ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

STES [Sprout]: A New Method for Securing Data Using Hashing

Ansy V Muhammed [1], Geethu T George[2]

P.G. Student, Department of Electronics and Communication Engineering, Indira Gandhi Institute of Engineering and Technology for Women, Nellikuzhy, Kerala, India[1]

Associate Professor, Department of Electronics and Communication Engineering, Indira Gandhi Institute of Engineering and Technology for Women, Nellikuzhy, Kerala, India[2]

ABSTRACT: Traditionally, cryptography has been mainly used to secure data in transit. In the last decade, however, there has been an increase in interest in securing stored data. This interest is reflected in some recent standardizing efforts and a galaxy of algorithms that have been proposed for securing stored data. A consensus has been reached among researchers that a type of symmetric encryption scheme, called Tweakable Enciphering Scheme (TES) is appropriate for the application of encrypting data stored in storage devices which have a sector-wise organization including hard disks and NAND flash memories.The aim of this project is to encrypt flash memories which have a block-wise organization. Our target applications include USB memories and memory cards like those specified in the SD standard and high density smart cards which contain NAND flash memories of the order of megabytes for storing user data. The stream cipher based construction uses a high level a stream cipher and a hash function. The design of the hash function and the consequent development of a new TES scheme results in STES. The eStream Profile-2 portfolio provides three stream ciphers which have very small hardware footprint, namely Grain128, Mickey 2.0 and Trivium. We consider Grain128, Mickey 2.0 and a new stream cipher named Sprout is used to describe implementation results.

KEYWORDS:TES, NAND flash memories, hash function, Grain128, Mickey 2.0, Sprout,

I. INTRODUCTION

The problem of securing data present on USB memories and SD cards has not been adequately addressed in the cryptography literature. While the formal notion of a TES is well accepted as the proper primitive for secure data storage, the real challenge is to design a low cost TES which can perform at the data rates of the targeted memory devices. In this work, we provide the first answer to this problem. Our solution, called STES, combines a stream cipher with a XOR universal hash function. The specific application which a TES is meant to serve is called low level disk encryption or in-place disk encryption. In this application, it is assumed that the encryption/decryption algorithm resides in the disk controller and it views the storage media as a collection of sectors. The disk controller encrypts the data before writing and decrypts it after reading and before sending it to the operating system. This generic model of disk encryption is independent of other details like operating systems, file system types, etc.

Almost all the known constructions of TES use a block cipher as a building block. Some schemes like CMC, EME, EME* use only block ciphers, whereas schemes such as XCB, HCTR, HCH, TET, HEH, use a block cipher along with a suitable hash function. In terms of efficiency, the current state-of-the-art for the cost of encryption/decryption per block is either two block cipher calls or 1 block cipher call plus 1 $GF(2^{n})$ multiplication, where n is the size of the underlying block cipher. Though there has been an active effort in designing new TES, there are only a few works which report efficient implementations of such schemes.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

The aim of this project is to encrypt flash memories which have a block-wise organization. Our target applications include USB memories and memory cards like those specified in the SD standard and high density smart cards which contain NAND flash memories of the order of megabytes for storing user data. The SD standard classifies memory cards into four categories based on their speeds. These categories are named as normal speed, high speed, ultra high speed-I (UHS-I) and ultra-high speed-II (UHS-II) and the required bus speeds are 12.5 MB/sec; 25 MB/sec; 50-104 MB/sec; and 156-312 MB/sec respectively. The UHS-II categories of devices are only recommended for special applications like storing high quality streaming video.

These speed requirements are to be contrasted with the speeds of modern hard disks using technologies like serial ATA and native command queuing which can achieve data rates of more than 3 Gb/sec (the SATA revision 2 specifies a speed of 6 Gigabits/sec). So, for encryption of SD cards, the speed of encryption is not much demanding, and is much less than the speeds achieved by the present TES implementations. As discussed above, speed is only one of the issues. The existing designs are neither low area nor low power and also not low cost. Our target platform is low cost yet performance oriented. It may be possible to put such a platform in a personal device like a smart phone. TES designs targeted towards such platforms can be directly deployed to small devices.

II. RELATED WORK

A. TES

The known constructions of TESs can be broadly divided into two types: encrypt-mix-encrypt type and the hash-encrypt-hash type. The first type uses a block cipher while the second type additionally requires a finite field multiplier. The feature common to all previous constructions is that they require both the encryption and the decryption functions of the underlying block cipher. The major application of TES is disk encryption where the implementation of TES is done in hardware which typically resides just above the disk controller. Thus, for disk encryption application the new constructions lead to substantial savings in hardware.

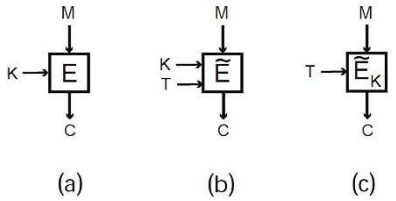

Fig.1: Tweakable enciphering scheme

Figure (a) shows the standard enciphering where message M is under the influence of a key K to produce cipher text C. Figure (b) shows the tweakable enciphering where message M is under the influence of not only a key K but also a tweak T to yield a cipher text. (c) Figure shows another method of representing tweakable enciphering scheme; here the key K is shown inside the box.

A tweakable enciphering scheme is a pair of indexed family of functions $(D_K, E_K)K\epsilon$ K. For each K in K, E_K: Tweak X Msg $\rightarrow$Cpr, and D_K: Tweak X Cpr $\rightarrow$ Msg, where Tweak, Msg and Cpr are non-empty finite sets of binary strings. The set K is called the key space, Tweak is the tweak space, Msg is the message space and Cpr is the ciphertext space. The functions have to satisfy the following condition. For fixed K ϵ K and T ϵ Tweak,

1) $E_K(T)$ and $D_K(T)$ are length preserving permutations.
2) $D_K(T, E_K(T, X) = X$

Usually $E_K(T,X)$ and $D_K(T,Y)$ are written as $E_K^T(X)$and $D_K^T(Y)$ respectively. E is called the encryption function and D is called the decryption function. The applicability to disk encryption comes via the following relation. For encryption, the tweak T is taken to be the sector address and the message X is taken to be the content of the sector. Let Y be the output of the encryption of X under the tweak T and the secret key K. By the length preserving constraint, this Y is of the same length as that of X. The value of X is overwritten using Y in the sector pointed to by T. Thus, after the whole disk is encrypted, it consists only of the encrypted values. Note that the encryption is done sector-wise and so

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

decryption can also be done sector wise. This means that the value Y in a particular sector with address T can be decrypted without disturbing the values of the other sectors.
In a typical in-place disk encryption application, the sectors are individually encrypted and stored in the encrypted form. The encryption/decryption module resides just above the disk controller. An encrypted sector read by the disk controller is decrypted by the module and returned to the calling routine. Similarly, the writing of any sector to the disk is first encrypted by the module and then the disk controller writes it to the appropriate sector. This mode of operation ensures that if the disk is accessed without the secret key, then it appears to be random. In the definition of TES, Tweak, Msg and Cpr are mentioned to be non-empty finite sets of binary strings. For disk encryption applications, Msg = Cpr and is the set of all binary strings whose lengths are equal to the length of a sector. Similarly, Tweak is taken to be the set of all binary strings of some fixed length l where 2^l is greater than the number of sectors in a disk.

B. The MICKEY Stream Ciphers

The family of stream ciphers MICKEY (which stands for Mutual Irregular Clocking KEY stream generator) is aimed at resource-constrained hardware platforms. It is intended to have low complexity in hardware, while providing a high level of security. It uses irregular clocking of shift registers, with some novel techniques to balance the need for guarantees on period and pseudo randomness against the need to avoid certain cryptanalytic attacks.

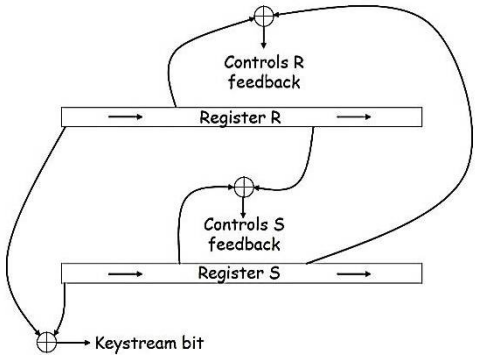

Fig.2. MICKEY algorithm structure

The MICKEY family of algorithms was designed in response to the ECRYPT “Call for Stream Cipher Primitives” in 2005, and directed at Profile 2 stream ciphers intended for use on resource-constrained hardware platforms. Specifically, it is intended to have low complexity in hardware, while providing a high level of security. In fact, two variants of the algorithm have been defined: MICKEY, with an 80-bit key, and MICKEY-128, with a 128-bit key.`MICKEY' is an abbreviation of `Mutual Irregular Clocking KEYstream generator', and this encapsulates the original design concept, illustrated in Figure 2.
The algorithm is based around two registers R and S, each of which has two modes of clocking selected by a control bit. With this as a starting point, we were lead to design a clocking rule for the ensemble (R, S), in which the control bit for each register is formed from combination of bits dependent on both registers. It was also intended from the outset that R should clock as a Galois-stepping feedback shift register either 1 or J times, given that J steps can be implemented efficiently in a single clock cycle by taking advantage of an idea introduced by Jansen. The register S, on the other hand, was intended to clock non-linearly, in two different ways. Successive bits of keystream are formed by combining bits from the registers R and S. Broadly speaking, the idea was that the linearity of R would ensure good statistical properties and guarantees about period, whilst the non-linearity of S would protect against attacks that might be mounted against a linear system.

C. The Grain-128 Stream Ciphers

Symmetric cryptographic primitives for encryption are divided into block ciphers and stream ciphers. The general opinion seems to be that pure stream ciphers are still interesting for two reasons. First, they can be designed to allow

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

much faster keystream generation in software than a block cipher in stream cipher mode. Second, they can be designed to be smaller in hardware. Hence, for a stream cipher to be interesting it must either be very fast in software or very small in hardware. This opinion has been reflected by the eSTREAM project, which is an effort to identify new stream ciphers that might be interesting for widespread adoption. The eSTREAM project focuses on stream ciphers satisfying one or both of the above mentioned criteria.

The encryption operation in a binary additive stream cipher is simple. The keystream is binary xored with the plaintext to form the ciphertext. Similarly, decryption is done by xoring the same keystream with the ciphertext to obtain the original plaintext. The most important property of a stream cipher is the resistance against different attacks. It should not be possible to recover the initial state of the cipher and the generated keystream should be indistinguishable from a truly random sequence. There are several ways to construct a stream cipher. A common building block is a linear feedback shift register (LFSR). These have several statistical properties that coincide with the statistical properties of truly random strings. However, since the generation of new symbols is linear, some additional building block is needed in order to remove the linearity. Irregular clocking, nonlinear filters and decimation algorithms are commonly used to introduce nonlinearity in the keystream.

Grain is a binary additive stream cipher. The previous version, which is denoted Grain Version 1, targets applications which have very limited hardware resources. Grain Version 1 supports a key size of 80 bits (as specified in eSTREAM), which is not feasible to exhaustively search with modern computers. Recent research in time memory data trade-off attacks suggests that it is possible to mount an attack with complexity $O(2^{K/2})$ where K is the size of the key. In this scenario the attacker has a collection of $2^{K/2}$ plaintexts encrypted under different keys and the aim of the attack is to find one of these keys. In this attack scenario 80 bit key size is not enough since an attack would have complexity $O(2^{40})$. Several researchers have recently expressed the opinion that 128 bit keys is a minimum in secure applications.

Grain-128 is designed to meet this requirement while preserving the advantages of Grain Version 1. It supports key size of 128 bits and IV size of 96 bits. The cipher is still very small and easy to implement in hardware. Furthermore, it is possible, and easy, to increase the speed at the expense of more hardware. This is a distinguishing feature of the Grain family of stream ciphers and is not explicitly found in many other ciphers. Grain-128 uses a linear feedback shift register to ensure good statistical properties and to guarantee a lower bound for the period of the keystream. To introduce nonlinearity, a nonlinear feedback shift register (NFSR) is used together with a nonlinear filter. The nonlinear filter takes input from both shift registers.

D. The Sprout Stream Ciphers

The design of Sprout is an adaptation of the basic design used for the Grain family of stream ciphers. More precisely, we used Grain 128a as the starting point as this is the newest member of the Grain family which overcomes some weaknesses found for previous version. Each Grain cipher is composed of a linear feedback shift register (LFSR), a non-linear feedback shift register (NLFSR), and an output function. The LFSR and NLFSR states represent the internal states which at the beginning are initialized with the key and the IV. The output of the LFSR is fed into the NLFSR to ensure a minimum period with respect to the internal states while the purpose of the NLFSR is to make certain standard attacks like algebraic attacks infeasible. Moreover, several bits are taken from both FSRs as input to the output function. During the initialization phase, the outputs of the output function are fed back into the FSRs, while in the keystream generation phase, they represent the keystream bits.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

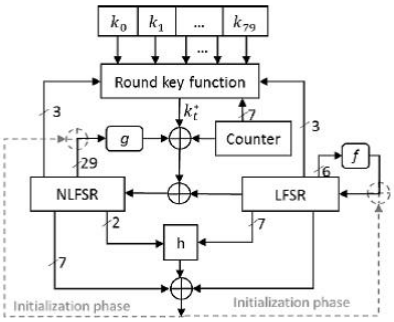

Fig.3: The structure of sprout.

Sprout uses significantly less area than comparable existing lightweight stream ciphers. Sprout reduces the size of the internal state used in stream ciphers while resisting to time-data-memory trade-off (TMDTO) attacks. This is done by introducing a state update function that depends on a fixed secret key. The designers expect a minimum time effort equivalent to an exhaustive search of the key for an attacker to lead an attack, since she has to determine the key prior to realise the TMDTO. It has an IV and a key size of 80 bits.

E. Multilinear Universal Hash (MLUH)

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

A keyed hash function is chosen from an indexed family $\{H_\tau\}_\tau$ of hash functions. The key τ is chosen uniformly at random from the index set. Suppose the range consists of *l*-bit strings. The hash function is said to be universal if for distinct X and X′, $\Pr[H_\tau(X) = H_\tau(X')] = 2^l$; further it is said to be XOR universal (XU), if for distinct X and X′ and any *l*-bit string Y, $\Pr[H_\tau(X) \oplus H_\tau(X') = Y] = 2^l$. We will be interested in a particular type of hash function called the multi-linear function. The following definition is a variant based on the so-called Toeplitz construction. A Multilinear Universal Hash with data path *d* uses an (m+b-1) *d* - bit key K to map a d_m - bit message M to a d_b - bit digest. The message M is written as $M = M_1 \| M_2 \| \ldots \| M_m$ and the key K is written $K = K_1 \| K_2 \| \ldots \| K_{m+b-1}$, where each M_i and K_j are *d* bits long. We define

$$\mathrm{MLUH}_K^{d,b}(M) = h_1 \| h_2 \| \ldots \ldots \| h_b ,$$

Where,

$$\left.\begin{aligned} h_1 &= M_1 \cdot K_1 \oplus M_2 \cdot K_2 \oplus \ldots\ldots \oplus M_m \cdot K_m \\ h_2 &= M_1 \cdot K_2 \oplus M_2 \cdot K_3 \oplus \ldots\ldots \oplus M_m \cdot K_{m+1} \\ &\ldots\ldots\ldots\ldots\ldots \\ h_b &= M_1 \cdot K_b \oplus M_2 \cdot K_{b+1} \oplus \ldots \oplus M_m \cdot K_{b+m-1} \end{aligned}\right\} \quad (1)$$

The additions and multiplications are in the field $GF(2^d)$. Note that the message and key lengths are multiples of *d*. This restriction can be lifted by appropriate padding. We do not perform this, since for our application it is easy to ensure that the condition holds. It is not difficult to show that an MLUH is an XU function. More specifically for a uniform random key K, any pair of distinct messages M_1, M_2 and any α

$$Pr_K[\mathrm{MLUH}_K^{d,b}(M_1) \quad \mathrm{MLUH}_K^{d,b}(M_2) = \alpha] \leq \frac{1}{2^{db}} \quad (2)$$

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

F. Pseudo-Dot Product Based Universal Hash

If Another construction of hash function can be based on Winograd's pseudo-dot product. We describe the Toeplitz variant of the pseudo-dot product construction. This will be denoted by PD. The PD construction uses an (M+2b-2) *d*-bit key K to map a d_m-bit message M to a d_b-bit digest. The message M is written as $M = M_1 \| M_2 \| \ldots \| M_m$, where each Mi is *d*-bit strings, and the key K written as $K = K_1 \| K_2 \| \ldots \| K_{m+2b-2}$, where each Kj is *d*-bit strings. We define

$$PD_K^{d,b}(M) = h_1 \| h_2 \| \ldots \| h_b,$$

Where,

$$\left.\begin{aligned} h_1 &= (M_1 \oplus K_1)(M_2 \oplus K_2)(M_3 \oplus K_3)(M_4 \oplus K_4) \oplus \ldots \oplus (M_{m-1} \oplus K_{m-1})(M_m \oplus K_m) \\ h_2 &= (M_1 \oplus K_3)(M_2 \oplus K_4)(M_3 \oplus K_5)(M_4 \oplus K_6) \oplus \ldots \oplus (M_{m-1} \oplus K_{m+1})(M_m \oplus K_{m+2}) \\ &\ldots\ldots\ldots\ldots\ldots \\ h_b &= (M_1 \oplus K_{2b-1})(M_2 \oplus K_{2b})(M_3 \oplus K_{2b+1}) \oplus \ldots \oplus (M_{m-1} \oplus K_{2b+m-3})(M_m \oplus K_{2b+m-2}) \end{aligned}\right\} \quad (3)$$

The PD function is also an XU hash function and can be proved by a simple probability calculation.

STES

The description of the encryption algorithm using STES is given in Figure 5.The construction is parameterized by a stream cipher SC supporting l bit IVs, a hash function MLUH with data path d and a fixed l bit string fStr. This is emphasized by writing STES [SC,MLUH,fStr] When one or more of the parameters are clear from the context, we drop these for simplicity of notation; if all three parameters are clear, then we simply write STES. We assume that Plaintexts and tweaks are of fixed length. If P is any plaintext and T is any tweak, then we also assume that d |(|P|+|T|-|2l|).

The secret key for STES is the secret key K of the underlying stream cipher. In this context, we would like to mention the role that the parameter fStr can play. From the point of view of the formal security analysis, there is no restriction on fStr. Thus, this can be used as a secret customization option. In other words, for actual deployment, one may choose a uniform random value for fStr and keep it secret. This provides an additional layer of obscurity over and above the provable security analysis that we perform.

There is another advantage to using fStr as part of the secret key. The security bound that is obtained is in terms of the IV length *l* and the number of queries for which security holds can be obtained as a function of 2^l. The key length |K| should be at least *l* for the analysis to be meaningful. If the key length is equal to *l*, then certain "out of model" attacks may apply. Increasing the key length by keeping fStr as part of the secret key may help in preventing such attacks.Apart from the secret key K, the input to the encryption algorithm of STES is the tweak T and a plaintext P. Similarly, the input to the decryption algorithm of STES consists of T and the ciphertext C.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

$STES.Encrypt_K^T(P)$	$STES.Decrypt_K^T(C)$
1. $b \leftarrow \frac{\ell}{d}$;	1. $b \leftarrow \frac{\ell}{d}$;
2. $b_1 \leftarrow \frac{\lvert P\rvert+\lvert T\rvert-2\ell}{d}$;	2. $b_1 \leftarrow \frac{\lvert C\rvert+\lvert T\rvert-2\ell}{d}$;
3. $\ell_1 \leftarrow (b_1+b-1)d$;	3. $\ell_1 \leftarrow (b_1+b-1)d$;
4. $\ell_2 \leftarrow (2b-1)d$;	4. $\ell_2 \leftarrow (2b-1)d$;
5. $\ell_3 \leftarrow \lvert P\rvert - 2\ell$;	5. $\ell_3 \leftarrow \lvert C\rvert - 2\ell$;
6. $P_1 \leftarrow \mathsf{bits}(P,1,\ell)$; /* $\lvert P_1\rvert = \ell$) */	6. $C_1 \leftarrow \mathsf{bits}(C,1,\ell)$; /* $\lvert C_1\rvert = \ell$) */
7. $P_2 \leftarrow \mathsf{bits}(P,\ell+1,2\ell)$; /* $\lvert P_2\rvert = \ell$ */	7. $C_2 \leftarrow \mathsf{bits}(C,\ell+1,2\ell)$; /* $\lvert C_2\rvert = \ell$ */
8. $P_3 \leftarrow \mathsf{bits}(P,2\ell+1,\lvert P\rvert)$; /* $\lvert P_3\rvert = \ell_3$ */	8. $C_3 \leftarrow \mathsf{bits}(C,2\ell+1,\lvert C\rvert)$; /* $\lvert C_3\rvert = \ell_3$ */
9. $\tau \leftarrow \mathsf{SC}_K^{\ell_1+\ell_2+\ell}(\mathsf{fStr})$;	9. $\tau \leftarrow \mathsf{SC}_K^{\ell_1+\ell_2+\ell}(\mathsf{fStr})$;
10. $\tau' \leftarrow \mathsf{bits}(\tau,1,\ell_1)$;	10. $\tau' \leftarrow \mathsf{bits}(\tau,1,\ell_1)$;
11. $\beta \leftarrow \mathsf{bits}(\tau,\ell_1+1,\ell_1+\ell)$;	11. $\beta \leftarrow \mathsf{bits}(\tau,\ell_1+1,\ell_1+\ell)$;
12. $\tau'' \leftarrow \mathsf{bits}(\tau,\ell_1+\ell+1,\ell_1+\ell+\ell_2)$;	12. $\tau'' \leftarrow \mathsf{bits}(\tau,\ell_1+\ell+1,\ell_1+\ell+\ell_2)$;
13. $Z_1 \leftarrow \mathsf{MLUH}_{\tau'}^{d,b}(P_3\|\|T) \oplus \beta$;	13. $Z_2 \leftarrow \mathsf{MLUH}_{\tau'}^{d,b}(C_3\|\|T) \oplus (\beta \lll 1)$;
14. $A_1 \leftarrow P_1$;	14. $B_1 \leftarrow C_1 \oplus Z_2$;
15. $A_2 \leftarrow P_2 \oplus Z_1$;	15. $B_2 \leftarrow C_2$;
16. $(B_1,B_2,W) \leftarrow \mathsf{Feistel}_{K,\tau''}^{\ell,d}(A_1,A_2,\ell_3)$;	16. $(A_1,A_2,W) \leftarrow \mathsf{InvFeistel}_{K,\tau''}^{\ell,d}(B_1,B_2,\ell_3)$;
17. $C_3 \leftarrow P_3 \oplus W$;	17. $P_3 \leftarrow C_3 \oplus W$;
18. $Z_2 \leftarrow \mathsf{MLUH}_{\tau'}^{d,b}(C_3\|\|T) \oplus (\beta \lll 1)$;	18. $Z_1 \leftarrow \mathsf{MLUH}_{\tau'}^{d,b}(P_3\|\|T) \oplus \beta$;
19. $C_1 \leftarrow B_1 \oplus Z_2$;	19. $P_1 \leftarrow A_1$;
20. $C_2 \leftarrow B_2$;	20. $P_2 \leftarrow A_2 \oplus Z_1$;
$\textbf{return}(C_1\|\|C_2\|\|C_3)$;	$\textbf{return}(P_1\|\|P_2\|\|P_3)$;

Figure 4: TES using SC and MLUH Encryption/ Decryption algorithm (*l*-bit string fStr, IV length of SC is *l* and the data path of MLUH is d.)

The encryption algorithm begins with some length calculations, and fixes values for the variables l_1, l_2 and l_3 which determine the key lengths necessary for the different calls to MLUH made later in the algorithm. Next, the input plaintext is parsed into three parts P_1, P_2 and P_3 where P_1 and P_2 are both *l* bits long and P_3 is |P| - 2*l* bits long. In Line 9,(l_1+ l_2+ l_3) bits are generated from the stream cipher SC_K using the fStr as input. These bits are parsed into two strings ζ′ and ζ″ which are later used as keys for MLUH. The part P_3 of the message and the tweak T is hashed using the MLUH and mixed with the message parts P_1 and P_2 to generate two strings A_1 and A_2. These strings are used as an input to the function Feistel which is described in Figure 6. The function Feistel receives two keys K and ζ″ and it mixes the input strings A_1 and A_2 by appropriate use of the hash MLUH and the stream cipher SC. The inverse function for Feistel is also shown in Figure 5.

The call to Feistel also returns the string W of length equal to l_3 which is the length of P_3. This string W is XORed with P_3 to obtain C_3. The Feistel network produces two more *l*-bit outputs B_1 and B_2. B_1 and B_2 are used to produce the ciphertexts C_1 and C_2 respectively. From the description given in Figures 4 and 5, it may appear that the string W of length l_3 is required to be actually returned to the main body of the algorithm. This, however, is not the case. For example, depending on the specific design choices it may be possible to start computing W as soon as few bits of Z_2 are available.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

$\mathsf{Feistel}^{\ell,d}_{K,\tau''}(A_1, A_2, i)$	$\mathsf{InvFeistel}^{\ell,d}_{K,\tau''}(B_1, B_2, i)$
1. $H_1 \leftarrow \mathsf{MLUH}^{d,b}_{\tau''}(A_1)$;	1. $H_2 = \mathsf{MLUH}^{d,b}_{\tau''}(B_2)$;
2. $F_1 \leftarrow H_1 \oplus A_2$;	2. $F_2 = H_2 \oplus B_1$;
3. $(G_1, W) \leftarrow \mathsf{SC}^{\ell+i}_K(F_1)$;	3. $G_2 = \mathsf{SC}^{\ell}_K(F_2)$;
4. $F_2 \leftarrow A_1 \oplus G_1$;	4. $F_1 = B_1 \oplus G_2$;
5. $G_2 \leftarrow \mathsf{SC}^{\ell}_K(F_2)$;	5. $(G_1, W) = \mathsf{SC}^{\ell+i}_K(F_1)$;
6. $B_2 \leftarrow F_1 \oplus G_2$;	6. $A_1 = F_2 \oplus G_1$;
7. $H_2 \leftarrow \mathsf{MLUH}^{d,b}_{\tau''}(B_2)$;	7. $H_1 = \mathsf{MLUH}^{d,b}_{\tau''}(A_1)$;
8. $B_1 \leftarrow H_2 \oplus F_2$;	8. $A_2 \leftarrow H_1 \oplus F_1$;
return (B_1, B_2, W);	***return*** (A_1, A_2, W);

Figure.5: The Feistel network (and its inverse) constructed using a stream cipher and a MLUH.

III. EXPERIMENTAL RESULTS

As we stated in introduction the design objective would be to match the data rates of modern day disk controllers. Using a STES based on Sprout helps to achieve such data rates though this strategy provides more compact designs. If we are interested in encrypting hard disks of desktop or laptop computers the area constraint is not that high, but speed would be the main concern. So, a pipelined architecture will probably be the best choice for designing disk encryption schemes, if we are interested in encrypting SD cards the area constraint is high. From former studies it is clear that Sprout Exploits the fact that storing fixed values is less area consuming than using registers, thus Sprout has a significantly smaller area size.

- If the stream cipher acts like a random function then, for any arbitrary adversary which makes a reasonable number of queries, the advantage of distinguishing STES [SC] from a tweak-indexed family of length preserving permutations is small.
- For higher data paths, the constructions with PD have lesser area than the constructions with MLUH. This is consistently observed for, d > 8. For 4-bit data paths implementations the MLUH based constructions are smaller.
- The constructions with PD operates at a slightly lower frequency than the MLUH based constructions, this is due to the fact that PD has a more complex circuitry. Moreover, PD based constructions take 4|IV|/d cycles more than the MLUH based constructions. Thus the PD based constructions have a lower throughput.

The estimated results have been compared to evaluate the improvement of the design over the past designs. Table 1 compares the different designs of STES

Cipher	Area size(GE)	Throughput (Kb/s)	Process Time (sec)
Mickey	3188	100	0.13µm
Grain-120	1162	100	0.18µm
Sprout	813	100	0.18µm

Table 1: Comparison Results

From above table it is clear that the STES [GRAIN] and STES [SPROUT] has equal time requirements. Since the area constraint is an integral part of disc encryption we can say that the modified STES [SPROUT] is so far the best stream based TES.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

IV. CONCLUSION

The main design goal of STES was to obtain a TES which can be implemented in a compact form and would have low power consumption. Our experiments validate that STES does achieve these goals to a large extent. In the introduction we mentioned the speeds recommended by the SD standard. The commercially available memories do not achieve the values specified in the standard. From our studies it is clear that STES can serve as a viable scheme for encryption in a large class of non-volatile memory devices.

- Message length: Our target is encryption of fixed size blocks. So, STES has been designed for fixed length messages. In particular, we consider the message length to be 512 bytes. This value matches the current size of memory blocks. It is to be observed that the design philosophies are quite general and can be scaled suitably for other message lengths.
- Stream Cipher: We chose the following stream ciphers: Grain128,Mickey128 2.0 and Sprout. These stream ciphers has the hardware oriented designs. There are several works in the literature which reports compact hardware implementations of these ciphers
- Hash Function: The main component required to implement MLUH or PD is a finite field multiplier. We describe MLUH parameterized on the data path d, which signifies that the multiplications in MLUH with data path d take place in the field GF (2^d). The case of PD is a bit curious. For implementing PD with data path d we use multipliers in GF ($2^{d/2}$) This decision was taken to make the design suitable for STES. Note that in case of PD each multiplication requires two message and key blocks which is not the case in MLUH. As the message and key blocks are generated on the fly hence this design helps in preventing data stalls in the circuit. The 4-bit design of the PD is not more efficient than the MLUH architecture. Though the 4-bit multipliers used in PD are smaller than the 8-bit multipliers but we require the double of them and also the double amount of registers and the extra xors required at the input of the multipliers makes this PD architecture marginally more costly than the MLUH architecture.
- Security Statement for STES: The following theorem specifies the security of STES. Let SC_K: $\{0,1\}^l \rightarrow \{0,1\}^L$ be a stream cipher with IV and H be one of the hash functions MLUH or PD. For $\sigma \geq q \geq 1$ and $t \geq 1$,

$$\mathrm{Adv}^{\pm \widetilde{prp}}_{\mathrm{STES}\,[\mathrm{SC,H,fStr}]}(t, q, \sigma) \leq \mathrm{Adv}^{prf}_{\mathrm{SC}}(t', q, \sigma) + \frac{9q^2}{2^{l+1}}$$

Here t' is $t + t''$, where t'' is the time to process q queries using STES [SC, H, fStr].The theorem guarantees that if the stream cipher acts like a random function then, for any arbitrary adversary which makes a reasonable number of queries, the advantage of distinguishing STES [SC] from a tweak-indexed family of length preserving permutations is small.To thwart TMDTO attacks, it is not necessary to achieve 2^k equivalence classes. Even if a cipher achieves less than these, it may still be sufficient if the effort for identifying these classes is sufficiently high. In other words, if the effort for finding all equivalence classes times the effort for executing a TMDTO attack is above the effort for exhaustive key search, we are on the safe side.

REFERENCES

[1] Martin Hell, Thomas Johansson, Alexander Maximov" A Stream Cipher Proposal: Grain-128" Published in: Information Theory,IEEE International Symposium on 9-14 July 2006
[2] M. Hell, T. Johansson, and W. Meier, "Grain - a stream cipher for constrained environments." International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems., 2006.
[3] S. Babbage and M. Dodd, "The MICKEY Stream Ciphers," in New Stream Cipher Designs, M. Robshaw, and O. Billet, Eds. New York, NY, USA: Springer, , pp. 191–209,2008.
[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in Proc. 9th Int. Workshop Cryptographic Hardware Embedded Syst., pp. 450–466,2007.
[5] S. Babbage, *Improved Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference Publication no. 408, pp. 161-166, IEE, 1995.
[6] S. H. Babbage, M. W. Dodd, *The stream cipher MICKEY (version 1)*, SKEW Workshop (Arhus, May 2005).
[7] ECRYPT, "eSTREAM: ECRYPT Stream Cipher Project, IST-2002- 507932," Available at http://www.ecrypt.eu.org/stream/.