Reconfigurable Hardware Implementations of Tweakable Enciphering Schemes

Cuauhtemoc Mancillas-López, Debrup Chakraborty
and Francisco Rodríguez-Henríquez

Abstract

Tweakable enciphering schemes are length preserving block cipher modes of operation that provide a strong pseudo-random permutation. It has been suggested that these schemes can be used as the main building blocks for achieving in-place disk encryption. In the past few years there has been an intense research activity towards constructing secure and efficient tweakable enciphering schemes. But, actual experimental performance data of these newly proposed schemes are yet to be reported. Accordingly, in this paper we present optimized FPGA implementations of five tweakable enciphering schemes, namely, HCH, HCTR, XCB, EME and TET, using a 128-bit AES core as the underlying block cipher. We report performance timings of these modes when using both, pipelined and sequential AES structures. The universal polynomial hash function included in the specification of HCH, HCHfp (a variant of HCH), HCTR, XCB and TET, was implemented using a Karatsuba-Ofman multiplier as the main building block. We provide detailed analyses of each of the schemes and their experimental performances achieved in various scenarios. Our experiments show that a sequential AES core is not an attractive option for the design of these modes as it leads to rather poor throughputs. In contrast, by using an encryption/decryption pipelined AES core we get a throughput of 3.67 Gbps for HCTR and by using a encryption only pipeline AES core we get a throughput of 5.71 Gbps for EME. The performance results reported in this paper provide experimental evidence that hardware implementations of tweakable enciphering schemes can actually match and even outperform the data rates achieved by state-of-the-technology disk controllers, thus showing that they might be used for achieving provably secure in-place hard disk encryption. [1]

The authors are with the Computer Science Department, Centro de Investigación y Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, México D.F 07360

[1]Some parts of the results reported here have been published in [20].

I. Introduction

A block-cipher mode of operation is a specific way to use a block-cipher to enable it to encrypt arbitrary long messages. In the literature, different kinds of modes of operations have been reported, each providing several security services like confidentiality, authentication, etc. A Tweakable Enciphering Scheme (TES) is a specific kind of mode of operation that is based on the notion of tweakable block ciphers introduced in [18]. TES is a length preserving encryption scheme which can encrypt variable length messages. The security that a TES provides is that of a strong pseudorandom permutation (SPRP), i.e., a TES is considered secure if it is infeasible for any computationally bounded adversary to distinguish between the TES and a random permutation. A TES takes as input a quantity called a tweak other than the message and the key. The tweak is supposed to be a public quantity which enriches the variability of the cipher-text produced.

A fully defined TES for arbitrary length messages using a block cipher was first presented in [12]. In [12] it was also stated that a possible application area for such encryption schemes could be low level disk encryption, where the encryption/decryption algorithm resides on the disk controller which has access to the disk sectors but has no knowledge of the disk's high level partitions such as directories files, etc. The disk controller encrypts a message before writing it to a specific sector and decrypts the message after reading it from the sector. Additionally it was suggested in [12] that sector addresses can be used as tweaks. Because of the specific nature of this application, a length preserving enciphering scheme is required and under this scenario, a strong pseudorandom permutation can provide the highest possible security.

In the last few years there have been numerous proposals for TES. These proposals fall in three basic categories: Encrypt-Mask-Encrypt type, Hash-ECB-Hash type and Hash-Counter-Hash type. CMC [12], EME [13], EME* [10] fall under the Encrypt-Mask-Encrypt group. PEP [4], TET [11], HEH [31] fall under the Hash-ECB-Hash type and XCB [24], HCTR [34], HCH [5], ABL [26] fall under the Hash-Counter-Hash type.

Although till date about ten different TES constructions have been proposed, no experimental data of these schemes have been published yet. From the constructions one can easily conclude that the Encrypt-Mask-Encrypt type schemes use approximately two block cipher calls per message block, while the two other types require one block cipher call and two $GF(2^n)$ field multiplications per message block. Hence, from the point of view of efficiency, it was argued in [5], [6], [25] that the first type is slower than the latter two types, provided that one block-cipher call is more expensive than two field multiplications. This

argument, however, can only be sustained if one assumes a software implementation of the mode, where a block-cipher call has the same cost regardless of the data dependencies. It is noticed that this is not the case for a hardware design, where block-cipher invocations can have different timing costs according to the way that the cipher core is implemented (using either a sequential or a pipeline architecture) and the data dependencies associated with the particular mode of operation algorithm under analysis. The same arguments hold for the multipliers as one can have a fully parallel efficient multiplier which can multiply two field elements in one clock-cycle.

A speculative performance comparison of the EME*, XCB, HCH and TET modes of operation in hardware is provided in [11]. This comparison assumes the same hardware implementation setting reported in [2], where a fully-parallel $GF(2^n)$ field multiplier was implemented in one clock cycle at a hardware cost in area of about three times the cost associated with one AES round function, and where the AES core is implemented through the computation of ten such modules. However, this analysis might not be quite accurate because, as we will see in the rest of this paper, one can implement a $GF(2^n)$ field multiplier with an efficiency comparable to the one of an AES round function in terms of both, the critical path and the cost in area.

Keeping in mind the specific application goal of low level disk encryption, a comparative study of performance and cost of the various proposed schemes in hardware is very necessary. The recent standardization activities for such modes by the IEEE working group on storage security [15] also demands performance data for the many proposed schemes.

In this paper we present optimized hardware implementation of five TES. The modes we chose are HCH, HCTR, XCB, EME and TET. Also we provide performance data for a variant of HCH, called HCHfp, which is particularly useful for disk encryption. The rationale behind the choice of these specific modes is discussed next.

The modes that we left out in this study are CMC, PEP, EME*, ABL and HEH. CMC which uses two layers of CBC type encryption cannot be pipelined. PEP and ABL are particularly inefficient compared to their counterparts. EME* is a modification over EME so that it can be used for arbitrary length messages. For the application of disk sector encryption this functionality is not required. HEH is a recently proposed mode which improves the TET mode of operation. HEH is supposed to be more efficient than TET in certain scenarios. In this work we do not provide performance data for HEH, but we provide some discussions and speculations regarding its performance.

For all the implementations we use AES-128 as the underlying block-cipher. Whenever required we use a fully parallel Karatsuba Ofman multiplier to compute the hash functions. We carefully analyze

and present our design decisions and finally report hardware performance data of the five modes. Our implementations show that in terms of area HCTR, HCH, TET and XCB require more area than EME. HCTR performs the best in terms of speed followed by HCHfp, EME, TET, HCH and XCB.

II. NOTATIONS

An n-bit block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, where $\mathcal{K} \neq \emptyset$ is the key space and for any $K \in \mathcal{K}$, $E(K,.)$ is a permutation. We write $E_K(\,)$ instead of $E(K,.)$. We shall generally denote the block-length by n and the number of blocks by m. By $X||Y$ we shall mean the concatenation of two binary strings X and Y and $\mathsf{bin}_n(|X|)$ will denote the n-bit binary representation of $|X|$, which denotes the length of X. By $\mathsf{pad}_r(X)$ we shall mean concatenation r zeros to the end of X and $\mathsf{drop}_r(X)$ will denote the $r \leq |X|$ most significant bits of X.

We will treat n bit strings as polynomials of degree less than n with coefficients in $GF(2)$, thus they are elements of the field $GF(2^n)$. If X and Y are n bit strings then by $X \oplus Y$ we shall mean a addition in the field and by XY a multiplication in the field. The operation $X \oplus Y$ can be easily performed by a bitwise xor of X and Y and XY can be realized as a multiplication of the polynomials X and Y modulo a fixed polynomial $P(x)$ of degree n irreducible over *GF*(2). By xX we would represent the field multiplication of X by the polynomial x.

III. THE SCHEMES

In the subsequent subsections we discuss the constructions of the modes HCH, HCTR, XCB, EME and TET. A pictorial high level description of the five modes are provided in Fig. 1.

A. HCH

As mentioned earlier HCH falls under the category of Hash-Counter-Hash constructions. HCH uses an universal hash function of the form:

$$H_{R,Q}(A_1, \ldots, A_m) \;=\; Q \oplus A_1 \oplus A_2 R^{m-1} \oplus A_3 R^{m-2} \cdots \oplus A_{m-1} R^2 \oplus A_m R \tag{1}$$

Where $A_1, A_2, \ldots, A_m, R, Q$ are n bit strings. In addition to the hash function HCH requires a counter mode of operation. Given an n-bit string S, a sequence $S_1, \ldots, S_m$ is defined, where each S_i depends on S. Given such a sequence and a key K the counter mode is defined as follows.

$$\mathsf{Ctr}_{K,S}(A_1, \ldots, A_m) \;=\; (A_1 \oplus E_K(S_1), \ldots, A_m \oplus E_K(S_m)). \tag{2}$$

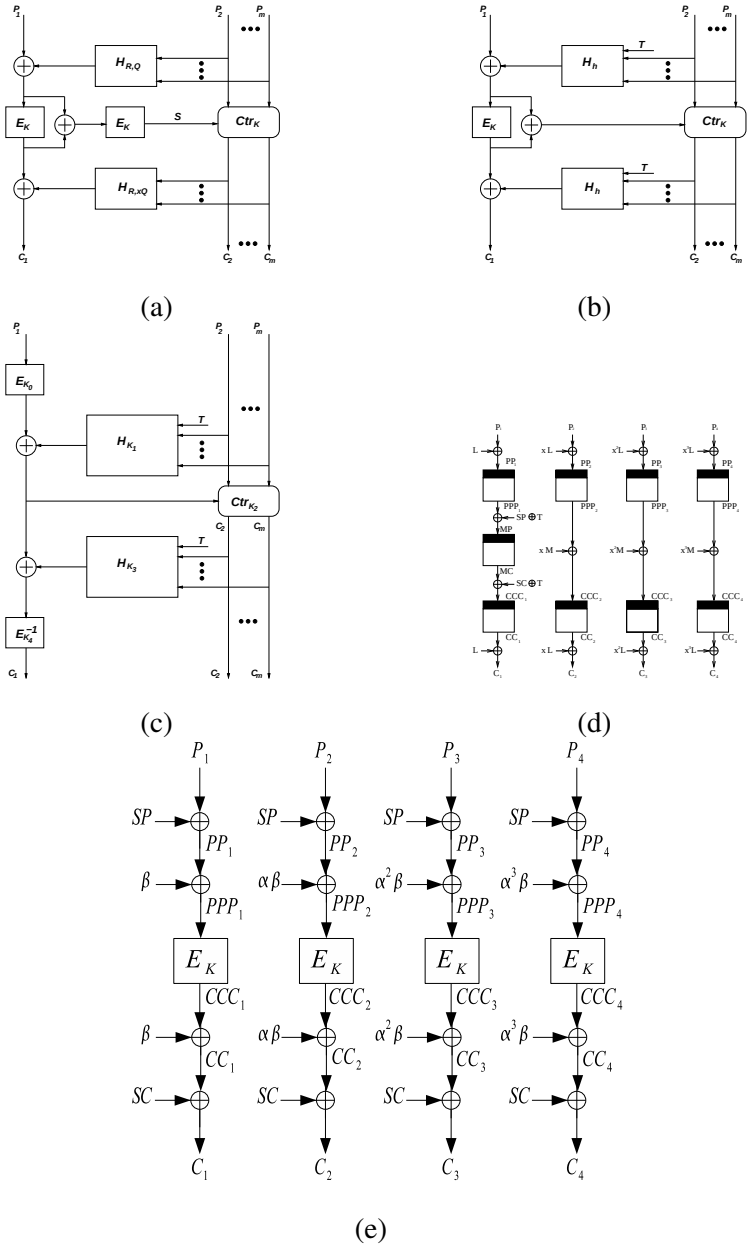


Fig. 1. High level block diagram of the schemes: (a) Encryption using HCH. Here $R = E_K(T)$ and $Q = E_K(R \oplus \mathsf{bin}_n(l))$, where l is the total length of the plain-text. (b) Encryption using HCTR. Here K is the key for the block cipher $E_K()$ and h is the key for the universal hash function $H_h()$. (c) Encryption using XCB, here 3 block cipher keys and 2 hash keys are derived using a single key K. (d) Encryption of 4 blocks of plaintext using EME. Here, $L = xE_K(0^n)$, $SP = PPP_2 \oplus PPP_3 \oplus PPP_4$, $M = MP \oplus MC$ and $SC = CCC_2 \oplus CCC_3 \oplus CCC_4$. (e) Encryption of four blocks using TET.

Fig. 2. Encryption and decryption using HCH. The tweak is T and the key is K. For $1 \leq i \leq m-1$, $|P_i| = n$ and $|P_m| = r$ where $r \leq n$.

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

Among various choices of the sequence S_i, in [5] the authors propose the use of $S_i = S \oplus \mathsf{bin}_n(i)$. In our implementations we also stick to this definition of the counter mode.

The complete encryption and decryption algorithm of HCH is given in Fig. 2. A schematic diagram of encryption is given in Fig. 1(a).

HCH can encrypt arbitrary long messages greater than n bits. It uses a single key which is same as the block-cipher key. It requires $m+3$ block cipher calls and $2m-2$ finite field multiplications to encrypt a m block message. The key for the universal hash is R, which is derived by encrypting the tweak. Thus R changes across encryption calls and this does not allow the use of pre-computations for computing the hash. HCH requires two passes over the data.

In [6] a modification of HCH is also proposed which is called HCHfp. HCHfp uses a different hash key which is not dependent on the tweak but is randomly chosen by the user. In HCHfp pre-computation can be possible. Another modification of HCH reported in [6] is HCHfp. HCHfp can only be used in those applications where the message length is fixed. This construction simplifies the general HCH mode and requires one less block-cipher call, but it requires two separate keys for the hash and the block-cipher. Thus, pre-computation for computing the hash is also possible in HCHfp. HCHfp is particularly of interest for disk encryption applications as here the message length is fixed and same as the sector length. The encryption-decryption algorithm using HCHfp is provided in Fig. 3.

All variants of HCH are provably secure and the authors guarantee that the advantage of any compu-

Fig. 3. Encryption and decryption using HCHfp. The tweak is T and the key is (K, α), where K is the block cipher key and α is the universal hash key. The number of blocks m is fixed.

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

tationally bounded chosen plaintext chosen ciphertext adversary in distinguishing HCH from a random permutation can be at most $O(\sigma_n^2)/2^n + \delta$ where σ_n denotes the number of n bit plaintexts and/or ciphertexts the adversary has access to, and δ denotes the advantage of an adversary to distinguish the underlying block-cipher from a random permutation.

B. HCTR

The structure of HCTR is similar to that of HCH with some important differences. The hash used in case of HCTR is defined as:

$$H_h(X) = X_1 h^{m+1} \oplus X_2 h^m \oplus \ldots \oplus \mathsf{pad}_{n-|X_m|}(X_m) h^2 \oplus \mathsf{bin}_n(|X|) h \tag{3}$$

Where h is the hash key and $X = X_1 || X_2 || \ldots || X_m ||$, where $|X_i| = n$ bits ($i = 1, 2, \ldots m-1$) and $|X_m| \leq n$.

The counter mode used in HCTR is the same as in (2). The encryption and decryption operations using HCTR are described in Fig. 4, and a high-level description is provided in Fig. 1(b).

HCTR can also encrypt arbitrary long messages. It requires m block cipher calls and $2m + 2$ field multiplications to encrypt an m block message. It requires two different keys and it is proved to be secure with a security bound of $O(\sigma_n^3)/2^n + \delta$. Thus it provides lesser security than HCH.

Fig. 4. Encryption and decryption using HCTR. The tweak is T and K is the block-cipher key and h the hash key.

Algorithm $\mathbf{E}^T_{K,h}(P_1,\ldots,P_m)$	**Algorithm** $\mathbf{D}^T_{K,h}(C_1,\ldots,C_m)$
1. $MM \leftarrow P_1 \oplus H_h(P_2\|\|\ldots\|\|P_m\|\|T)$;	1. $CC \leftarrow C_1 \oplus H_h(C_2\|\|C_3\|\|\ldots\|\|C_m\|\|T)$;
2. $CC \leftarrow E_K(MM)$;	2. $MM \leftarrow E_K^{-1}(CC)$;
3. $S \leftarrow MM \oplus CC$	3. $S \leftarrow MM \oplus CC$
4. $(C_2,\ldots,C_{m-1},C_m)$ $\leftarrow \mathsf{Ctr}_{K,S}(P_2,\ldots,P_m)$;	4. $(P_2,\ldots,P_{m-1},P_m)$ $\leftarrow \mathsf{Ctr}_{K,S}(C_2,\ldots,C_m)$;
5. $C_1 \leftarrow CC \oplus H_h(C_2\|\|C_3\|\|\ldots\|\|C_m\|\|T)$;	5. $P_1 \leftarrow MM \oplus H_h(P_2\|\|\ldots\|\|P_m\|\|T)$;
6. return $(C_1,\ldots,C_m)$.	6. return $(P_1,\ldots,P_m)$.

C. XCB

XCB is another mode which belongs to the hash-counter-hash family of constructions. XCB uses a hash $\boldsymbol{h}(h,X,T)$ which takes as inputs an n-bit hash key h along with two strings X and T. Let $X = X_1||\ldots||X_m$ and $T = T_1||\ldots||T_{m'}$ where $|X_i| = n$ for $i = 1,2,\ldots m$, and $|T_i| = n$ for $i = 1,2,\ldots m'$. The last block for both X and T can be incomplete. $\boldsymbol{h}_h(X,T)$ is defined as

$$\begin{aligned}\boldsymbol{h}_h(X,T) &= (X_1 h^{m+m'+1} + X_2 h^{m+m'} + \ldots \mathsf{pad}(X_m)h^{m'+2}) + \\ &\quad (T_1 h^{m'+1} + T_2 h^{m'} + \ldots \mathsf{pad}(T'_m)h^2) + (\ell(X)||\ell(T))h. \end{aligned} \tag{4}$$

Where $\ell(X)$ and $\ell(T)$ represent the $n/2$ bit binary representation of $|X|$ and $|T|$, respectively. The counter mode of XCB is defined as in (2), but they define $S_i = incr^i(S)$, where $incr(S)$ increments the last 32 bits of S modulo 2^{32}. The encryption and decryption using XCB is shown in Fig. 5. A highlevel description is provided in Fig. 1(c).

The variant of XCB that we present in Fig. 5, has no security proof. The authors suggested some minor modifications of the original proposal in [25] and provided a security proof for it. In terms of efficiency it seems that the two variants are not much different.

The variant of XCB that we implemented requires $2m + 4$ multiplications and $m + 6$ block-cipher calls for a single block tweak. The algorithm can work with arbitrary long messages less than 2^{39} bits. The modified version of XCB is shown to be secure with a quadratic security bound.

Fig. 5. Encryption and decryption using XCB. The tweak is T and K is the key from which two different hash keys and 3 different block-cipher keys are derived.

Algorithm $\mathbf{E}_K^T(P_1, \ldots, P_m)$	**Algorithm** $\mathbf{E}_K^T(P_1, \ldots, P_m)$
1. $K_0 \leftarrow 0^n$;	1. $K_0 \leftarrow 0^n$;
2. $K_1 \leftarrow 0^{n-1}\|\|1$;	2. $K_1 \leftarrow 0^{n-1}\|\|1$;
3. $K_2 \leftarrow 0^{n-2}\|\|10$;	3. $K_2 \leftarrow 0^{n-2}\|\|10$;
4. $K_3 \leftarrow 0^{n-2}\|\|11$;	4. $K_3 \leftarrow 0^{n-2}\|\|11$;
5. $K_4 \leftarrow 0^{n-3}\|\|100$;	5. $K_4 \leftarrow 0^{n-3}\|\|100$;
6. $A \leftarrow E_{K_0}(P_1)$;	6. $H_2 \leftarrow E_{K_4}(C_1)$;
7. $H_1 \leftarrow A \oplus \mathbf{h}_{K_1}(P_2\|\| \ldots \|\|P_m, T)$;	7. $H_1 \leftarrow H_2 \oplus \mathbf{h}_{K_3}(C_2\|\| \ldots \|\|C_m, T)$;
8. $(C_2, \ldots, C_{m-1}, C_m)$ $\leftarrow \mathsf{Ctr}_{K_2,H_1}(P_2, \ldots, P_m)$;	8. $(P_2, \ldots, P_m)$ $\leftarrow \mathsf{Ctr}_{K_2,H_1}(C_2, \ldots, C_m)$;
9. $H_2 \leftarrow H_1 \oplus \mathbf{h}_{K_3}(C_2\|\|C_3\|\| \ldots \|\|C_m, T)$;	9. $A \leftarrow H_1 \oplus \mathbf{h}_{K_1}(P_2\|\|P_3\|\| \ldots \|\|P_m, T)$;
10. $C_1 \leftarrow E_{K_4}^{-1}(H_2)$;	10. $P_1 \leftarrow E_{K_0}^{-1}(A)$;
11. return $(C_1, \ldots, C_m)$;	11. return $(P_1, \ldots, P_m)$;

D. EME

Now we will discuss a disk encryption mode called ECB-Mask-ECB (EME) [13]. As the name suggests, the mode consists of two electronic code-book layers with a masking layer in between.

The structure of EME is quite different from HCH and HCTR. EME falls under the category of Encrypt-mask-Encrypt constructions. It does not use any hash function, but instead uses two layers of encryption. The encryption and decryption algorithms are given in Fig. 6. A pictorial description of EME is given in Fig. 1(d).

EME requires $2m + 2$ block cipher calls for encrypting an m block message. It requires no multiplication. EME uses a single key same as the block-cipher key. EME has some message length restrictions. If the block length of the underlying block cipher is n then the message length should always be a multiple of n. Moreover, EME cannot encrypt more than n blocks of messages. This means that if an AES-128 is used as the underlying block-cipher then EME cannot encrypt more than 2048 bytes (2 KB) of data. This message length restriction was removed in a construction called EME* which requires more block-cipher calls than EME. But for the purpose of disk encryption EME is sufficient, as generally disk sectors lengths are less than 2KB and their lengths are multiples of 128 bits. EME has a security bound

Fig. 6. Encryption and Decryption using EME.

Algorithm EME.Encrypt$_K^T(P)$	**Algorithm** EME.Decrypt$_K^T(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $L \leftarrow xE_K(0^n)$	2. $L \leftarrow xE_K(0^n)$
3. **for** $i \leftarrow 1$ to m **do**	3. **for** $i \leftarrow 1$ to m **do**
4. $\quad PP_i \leftarrow x^{i-1}L \oplus P_i$	4. $\quad CC_i \leftarrow x^{i-1}L \oplus C_i$
5. $\quad PPP_i \leftarrow E_K(PP_i)$	5. $\quad CCC_i \leftarrow E_K^{-1}(CC_i)$
6. **end for**	6. **end for**
7. $SP \leftarrow PPP_2 \oplus PPP_3 \oplus \ldots PPP_m$	7. $SC \leftarrow CCC_2 \oplus CCC_3 \oplus \ldots CCC_m$
8. $MP \leftarrow PPP_1 \oplus SP \oplus T$	8. $MC \leftarrow CCC_1 \oplus SC \oplus T$
9. $MC \leftarrow E_K(MP)$	9. $MP \leftarrow E_K^{-1}(MC)$
10. $M \leftarrow MP \oplus MC$	10. $M \leftarrow MP \oplus MC$
11. **for** $i \leftarrow 2$ to m **do**	11. **for** $i \leftarrow 2$ to m **do**
12. $\quad CCC_i \leftarrow PPP_i \oplus x^{i-1}M$	12. $\quad PPP_i \leftarrow CCC_i \oplus x^{i-1}M$
13. **end for**	13. **end for**
14. $SC \leftarrow CCC_2 \oplus CCC_3 \oplus \ldots CCC_m$	14. $SP \leftarrow PPP_2 \oplus PPP_3 \oplus \ldots PPP_m$
15. $CCC_1 \leftarrow MC \oplus SC \oplus T$	15. $PPP_1 \leftarrow MP \oplus SP \oplus T$
16. **for** $i \leftarrow 1$ to m **do**	16. **for** $i \leftarrow 1$ to m **do**
17. $\quad CC_i \leftarrow E_K(CCC_i)$	17. $\quad PP_i \leftarrow E_K(PPP_i)$
18. $\quad C_i \leftarrow x^{i-1}L \oplus CC_i$	18. $\quad P_i \leftarrow x^{i-1}L \oplus PP_i$
19. **end for**	19. **end for**
20. **return** $C_1, C_2, \ldots, C_m$	20. **return** $P_1, P_2, \ldots, P_m$

of $O(\sigma^2)/2^n + \delta$.

E. TET

TET uses two layers of block-wise invertible universal hash functions with an ECB layer of encryption in-between them. To compute the hash function, TET requires a hash key τ which requires certain properties. To ensure invertibility of the hash function, τ must be such that for a m block message $\sigma = \sum_{i=1}^{m} \tau^m \neq 0$. For this to be true one requires different hash keys for different message lengths. The authors propose a way to generate the hash key τ for different messages using a key. The encryption algorithm also requires the value of σ^{-1}. This makes TET rather complicated for applications which requires encryption of variable length messages. As our target application will use only fixed length

Fig. 7. Encryption and Decryption using TET.

Algorithm TET.Encrypt$^{T}_{K_1,K_2,\tau,\sigma}(P)$	**Algorithm** TET.Decrypt$^{T}_{K_1,K_2,\tau,\sigma}(C)$
1. Partition P into $P = P_1, ..., P_m$	1. Partition C into $C = C_1, \ldots, C_m$
2. $L \leftarrow \|P\|$;	2. $L \leftarrow \|C\|$;
3. $X \leftarrow E_{K_1}(L)$;	3. $X \leftarrow E_{K_1}(L)$;
4. $\beta \leftarrow E_{K_1}(T \oplus xX)$;	4. $\beta \leftarrow E_{K_1}(T \oplus xX)$;
5. $SP \rightarrow 0^n$; $SC \leftarrow 0^n$;	5. $SP \rightarrow 0^n$; $SC \leftarrow 0^n$;
6. **for** $i \leftarrow 1$ to m	6. **for** $i \leftarrow 1$ to m
7. $SP \leftarrow (SP \oplus P_i)\tau$;	7. $SC \leftarrow (SC \oplus C_i)\tau$;
8. **end for**	8. **end for**
9. $SP \leftarrow SP\sigma^{-1}$	
10. **for** $i \leftarrow 1$ to m	10. **for** $i \leftarrow 1$ to m
11. $PP_i \leftarrow P_i \oplus SP$;	11. $CC_i \leftarrow C_i \oplus SP$;
12. $PPP_i \leftarrow PP_i \oplus \alpha^{i-1}\beta$;	12. $CCC_i \leftarrow CC_i \oplus \alpha^{i-1}\beta$;
13. $CCC_i \leftarrow E_{K_2}(PPP_i)$;	13. $PPP_i \leftarrow E^{-1}_{K_2}(PPP_i)$;
14. $CC_i \leftarrow CCC_i \oplus x^{i-1}\beta$	14. $PP_i \leftarrow PPP_i \oplus x^{i-1}\beta$
15. $SC \leftarrow (SC \oplus CC_i)\tau$	15. $SP \leftarrow (SP \oplus PP_i)\tau$
16. **end for**	16. **end for**
17. $SC \leftarrow SC\sigma^{-1}$;	
18. **for** $i \leftarrow 1$ to m	18. **for** $i \leftarrow 1$ to m
19. $C_i \leftarrow CC_i \oplus SC$;	19. $P_i \leftarrow PP_i \oplus SP$;
20. **Return** $C_1, ..., C_m$;	20. **return** $P_1, ..., P_m$;

messages, we present the TET algorithm only for fixed lengths which are multiples of the block length of the block-cipher. Also we assume a fixed value of τ and pre-computed values of σ^{-1}. The algorithm for TET is shown in Figure 7. A high level description is provided in Fig 1(e). Note, that the encryption and decryption operations in TET are quite different. Compared with TET decryption operation, the encryption operation requires two extra multiplications by σ^{-1}.

Some of the characteristics of the five modes of operations discussed above are summarized in Table I.

TABLE I

SUMMARY OF THE CHARACTERISTICS OF THE FIVE TES DESCRIBED. THE FIGURES ARE CONSIDERING A FIXED LENGTH m BLOCK MESSAGE.

Mode	BC Calls	Field Mult.	no. of keys
HCH	$m+3$	$2(m-1)$	1
HCHfp	$m+2$	$2(m-1)$	2
HCTR	m	$2(m+1)$	2
XCB	m+6	$2(m+1)$	1
EME	$2(m+1)$	0	1
TET	$m+2$	2m + 2	3

IV. DESIGN DECISIONS

For implementing all five schemes we chose the underlying block cipher as AES-128. As it was mentioned in the Introduction, the designs that we present here are directed towards the application of disk sector encryption. In this specific application the messages are all of fixed length and we consider them to be multiples of 128 bits. In particular, our designs are optimized for applications where the sector length is fixed to 512 bytes. As the sector address is considered to be the tweak, thus the tweak length itself is considered to be fixed and equal to one block length of the block cipher.

The speed of a low level disk encryption algorithm must meet the current possible data rates of disk controllers. With emerging technologies like serial ATA and Native Command Queuing (NCQ) the modern day disks can provide data rates around 3Giga-bits per second[32]. Thus, the design objective should be to achieve an encryption/decryption speed which matches this data rate.

The modes HCH, HCTR, XCB and TET use two basic building blocks, namely, a polynomial universal hash and the block-cipher. EME requires only a block-cipher. Since AES-128 was our selection for the underlying block-cipher, proper design decisions for the AES structure must meet the desired speed. Out of many possible designs reported in the literature [19], [9], [3], [14], [8] we decided to design the AES core so that a 10-stage pipeline architecture could be used to implement the different functionalities of the counter mode, the electronic code book (ECB) mode and the encryption of one single block that we will call in the rest of this paper *single mode*.

This decision was taken based on the fact that the structure of the AES algorithm admits to a natural ten-stage pipeline design, where after 11 clock cycles one can get an encrypted block in each subsequent clock-cycle. It is worth mentioning that in the literature, several efficient designs with up to 70 pipeline

stages have been reported [16], but such designs would increase the latency, i.e., the total delay before a single block of cipher-text can be produced. As the message lengths in the target application are specifically small (in particular the most used sector size is of 512 bytes), such pipeline designs are not suitable for our target application.

The main building block needed in the polynomial hash included in the specification of the HCH, HCTR, XCB and TET modes, is an efficient multiplier in the field $GF(2^{128})$. Out of many possible choices we selected a fully parallel Karatsuba-Ofman multiplier which can multiply two 128-bit strings in a single clock-cycle at a sub-quadratic computational cost [30]. This time efficient multiplier occupies about 1.4 times the hardware resources required by one single AES round.[2]. Because of this, the total hardware area of EME (which does not require multipliers) is significantly lesser than that required by the other modes that we study. A more compact multiplier selection would yield significantly lower speeds which violates the design objective of optimizing for speed.

It is noted that the specifications of HCTR, XCB, TET and HCHfp algorithms imply that one multiplicand is always fixed, thus allowing the usage of pre-computed look up tables that can significantly speed up the multiplication operation. Techniques to speed up multiplication by look-up tables are discussed in [33], [23], [27], [2] for the software platform scenario. These techniques can be somehow be extended to hardware implementations also. However, there is a tradeoff in the amount of speed that can be obtained by means of pre-computation and the amount of data that needs to be stored in tables. Significantly higher speeds can be obtained if one stores large tables. This speedup thus comes with an additional cost of area and also the potentially devastating penalty of secure storage. Moreover, if pre-computation is used in a hardware design then the key needs to be hardwired in the circuit which can lead to numerous difficulties in key setup phases and result in lack of flexibility for changing keys. Because of the above considerations, we chose not to store key related tables for our implementations. Thus the use of an efficient but large multiplier is justified in the scenario under analysis. [3] Regarding storage of key related materials we make an exception to this in case of TET. TET requires computation of a inverse, which is a particularly expensive arithmetic operation. In case of TET we store the hash key τ along with the pre-computed value of σ^{-1}, this storage helps us to get rid of a field inversion circuit but does not help us to speed up multiplications.

We implemented the schemes on a FPGA device which operates at lower frequencies than true VLSI

[2]For specific experimental details see Table II.

[3]The same design decision was taken in [23].

Fig. 8. The Horner's Rule

Algorithm $\mathbf{HORN}_h(X_1, \ldots, X_m)$
$Y \leftarrow 0^n$;
for $i = 1$ to m,
  $Y \leftarrow (Y \oplus X_i)h$;
end for
return Y

circuits. Thus the throughput that we obtain probably can be much improved if we use the same design strategies on a CMOS VLSI technology. Our target device was a XILINX Virtex 4, xc4v1x100-12FF1148.

V. The Design Overviews

In this Section we give a careful analysis of the modes' data dependencies and we explain how the parallelism present in the algorithms can be exploited. In the analysis which follows we assume the message to be of 512 bytes (32 AES blocks). Furthermore, we assume a single AES core designed with a 10 stage pipeline and a fully parallel single clock cycle multiplier. We also calculate the key schedules for AES on the fly, this computation can be parallelized with the AES rounds. The polynomial universal hash functions are computed using the Horner's rule shown in the algorithm of Fig. 8:

A. HCH mode of Operation

Referring to the Algorithm of Fig. 2 the algorithm starts with the computation of the parameter R in Step 1. For computing R the AES pipeline cannot be utilized and must be accomplished in simple mode, implying that 11 clock cycles will be required for computing R. At the same time, the AES round keys can be computed by executing concurrently the AES key schedule algorithm. The hash function of Step 3 can be written as

$$H_{R,Q}(P_1, P_2, \ldots P_{32}) = P_1 \oplus Q \oplus Z$$

where $Z = \mathbf{HORN}_R(P_2, \ldots P_{32})$. So, Z and Q can be computed in parallel. For computing Z, 31 multiplications are required and computation of Q takes 11 clock cycles. So the computation of the hash in step 2 takes 31 clock cycles. Then, the computation of Step 4 requires two simple mode encryption which implies 22 more clock cycles. So we need to wait 64 clock cycles before the counter mode starts.

The counter mode in step 5 requires 31 blockcipher calls which can be pipelined. So computation of step 5 requires a total of $30 + 11 = 41$ clock cycles. The first cipher block C_2 is produced 11 clock cycles after the counter starts. The second hash function computation of Step 7 can start as soon as C_2 is available in the clock cycle 75. Hence the computation of the hash function can be completed at the same time that the last cipher block (C_m) of Step 5 is produced. Figure 9(a) depicts above analysis. It can be seen that a valid output will be ready after the cycle 75 and a whole disk sector will be ready in the cycle 106. In case of HCHfp the computation of Q is not required, and it uses a hash key which is different from R. Thus R and the hash function can be computed in parallel, which gives rise to a savings of 11 clock cycles. So HCHfp will produce a valid output in 64 clocks and it will take 95 clock-cycles to encrypt the 32 block message (see Figure 9(b)).

B. HCTR Mode of Operation

Referring to the Algorithm of Fig. 4, the computation of the hash function of Step 1 requires 33 clock cycles. At the same time, the design can derive the AES round keys by executing concurrently the AES key schedule algorithm. Then, the computation of the parameter CC in Step 2, must be accomplished in simple AES mode, implying that 11 clock cycles will be required for completing that calculation. As in HCH mode, the $m - 1$ block cipher calls included in Step 4 are performed in counter mode, which once again can be computed in parallel via the pipeline architecture. Hence, the computation of all the C_i for $i = 2, \ldots, m = 32$, can be computed in $(32 - 1) + 11 = 42$ clock cycles. At the same time, the second hash function computation of Step 7 can start as soon as C_2 is available in the clock cycle 56. Hence the computation of the hash function can be completed at the same time that the last block cipher (C_m) of Step 5 is produced. Figure 9(c) depicts the timing analysis just given. It can be seen that a valid output will be ready after the cycle 55 and a whole disk sector will be ready in the cycle 88.

C. XCB Mode of operation

The computation of XCB starts with the derivation of the five keys. Derivation of each key requires a block-cipher call. These five block-cipher calls can be parallelized thus requiring 15 clock cycles. The computation of the first hash requires 33 clock cycles and in the mean time the computation of the two key schedules and A can be completed. The computation of the first hash which require 33 clock cycles can thus be completed within clock-cycle 49. Then the counter mode starts, which requires 41 clock cycles to complete. The second hash can start at clock cycle 59 and thus it is completed in clock-cycle 93. For computing the last block, an AES decryption call is required with a different key. Hence, completing

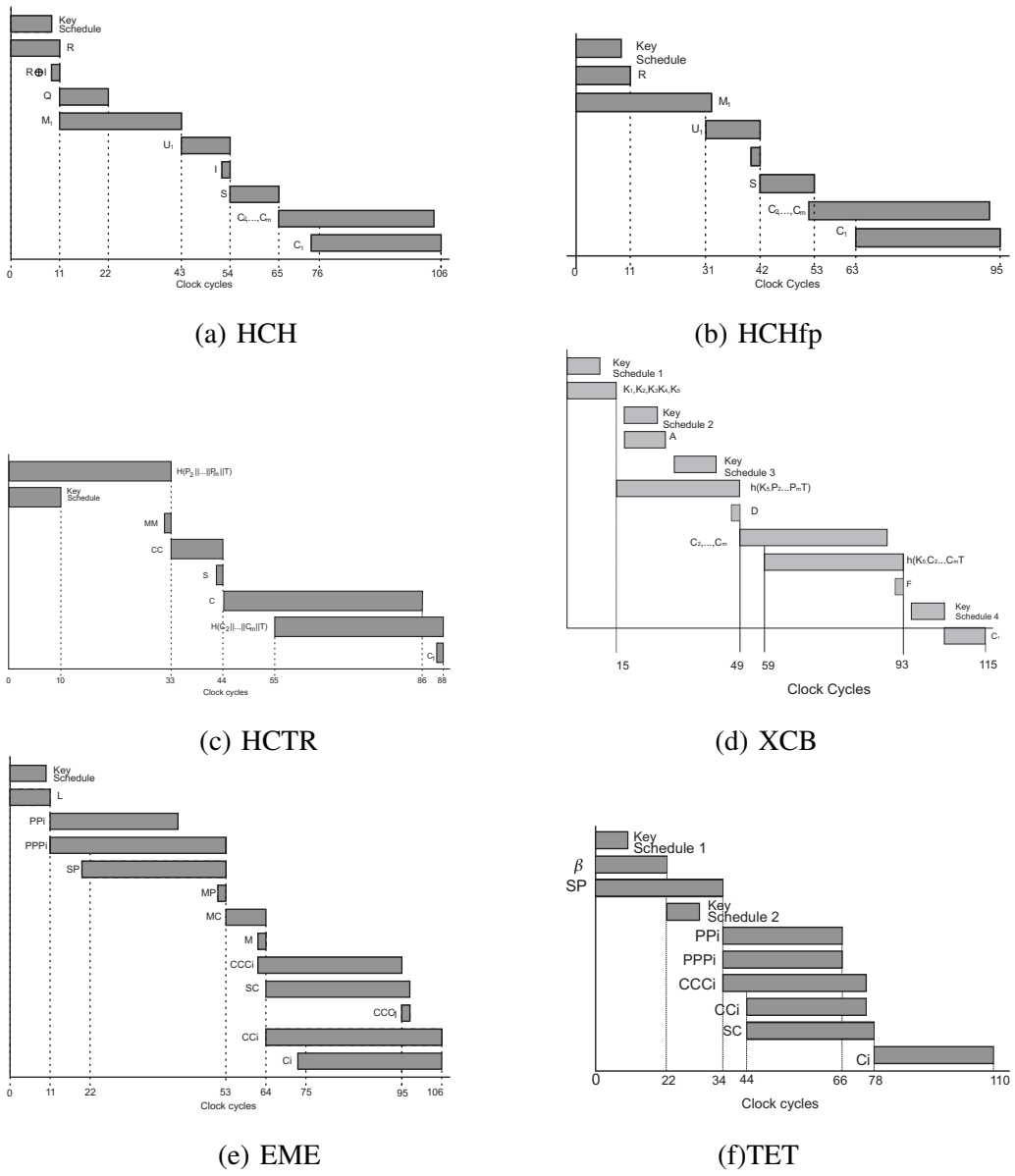
the key schedule and the decryption requires another 21 clock cycles, and a reset operation is necessary before computing the new key schedule. This thus gives rise to a requirement of 115 clock cycles to encrypt the whole sector. But the first cipher block would be ready in case of XCB after clock cycle 59.

D. EME mode of Operation

Referring to the Algorithm of Fig. 6, the computation of the parameter L in Step 2, must be accomplished in a sequential fashion, implying that 11 clock cycles will be required for completing that calculation. Thereafter, the 32 block cipher calls included in Steps 3-6 can be accomplished using the benefits of the parallelism associated to the pipeline approach. So, the computation of all the PPP_i for $i = 1, 2, \ldots, m = 32$, can be computed in $(32 - 1) + 11 = 42$ clock cycles. On the contrary, the cipher call in Step 9 for obtaining MC must be performed in a sequential fashion, which implies 11 extra clock cycles. The second layer of encryption can also be performed in 42 clock cycles and the operations $x^i M$ and $x^i L$ in steps 12 and 18 can be parallelized with the block-cipher calls. So to complete encryption of 32 blocks EME should require $11 + 42 + 11 + 42 = 106$ clock cycles. And the first block of valid output would be produced after 75 clock cycles. Some pre-computations may save some of the EME costs. For example, L in Step 2 is a quantity only dependent on the key K, thus, L can be pre-computed yielding the saving of 11 clock cycles. But this will require storage of key materials. Figure 9 (e) shows the EME operations that are suitable for being computed in parallel.

E. TET mode of Operation

Referring to the algorithm of Fig. 7, the computation of the parameter β requires two AES calls in the simple mode, which can be accomplished in 22 clock cycles. The computation of SP can be done in parallel. Computation of SP will require 32 multiplications which can be completed in 33 clock cycles. In encryption it requires an extra multiplication with σ^{-1}, Thus the computation of SP would be complete in 34 clock cycles. In the mean-time the key schedule for the second block-cipher key can also be completed. The computation of PP_i and PPP_i can be parallelized and they can be computed in 33 clock cycles. As soon as PPP_1 is available (which would be available at clock cycle 35), computation of CCC_i can start. Computation of CCC_i, $i = 1, \ldots, 32$ will take a total of 32 block cipher calls which can be completed in 42 (32 + 10) clock cycles. Thus, after clock-cycle 78 all CCC_i s would be ready. The computation of the final cipher texts C_i-s would take another 32 clock cycles. Thus, the whole disk would be ready after 110 clock cycles. And the first cipher block would be ready after 79 clock



(a) HCH (b) HCHfp

(c) HCTR (d) XCB

(e) EME (f)TET

Fig. 9. Timing diagrams

cycles. Note that in this analysis we do not consider computation of the inverse, which may give rise to a significant increase in the number of the required clock cycles.

The analysis that we just outlined is summarized in Figure 10, where the throughput predictions for all the five TES modes considered in this work are included. As an example, if we can design an architecture for HCHfp running at a clock frequency of 100 MHz, then, the algorithm analysis will predict a throughput of 4.27 Gbps, whereas the same clock frequency will render just 3.69 Gbps for TET.

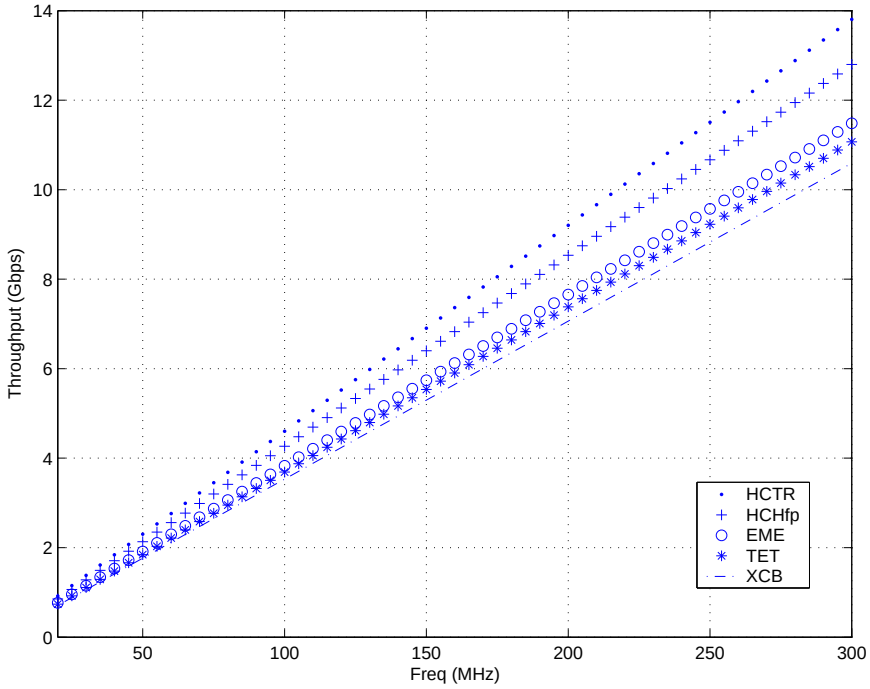


Fig. 10. Throughput Vs. Frequency for the Five TES

The above analysis of the algorithms that we presented is independent of any implementation strategy. It seems that the number of clock cycles required for each of the algorithms cannot be further reduced if the design decisions taken by us are followed (10 stage AES pipeline and 1 clock cycle multiplier). Pre-computation in the multiplier or other multiplication strategies can bring down the cost of computing the hash function in HCHfp, HCTR and TET.

VI. Implementation Aspects

In practice, the timing performance of a generic hardware design will be determined by its associated critical path. As for the area utilization of a given design, some of the factors that have impact in hardware resource utilization include: the number of temporary variables involved in the design (which implies extra registers) and the number of possible inputs that the main building blocks may have (which translates in extra multiplexer blocks).

In order to give a rough estimation of the critical path associated to each one of the modes, we show in Table II the performance of the architectures' main building blocks, namely, a key generation/encryption/decryption AES round, an encryption-only AES round, and a 128-bit fully-parallel

TABLE II

PERFORMANCE OF AN ENCRYPTION/DECRYPTION AES ROUND AND A 128-BIT FULLY-PARALLEL KARATSUBA-OFMAN MULTIPLIER

Design	**Slices**	**B-RAM**	**Critical Path**(nS)
Full AES round	1215	8	10.998
Encryption-only AES round	298	8	6.71
multiplier	3223	-	9.85

Karatsuba-Ofman multiplier. [4]

Considering the utilization of B-RAMs and slices, the size of one full AES round in our design is about 30% smaller than that of the Karatsuba-Ofman multiplier. However, the critical path delay associated to an encryption/decryption AES round, is longer than that of the multiplier block by about 10%.

TABLE III

HARDWARE RESOURCES UTILIZED BY THE FIVE TES

Modo	Mux inAES	Extra B-RAM	Mul $xtime$	Registers 128 bits	Mux 2×128	Mux 3×128
HCTR	3×128	-	-	3	2	1
HCHfp	4×128	-	1	5	5	-
HCH	5×128	-	1	6	5	-
EME	4×128	2	3	5	5	-
TET	3×128	2	3	2	4	-
XCB	4×128	-	-	8	6	2

Table III shows the hardware resource usage by each of the five TES modes of operation. Besides an obvious impact in the area complexity of the modes, the resources occupied by each TES tend to increase its critical path. In the rest of this Section, we briefly analyze the resource utilization and timing potential performance of the five TES modes under analysis.

[4]The experimental results shown in Table II correspond to a place &route simulations using Modelsim XE III 6.0d and Xilinx ISE 8.3 and a Xilinx Virtex4 XC4VLX100-12FF1148 FPGA as a target device.

A. HCH and HCHfp

For HCH and HCHfp the critical path will be also lower bounded by the minimum critical path between the AES core and the hash function. Considering the values given in Table II the maximum throughputs that we can expect for these two modes when using the full and the encryption-only AES cores is 3.8 Gbps and 4.24 Gbps, respectively. However, we should expect that HCH and HCHfp timing performances will be appreciably lower than those bounds, because these modes requires six and five extra registers for temporary variables, respectively. Moreover as shown in Table III, the possible inputs for the AES core and the hash function is more than that of HCTR, which will force us to use multiplexer blocks with more inputs.

B. HCTR

As shown in Table III, HCTR requires three extra 128-bit registers in order to allocate temporary computation values and also the possible inputs for the AES core and the hash function is three. This feature makes HCTR both, a fast an economical TES mode. The critical path of HCTR will be lower bounded by the one associated to either the hash function or the AES core, whichever is larger. Therefore, and according to the critical paths reported in Table II, we can expect that an implementation of HCTR will have a critical path of at least 10.998ηS when using the full AES core and 9.85 ηS when using the encryption-only AES round. In terms of throughput, this translates to a maximum of 4.185 Gbps and 4.672 Gbps, respectively.

C. XCB

Out of the five modes analyzed, XCB is both, the most expensive in terms of hardware resource utilization, and the slowest. XCB's latency however, is relatively low but the total time required is quite high. Among other factors, XCB's total time is high because in its final step the calculation of E_K^{-1} cannot start till a key schedule computation of C_1 has finished. In total, 21 clock cycles are consumed in that final step. As shown in Table III, storage of XCB temporary variables requires eight extra 128-bit registers and one 128-bit four-to-one multiplexer. Thus, the maximum throughput that one can expect for XCB when using the full and the encryption-only AES cores should be significantly lower than 3.21 Gbps and 3.59 Gbps, respectively.

D. EME

In the case of EME, its most notorious building block is the AES core. However, other smaller components that have some impact in the EME performance are the *xtime* multiplication algorithm

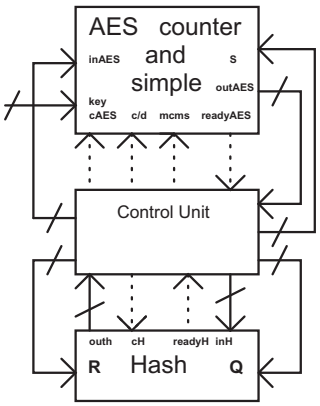


Fig. 11. HCH Main Architecture

along with the chain additions characteristic of this mode. [5] Since the *xtime* operation can be performed efficiently in hardware, the critical path of EME is mainly given by that of the AES core utilized. Hence, we can expect that an implementation of EME will have a critical path of at least 10.998ηS when using the full AES core and 6.71 ηS when using the encryption-only AES round. In terms of throughput, this translates to a maximum of 3.48 Gbps and 5.71 Gbps, respectively. Regarding area utilization, EME is consistently the most economical TES mode. However, the computation of EME requires to store all the PPP_i values for $i = 2, \ldots, m$. This issue was solved by utilizing two extra FPGA block RAMs as reported in Table III.

E. TET

Once again, the critical path of TET will be lower bounded by the minimum critical path between the AES core and the hash function. According to Table II, the maximum throughput that we can expect for TET when using the full and the encryption-only AES cores is 3.36 Gbps and 3.75 Gbps, respectively. As reported in Table III, the computation of TET requires the allocation of two extra FPGA block RAMs. Moreover, two 128-bit registers and four 128-bit two-to-one multiplexers blocks are required. For HEH [31] which is an improvement over TET, the same resources would be required. But HEH does not require computation of the inverse, which is a good advantage over TET.

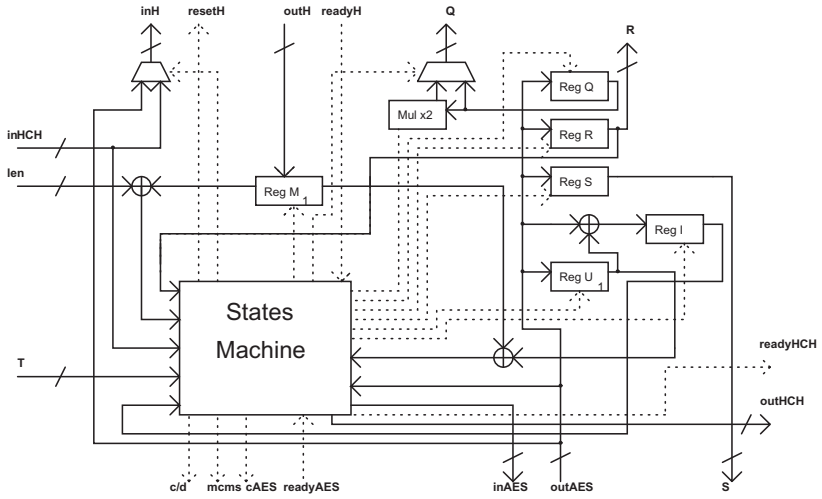


Fig. 12. HCH Control Unit Architecture

VII. Architecture of HCH

As a representative design example we shall discuss the architecture of HCH in details. The architectures for the other modes are quite similar and we do not discuss them here.

A. HCH Implementation

Fig. 11 shows the general architecture of the HCH mode of operation. It can be seen that AES must be implemented both, in counter and in simple mode. Moreover a hash function is also required as one of the main building blocks. Design details of both, the AES core and the hash function can be found in the Appendix.

The architecture operation is synchronized through a control unit that performs the adequate sequence of operations in order to obtain a valid output.

The HCH control unit architecture is shown in Fig. 12. It controls the AES block by means of four 1-bit signals, namely: **cAES** that initializes the round counter, the **c/d** signal that selects between encryption or decryption mode, the **msms** signal that indicates whether one single block must be processed or rather, multiple blocks by means of the counter mode. Finally, **readyAES** indicates whenever the architecture has just computed a valid output. The AES dataflow is carried out through the usage of three 128-bit busses, namely, **inAES** that receives the blocks to be encrypted, **outAES** that sends the encrypted blocks

[5]For the EME mode specifics see the algorithm in Fig.6 and its schematic architecture in Fig.1(d).

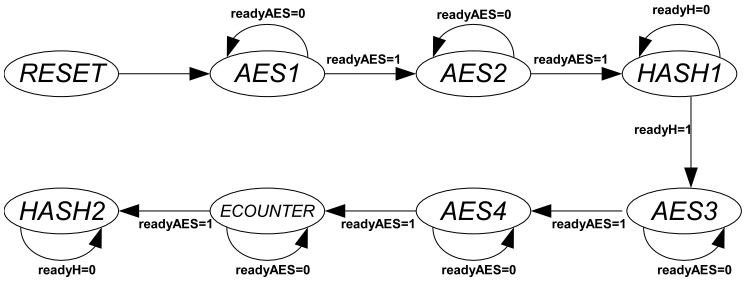


Fig. 13. HCH State Machine Diagram

and **S** that receives the initialization parameter for the counter mode. The communication with the hash function block is done using two signals: **cH** for initializing the accumulator register and the counter of blocks already processed and **readyH** that indicates that the hash function computation is ready. The data input/output is carried by the **inH** and **outH** busses, respectively. The parameters **R** and **Q** are calculated in the control unit and send through the busses to the hash function.

The HCH control unit implements a finite state automaton that executes the HCH sequence of operations. It defines the following eight states: *RESET*, *AES1*, *AES2*, *HASH1*, *AES3*, *AES4*, *ECOUNTER* and *HASH2*. In each state, an appropriate control word is generated in order to perform the required operations. The correct algorithm execution requires storing the **R**, **Q**, **S**, **I**, $\mathbf{U}_1$ and $\mathbf{M}_1$ values. Thus, six extra 128-bit registers are needed. In particular the hash function input **inH** can come from the system input or from the output of the AES counter mode. Therefore, a multiplexer is needed for addressing the correct input, where the multiplexer signals are handled by the state machine's control word. We compute the **xQ** signal by means of an *xtimes* operation in the finite field GF(2^{128}), which was implemented as described in for example [28], [7].

The sequence of operations described in Algorithm 2 is performed through the execution of the state machine diagram shown in Figure 13. The state transition among states is controlled by two signals, namely, **readyAES**, which indicates that the current output in the bus **outAES** is valid and; **readyH** that indicates that the computation of the hash function has just been completed.

In *RESET* the AES key schedule process starts and the value in T is assigned to **inAES**. Then, the control is transferred to *AES1*. In *AES1*, the value $R = E_k(T)$ is computed concurrently with the key generation process and when the signal **readyAES** becomes active the value in **outAES** is transferred to the register **regR**, the round counter is re-initialized, the value $R \oplus len$ is computed and the control is

transferred to the state *AES2*. In *AES2* the value $Q = E_k(R \oplus len)$ is computed and at the same time the computation of the hash value $\mathbf{M}_1$ starts. When the signal **readyAES** becomes active the round counter is re-initialized, the value in **Q** is transferred to the register **regQ**, **xQ** is computed and the automaton switches to state *HASH1*. In state *HASH1* the computation of the hash value $\mathbf{M}_1$ is completed, and when the signal $readyH$ is active the hash result is stored in the register $\mathbf{M}_1$ and the automaton transitions to *AES3*. In the state *AES3* the value $U_1 = E_k(M_1)$ is computed, and when the signal **readyAES** becomes ready, the value $I = M_1 \oplus U_1$ is computed and stored in **RegI** and the control is transferred to the state *AES4*. In *AES4*, the value $S = E_k(I)$ is computed and when the signal **readyAES** is ready that value is stored in **RegS** and then we arrive to the state *ECOUNTER*. In the state *ECOUNTER* the AES multiple block encryption in counter mode starts, when the signal **readyAES** is activated the hash function initiates and the automaton switches to the state *HASH2*. When in state *HASH2*, the encryption of $C_2, \ldots, C_m$ in counter mode is performed in parallel with the computation of the hash function. Finally, when the signal *readyH* is activated we have the hash result in C_1.

En each one of the states mentioned above, a 14-bit control word $cword$ that orchestrates all the architecture modules is continuously updated. The organization of the control word is summarized in Table IV.

The dataflow for encryption and decryption is essentially the same. The only modification is to determine whether **Q** or **xQ** should be used in the two hash function calls. This decision is taken in our architecture with the help of a multiplexer block whose output is the input hash signal **Q** (se Fig. 13). In the first call to the hash function block, and when the state is *HASH1* and the mode signal is off that multiplexer selects **Q**, otherwise, it selects **xQ**. In the second hash function call, **xQ** is selected if the automaton is in the state *HASH2* and the mode signal is off, otherwise, it selects **Q**.

1) Fixed Length HCH: In the specific case of the disk encryption application, it is known in advance that the plaintext messages will be of exactly 512 bytes. Taking advantage of this fact, we can optimize the implementation of the HCH mode of operation even further as shown in the algorithm of Fig. 3. The modification implies a total saving of 11 clock cycles as is shown in Fig. 9(b). The initial encryption of R is omitted since this parameter is substituted by a second key, while Q is substituted by R α. This modification also implies a saving in area resources as it will be seen in the next Section.

VIII. RESULTS

In this Section we present the experimental results obtained from our implementations. We measure the performances of our designs based on the following criteria: *total time* required for encrypting 32 blocks

TABLE IV

CONTROL WORD SPECIFICATION PERMUTATION.

Control Word bits	**Functionality**
$cword_0$	Synchronizes the input dataflow
$cword_1$	Indicates whether the hash function input comes from the system input or rather, from the output of the AES in counter mode
$cword_2$	hash function reset
$cword_4$	round counter reset
$cword_5$	AES in simple or counter mode
$cword_6$	AES counter mode ready for computing a new plaintext
$cword_7$	Load signal for register **R**
$cword_8$	Load signal for register **Q**
$cword_9$	Load signal for register $\mathbf{U}_1$
$cword_{10}$	Load signal for register $\mathbf{M}_1$
$cword_{11}$	Load signal for register **S**
$cword_{12}$	Determines if the AES will be working according to the stipulated in the input signal or in encryption mode
$cword_{13}$	Load signal for register **I**

of data; *latency*, i.e., the elapsed time needed to produce the first output block; the size of the circuit in slices; the number of B-RAMs used and the *throughput*. The performance/area tradeoff is evaluated using the Throughput per Area (TPA) metric, which is computed as,

$$TPA = [(slices + 128 \cdot BRAMS) \cdot \text{Total\_Time}]^{-1} .$$

Where a higher TPA coefficient indicates higher design efficiency, i.e., a better performance/area tradeoff.

The rest of this Section is organized as follows. In Subsection §VIII-A we report the area and timing performance obtained in the implementation of the main building blocks required for the implementation of the TES modes. Then, Subsection §VIII-B reports the performance achieved by the five TES modes analyzed in this paper. The comparison was carried out using a pipelined full AES core, a sequential AES core and a pipelined encryption-only AES core. Finally, in Subsection §VIII-C, we give a speculative comparison of our EME design against the best software implementation reported in the open literature for the AES cipher.

TABLE V

PERFORMANCE OF THE AES AND HASH IMPLEMENTATIONS

Method	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Throughput (Gbps)	TPA
Full-core AES-Sequential	1301	18	81.97	10	1.05	2273.7
Full-core AES-Pipeline	6368	85	83.88	1	10.74	4863.17
Encryption-Only AES-Pipeline	2468	85	149.00	1	19.07	11162.7
Hash function	4014	-	101.45	1	13.00	25274.0

A. Main Building Blocks

The two building blocks shown in Table II, were used for implementing a full AES core (i.e., an encryption/decryption/key generation core) in sequential and pipeline architecture; an encryption-only pipeline AES core and; a hash function for the HCH an the HCTR modes. Table V summarizes the performance obtained in the implementation on those blocks. The sequential AES gives significantly poor throughput and TPA, while the hash function has better throughput and TPA than the full AES-pipeline but smaller throughput than the encryption-only AES core.

B. Performance Comparison of the six TES Modes

In Table VI the experimental results for the six modes of operation implemented with an underlying full pipelined AES core are shown. Note that the number of clock-cycles reported in Table VI are one more than those estimated in Section V, this is because in the true implementations one clock cycle is lost due to the initial reset operation. The critical path of the designs shown in Table VI is mainly determined by the AES core, which as it was shown in Table V, has a longer path the hash function utilized in all five HCTR, HCH, HCHfp, TET and XCB modes.

From Table VI it is evident that EME is the most economical mode in terms of area resources, mainly due to the fact that this mode does not utilizes a hash function. Hence, the critical path of EME is given by the AES full core plus a chain of three layers of additions. Out of the five modes analyzed, XCB is both, the most expensive in terms of hardware resource utilization, and the slowest. HCHfp and HCH require more hardware resources than HCTR. TET performs about the same than HCHfp assuming that the parameter σ^{-1} has been previously precomputed.

In terms of speed, the fastest mode is HCTR since it only utilizes one AES block cipher call in sequential mode, whereas HCHfp requires a total of four such calls (although only three have conse-

quences in terms of clock cycles since the other is masked with the computation of the hash function). In this scenario, HCHfp is better than HCH, EME, TET and XCB in terms of speed. Finally, in terms of efficiency, the highest TPA is achieved by HCTR followed by EME, HCHfp and HCH, distantly followed by TET and XCB.

TABLE VI

HARDWARE COSTS OF THE HCTR, HCH, HCHFP, EME, TET AND XCB MODES WITH AN UNDERLYING FULL 10-STAGE PIPELINED 128-BIT AES CORE: THE TIME AND CLOCK-CYCLES ARE THE TIME REQUIRED TO ENCRYPT ONE SECTOR= 32 BLOCKS

Mode	**Slices**	**B-RAM**	**Frequency** (MHz)	**Clock Cycles Cycles**	**Time** (μS)	**Latency** (μS)	**Throughput** GBits/Sec	TPA
HCH	13622	80	65.939	107	1.623	26.06	2.524	25.82
HCHfp	12970	85	66.5	96	1.443	0.992	2.84	29.04
HCTR	12068	85	79.65	89	1.117	0.703	3.67	39.0
XCB	13418	85	54.021	116	2.147	1.110	1.91	19.168
EME	10120	87	67.835	107	1.577	1.120	2.60	29.82
TET	12072	87	60.505	111	1.834	1.305	2.23	23.494

In Table VII we show the six TES modes of operation when using a sequential implementation of the AES core. In a sequential architecture, EME is the slowest mode in terms of latency due to the two costly block cipher passes that requires eleven clock cycles per block. Hence, a significant increment in the total number of clock cycles is observed for the EME mode. This situation does not occur in the other four modes since they only need one encryption pass. The hash function computation is not affected in this scenario due to the fact that we use a multiplier which is essentially a combinatorial circuit able to produce a result in one clock cycle. HCHfp, HCH, TET and XCB perform behind in terms of speed, area and TPA, in that order.

While performing a sector decryption, all five TES, HCTR, HCH, HCHfp, EME and TET only require AES calls in forward mode, hence, we just need an encryption-only AES core for performing a sector decryption in those modes. It was this observation what motivated us to investigate the performance of those four modes when using an encryption-only AES underlying block cipher. The results of this experiment are summarized in Table VIII. The fastest throughput and TPA is achieved by EME (the only mode that does not use a Karatsuba-Ofman multiplier). In fact, in this case EME essentially achieves the same maximum clock frequency than that of the encryption-only AES core (see Table V). After EME, HCTR, HCHfp and HCH emerge as a distant second place. TET is even slower and more inefficient in

TABLE VII

HARDWARE COSTS OF THE HCTR, HCH, HCHFP, EME, TET AND XCB MODES WITH AN UNDERLYING SEQUENTIAL 128-BIT AES CORE: THE TIMES REPORTED ARE FOR 32 BLOCKS

Mode	**Slices**	**B-RAM**	**Frequency** (MHz)	**Clock Cycles**	**Time** (μS)	**Throughput** (Gbits/sec)	**TPA**
HCH	8688	18	64.026	416	6.497	0.631	14.00
HCHfp	7903	18	64.587	405	6.270	0.653	15.62
HCTR	7006	18	77.737	388	4.991	0.82	21.53
XCB	8351	18	52.108	455	8.731	0.469	10.749
EME	5053	20	65.922	716	10.861	0.377	12.09
TET	7030	20	58.592	421	7.185	0.570	14.512

TABLE VIII

HARDWARE COSTS OF THE HCTR, HCH, HCHFP, EME AND TET MODES WITH AN UNDERLYING ENCRYPTION-ONLY 10-STAGE PIPELINED 128-BIT AES CORE: THE TIMES REPORTED ARE FOR 32 BLOCKS

Mode	**Slices**	**B-RAM**	**Frequency** (MHz)	**Clock Cycles Cycles**	**Time** (μS)	**Latency** (μS)	**Throughput** GBits/Sec	TPA
HCHfp	7513	85	95.801	98	1.022	0.678	4.00	53.15
HCTR	6996	85	98.580	89	0.902	0.557	4.54	61.96
EME	4200	87	149.09	107	0.717	0.496	5.71	90.86
TET	7165	87	93.035	111	1.193	0.849	3.43	45.802

terms of the TPA coefficient.

C. Speculative Comparison against Software Solutions

According to Table I the implementation of the EME mode requires $2(m+1)$ block cipher calls plus some extra operations that in the rest of this discussion will be neglected. Notice that in our application, we have assumed that the plaintext length is fixed to 32 128-bit blocks. Therefore, the computational cost of an EME software implementation is lower bounded by 66 times the timing cost of one block cipher call. Using those estimations we compare in Table IX the speedup that our EME reconfigurable hardware implementation achieves compared with the best software implementations reported in the open literature.

Using the bit-slice technique, the (arguably) fastest software AES encryption implementation published till date was recently reported in [22]. That design requires about 147.2 clock cycles for encrypting one

TABLE IX

A PERFORMANCE COMPARISON OF THE EME MODE OF OPERATION USING SEVERAL AES ENCRYPTION CORES IMPLEMENTED IN SOFTWARE VS. OUR EME RECONFIGURABLE HARDWARE DESIGN.

Design	Processor	**Cycles/Sector**	**EME Latency**	**fold Speedup**
EME using AES in [22]	Intel Core 2 Duo	9715.2	4.55μS	6.34
EME using AES in [21]	AMD Athlon 64	11120	4.675μS	6.52
EME using AES in [17]	AMD Athlon 64	13134	5.47μS	7.63
EME using AES in [1]	AMD Athlon 64	14150.4	5.896μS	8.22
EME here	FPGA Virtex IV	107	0.717μS	1

128-bit block when implemented in a Intel Core 2 Duo platform running at 2.135 GHz. Other competitive software AES designs are also included in Table VIII [21], [17], [1]. As it is shown in Table IX the reconfigurable hardware EME design presented here outperforms the best software solutions by a factor of at least 6.34 fold speedup.

IX. DISCUSSIONS

As we stated in Section IV the design objective would be to match the data rates of modern day disk controllers which are of the order of 3Gbits/sec. Table VII shows that using a sequential design it is not possible 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 AES will probably be the best choice for designing disk encryption schemes.

From Table VI we see that while using a encryption/decryption pipeline AES core the most efficient mode in terms of speed is HCTR followed by HCHfp, EME, HCH, TET and XCB. The full functionality of HCH is not needed for disk encryption schemes as for this application messages would be of fixed length. Thus we can conclude that HCTR and HCHfp are the best modes to use for this application. But, the security guarantee that HCTR provides is quite weak as it has a cubic security bound. If we assume that a hard disk of 256 Giga Bytes is encrypted using HCTR and an adversary has access to this ciphertexts then (s)he has access to 2^{34} 128 bit blocks of ciphertext. Then probability with which the adversary can distinguish HCTR from a random permutation is about $\frac{1}{2^{26}} + \delta (= \frac{(2^{34})^3}{2^{128}} + \delta)$. In the same situation the distinguishing probabilities for HCH, HCHfp and EME would be around $\frac{1}{2^{60}} + \delta$. This means that using HCTR is not secure to even encrypt a full hard-disk with the same key. The throughput of

HCHfp is slightly better than EME, but as stated in [11] EME has pending patent claims. Nevertheless, according to [6] HCHfp has no such patent claims.

From table VIII we see that the encryption operation of all the modes considered here except XCB can be significantly improved if a encryption only AES core is implemented. So in certain scenarios it may be possible to have two different circuits for encryption and decryption where the encryption operation would be considerably faster. For disk encryption scenario, it is probable that a sector would be written once and would be read many times. So it is better to have a faster decryption circuit, as the decryption operation is probable to be performed more frequently. As the TES are all length preserving encryption schemes which are permutations, so one can easily replace the encryption operation with the decryption operation and vice-versa without any effect on the security guarantees provided by the modes. This subtle change can improve the total throughput of a disk-encryption considerably. If an encryption only AES core is used then EME gives the best throughput and other modes are far behind it.

X. Conclusion

Hard disk encryption for desktop and laptop computers is an application which is gaining much importance in the current days. It has been argued that TES proposals would be the candidates of choice for such applications. Proper security model for this application is already available, and there are many constructions which are provably secure under that security model. As the nature of the application dictates that the encryption/decryption algorithm should reside on the disk controller and the algorithm does not require to have knowledge of the high level partitions of the disk, a hardware implementation of the algorithms would be most preferred. From our study we have shown that a hardware implementation would be cost efficient and would be faster than software solutions (this is confirmed by the data provided in Table IX).

Though a rigorous security analysis for the different proposed TES has been done and all schemes have been claimed to be "efficient" based on grounds of speculative algorithm analysis, no study regarding their performances with compulsory data are yet available for various scenarios. In this paper we presented optimized hardware implementation of five TES. Our choice of the schemes covers all reported "efficient" schemes. To our knowledge this is the first work to report real performance data of any TES on hardware. We analyze the potential for parallelism for each of the chosen modes and argue regarding their hardware costs and performances. We also provide experimental data regarding hardware resources and throughput. Our study confirms for the first time that many proposed modes can be efficiently used for the in-place disk encryption application.

REFERENCES

[1] D. J. Bernstein. A state of the art message authentication code. Available at: http://cr.yp.to/mac.html.

[2] R. K. Bo Yang, Sambit Mishra. A high speed architecture for galois/counter mode of operation (gcm). Cryptology ePrint Archive, Report 2005/146, 2005. http://eprint.iacr.org/.

[3] D. Canright. A Very Compact S-Box for AES. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[4] D. Chakraborty and P. Sarkar. A New Mode of Encryption Providing a Tweakable Strong Pseudo-random Permutation. In Robshaw [29], pages 293–309.

[5] D. Chakraborty and P. Sarkar. HCH: A New Tweakable Enciphering Scheme Using the Hash-Encrypt-Hash Approach. In R. Barua and T. Lange, editors, *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006.

[6] D. Chakraborty and P. Sarkar. HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. Cryptology ePrint Archive, Report 2007/028, 2007. http://eprint.iacr.org/.

[7] J. Daemen and V. Rijmen. *The Design of Rijndael: AES The Advanced Encryption Standard*. Springer-Verlag, First edition, 2002.

[8] Y. Fu, L. Hao, and X. Zhang. Design of an Extremely High Performance Counter Mode AES Reconfigurable Processor. In *Proceedings of the Second International Conference on Embedded Software and Systems (ICESS'05)*, pages 262–268. IEEE Computer Society, 2005.

[9] T. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2005.

[10] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In *INDOCRYPT*, pages 315–327, 2004.

[11] S. Halevi. TET: A wide-block tweakable mode based on Naor-Reingold. Cryptology ePrint Archive, Report 2007/014, 2007. http://eprint.iacr.org/.

[12] S. Halevi and P. Rogaway. A tweakable enciphering mode. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.

[13] S. Halevi and P. Rogaway. A parallelizable enciphering mode. In T. Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

[14] S. F. Hsiao and M. C. Chen. Efficient Substructure Sharing Methods for Optimising the Inner-Product Operations in Rijndael Advanced Encryption Standard. *IEE Proceedings on Computer and Digital Technology*, 152(5):653–665, September 2005.

[15] IEEE Security in Storage Working Group (SISWG). PRP modes comparison IEEE p1619.2. IEEE Computer Society, March 2007. Available at:http://siswg.org/.

[16] K. Jarvinen, M. Tommiska, and J. Skytta. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. *Information Security, IEE Proceedings*, 152(1):3–12, October 2005.

[17] H. Lipmaa. Fast implementations: Complete aes (rijndael) library, October 2006. Available at: http://home.cyber.ee/helger/implementations/.

[18] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46, London, UK, 2002. Springer-Verlag.

[19] E. López-Trejo, F. R. Henríquez, and A. Díaz-Pérez. An Efficient FPGA Implementation of CCM Mode Using AES. In *International Conference on Information Security and Cryptology*, volume 3935 of *Lecture Notes in Computer Science*, pages 208–215, Seoul, Korea, December 2005. Springer-Verlag.

[20] C. Mancillas-López, D. Chakraborty, and F. Rodríguez-Henríquez. Efficient implementations of some tweakable enciphering schemes in reconfigurable hardware. In *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 414–424. Springer, 2007.

[21] M. Matsui. How far can we go on the x64 processors? In Robshaw [29], pages 341–358.

[22] M. Matsui and J. Nakajima. On the power of bitslice implementation on intel core2 processor. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 2007.

[23] D. McGrew and J. Viega. The galois/counter mode of operation (GCM), submission to nist modes of operation process, January 2004. Available at: http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec.pdf.

[24] D. A. McGrew and S. R. Fluhrer. The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. http://eprint.iacr.org/.

[25] D. A. McGrew and S. R. Fluhrer. The security of the extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2007/298, 2007. http://eprint.iacr.org/.

[26] D. A. McGrew and J. Viega. Arbitrary block length mode, 2004. http://grouper.ieee.org/groups/1619/email/pdf00005.pdf.

[27] D. A. McGrew and J. Viega. The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT*, pages 343–355, 2004.

[28] Phillip Rogaway, Mihir Bellare and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM Transactions on Information and System Security (TISSEC)*, volume 6, pages 365–403, 2003.

[29] M. J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.

[30] F. Rodríguez-Henríquez and Ç. K. Koç. On fully parallel karatsuba multipliers for GF(2^m). In *International Conference on Computer Science and Technology CST 2003, May 19-21 2003, Cancn, Mxico*, Lecture Notes in Computer Science, pages 405–410. Acta Press, May 2003.

[31] P. Sarkar. Improving upon the TET mode of operation. Cryptology ePrint Archive, Report 2007/317, 2007. http://eprint.iacr.org/.

[32] Seagate Technology. Internal 3.5-inch (sata) data sheet. Available at:http://www.seagate.com/docs/pdf/datasheet/disc/ds_internal_sata.pdf.

[33] V. Shoup. On fast and provably secure message authentication based on universal hashing. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.

[34] P. Wang, D. Feng, and W. Wu. HCTR: A variable-input-length enciphering mode. In D. Feng, D. Lin, and M. Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

APPENDIX

The AES Design

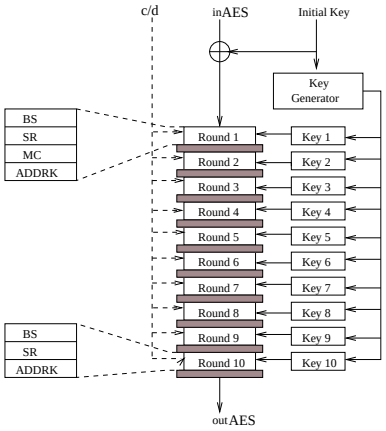


Fig. 14. AES Pipeline Architecture

We designed two AES implementations, one sequential architecture that yields a valid output after 11 clock cycles and another based on a 10-stage pipeline structure able to encrypt a block cipher per cycle after eleven cycles of latency. Both implementations utilized double port memories for computing the AES Byte Substitution (BS) transformation.

The sequential strategy uses only one round that contains the four AES steps along with a multiplexer block that eliminates the MixColumn step in the tenth round as is shown in Fig. 14. The control circuit consists of one 4-bit ascending/descending order counter for encryption/decryption, respectively. The counter output points to the correct address where the keys are stored and it controls the multiplexor **M1** that feedbacks the round or it allows that a new input data comes in, and the multiplexor **M2** that controls the omission of the MC transformation in the tenth AES round. For decryption we followed the procedure described in [7]. It is worth mentioning that the key schedule process is accomplished after 10 clock cycles. Each round key so generated is stored in a 128x32 RAM memory.

In the AES pipeline implementation the 10 AES rounds are unrolled, while the key generation is computed sequentially and each one of the round keys is stored in a register directly connected to the corresponding round as is shown in Fig. 14. Because of synchronization purposes, each AES round is isolated from the preceding one by a latch circuit. This implementation does not use a control unit, since this is added with the help of outside circuitry whenever the AES core will be used in simple or counter mode. Starting from cycle eleven, valid outputs will be produced every clock cycle.

As it was already explained, the encryption of multiple blocks in both, HCTR and HCH is accomplished using AES in counter mode. Additionally, HCTR and HCH modes also require the encryption of several

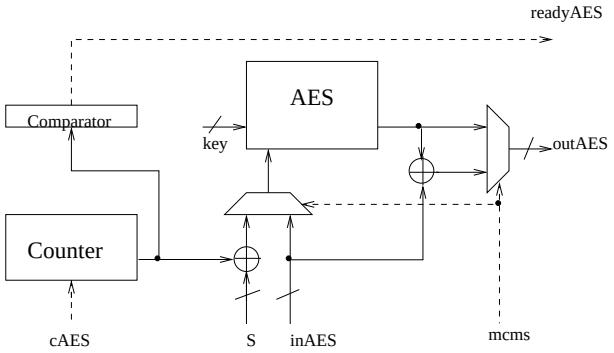


Fig. 15. AES Architecture in sequential and Counter Modes

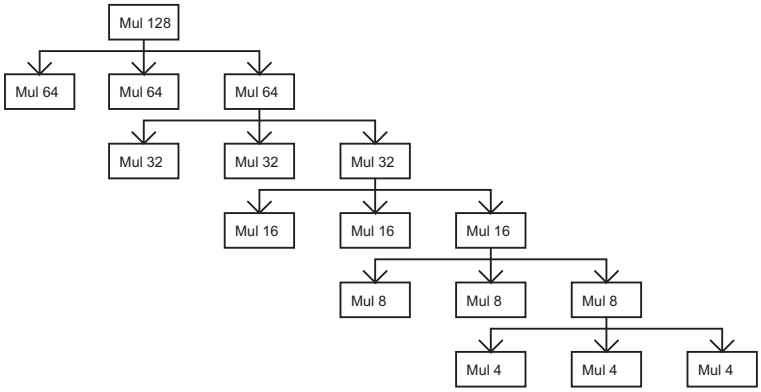


Fig. 16. Structure of the Karatsuba-Ofman Multiplier

1-block long plaintexts, which can be performed by simply using the AES core in single mode. For most applications, the implementation of two separated AES cores is prohibited, therefore we propose in this work to use a single AES core that can be programmed for implementing both functionalities, namely, the counter mode or the single mode computation as shown in Fig. 15. The **readyAES** signal in Fig. 15 indicates that a valid output has just been produced.

The Multiplier Design

Our strategy for multiplication is based on the Karatsuba-Ofman multiplier as it was presented in [30]. The Karatsuba-Ofman multiplier enjoys a superb sub-quadratic complexity of $O(n^{log_2 3})$ bit operations for multiplying two n-bit polynomials as we will briefly explain next.

Let the field $GF(2^{128})$ be constructed using the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$ of degree $n = 2^7$. Let A, B be two elements in $GF(2^{128})$. Both elements can be represented in the

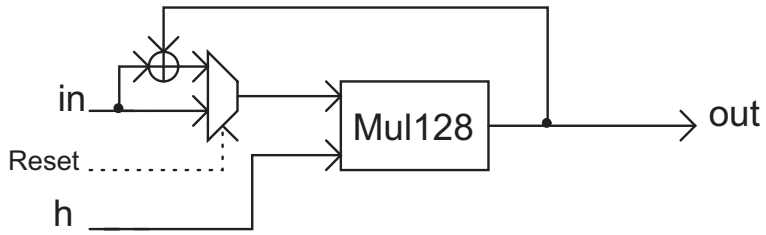


Fig. 17. Horner's rule main Architecture

polynomial basis as,

$$\begin{aligned} A &= \sum_{i=0}^{127} a_i x^i = \sum_{i=64}^{127} a_i x^i + \sum_{i=0}^{63} a_i x^i \\ &= x^{64} \sum_{i=0}^{63} a_{i+64} x^i + \sum_{i=0}^{63} a_i x^i = x^{64} A^H + A^L \end{aligned}$$

and

$$\begin{aligned} B &= \sum_{i=0}^{127} b_i x^i = \sum_{i=64}^{127} b_i x^i + \sum_{i=0}^{63} b_i x^i \\ &= x^{64} \sum_{i=0}^{63} b_{i+64} x^i + \sum_{i=0}^{63} b_i x^i = x^{64} B^H + B^L \end{aligned}$$

Then, using last two equations, the polynomial product is given as

$$C = x^{128} A^H B^H + (A^H B^L + A^L B^H) x^{64} + A^L B^L. \tag{5}$$

Karatsuba-Ofman algorithm is based on the idea that the product of last equation can be equivalently written as

$$\begin{aligned} C &= x^{128} A^H B^H + A^L B^L + \\ &\quad (A^H B^H + A^L B^L + (A^H + A^L)(B^L + B^H)) x^{64} \\ &= x^{128} C^H + C^L. \end{aligned} \tag{6}$$

It is easy to see that equation (6) can be used to compute the product at a cost of four polynomial additions and three polynomial multiplications. In contrast, when using equation (5), one needs to compute four polynomial multiplications and three polynomial additions. Karatsuba-Ofman algorithm can be applied recursively to the three polynomial multiplications in (6). Hence, we can postpone the computations of the polynomial products $A^H B^H, A^L B^L$ and $(A^H + A^L)(B^L + B^H)$, and instead we can split again each one of these three factors into three polynomial products. By applying this strategy recursively, in each iteration each degree polynomial multiplication is transformed into three polynomial multiplications with their degrees reduced to about half of its previous value.

Figure 16 shows the typical tree structure of a $GF(2^{128})$ Karatsuba-Ofman multiplier. The polynomial multiplier shown in Fig. 16 returns a 256-bit polynomial which we need to reduce back to 128 bit using the generating polynomial, $P(x) = x^{128} + x^7 + x^2 + x + 1$.

The field multiplier is the main building block for implementing the Horner's rule Algorithm described in Fig 8. The corresponding hardware architecture is shown in Fig. 17. The implementation of both hash functions are based on this module.