tweakable enciphering mode. In: *Crypto 03*, LNCS 2729, pages 482–499, Springer-Verlag.

[5] M. Liskov, R, Rivest, and D. Wagner. Tweakable block ciphers. In: *Crypto 02*, LNCS 2442, pages 31–46, Springer-Verlag.

[6] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Comput.*, **17**:373–386, 1988.

[7] U. Maurer. A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators. In *Proc. Eurocrypt 92, Lecture Notes in Computer Science 658*, pages 239–255, Springer-Verlag, 1992.

[8] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. In *Proc. of the 29th ACM Symp. on Theory of Computing*, pages 189–199, 1997, ACM Press.

[9] M. Naor and O. Reingold. A pseudorandom encryption mode. Technical report submitted to IEEE Storage Device Standard Working Group.

[10] NIST. Secure Hash Standards (SHS), FIPS 180-2. August, 2002.

[11] A. Yao. Theory and applications of trapdoor functions. *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 80–91, 1982.

[12] Y. Zheng, T. Matsumoto, and H. Imai. Impossibility and optimality results on constructing pseudorandom permutations. In *Eurocrypt 89, LNCS 434*, pages 412–422, Springer-Verlag, 1990.