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 pseudorandom 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 toward constructing secure and efficient tweakable enciphering schemes. But actual experimental performance data of these newly proposed schemes are yet to be reported. In this paper, we present optimized FPGA implementations of six tweakable enciphering schemes, namely, HCH, HCTR, XCB, EME, HEH, and TET, using a 128-bit AES core as the underlying block cipher. We report the 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, TET, and HEH was implemented using a Karatsuba multiplier as the main building block. We provide detailed algorithm analysis of each of the schemes trying to exploit their inherent parallelism as much as possible. 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 throughput. In contrast, according to our place-and-route results on a Xilinx Virtex 4 FPGA, our designs achieve a throughput of 3.95 Gbps for HEH when using an encryption/decryption pipelined AES core, and a throughput of 5.71 Gbps for EME when using a encryption-only pipeline AES core. 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-art disk controllers, thus showing that they might be used for achieving provably secure in-place hard disk encryption.

Index Terms—Disk encryption, tweakable enciphering schemes, block cipher modes of operation, Karatsuba multiplier, hardware accelerator, FPGA.

1 Introduction

The security of data stored in bulk storage devices like hard disks of laptop and desktop computers, flash memories, and a variety of small mobile devices is an important issue of today. Particularly, the problem of lost/stolen laptops is especially acute, as an ever increasing number of losses are reported everyday. It has been estimated that the laptop loss rates are around two percent per year [11], which signifies that an organization with 100,000 laptops may lose on average several of them per day. A stolen laptop amounts to the loss of the hardware and the data stored in it. But what is of more severe consequence is that sensitive information gets into the hands of an unwanted person who can potentially cause much greater damage than the one incurred by the mere physical loss of the hardware and the data. A possible countermeasure of this important problem is to encrypt the data being written in the hard disk. Although there exist numerous encryption schemes meant for varied scenarios, this special application brings with it specific design problems which cannot be readily solved by traditional encryption schemes.

It has been argued that the best solution to this issue would be a hardware-based scheme, where the encryption algorithm resides in the disk controller, which has access to the disk sectors but has no knowledge about the high-level logical partitions of the disk, such as files and directories, which are maintained by the operating system. Under this scenario, the disk controller encrypts the data before it writes a sector, and similarly, after reading a sector, the disk controller decrypts it before sending it to the operating system. This type of encryption has been termed in the literature as *low-level disk encryption* or *in-place disk encryption*.

A symmetric key cryptosystem with certain specific properties can serve as a solution to the low-level disk encryption problem. One particularly important property to achieve is length-preserving encryption, i.e., the length of the ciphertext should not be more than that of the plaintext. This implies that the ciphertext itself must be enough to decrypt the enclosed data, since there is no scope to store associated data like states, nonces, salts, or initialization vectors, which are common parameters in numerous symmetric key cryptosystems. Furthermore, the schemes to be selected must be secure against adaptive-chosen plaintext and adaptive-chosen ciphertext adversaries. Such schemes are generally called CCA secure schemes (secure against chosen ciphertext attacks). Achieving CCA security means that no adversary can be able to distinguish the ciphertexts from random strings, and additionally, the attacker must not be able to modify the ciphertext so that it gets decrypted to something meaningful.

During the last few years, there has been an intense research addressing this problem, and it appears that both practitioners and researchers have come to the opinion that a class of encryption algorithms called *tweakable enciphering scheme* (TES) offers the best solution to the low-level disk encryption problem [19].

• *The authors are with the Computer Science Department, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (IPN), Av. IPN #2508 Col. San Pedro Zacatenco, Del. Gustavo A Madero, 07360 Mexico City, Mexico.*
E-mail: mancilla@computacion.cs.cinvestav.mx, {debrup, francisco}@cs.cinvestav.mx

Manuscript received 13 Mar. 2009; revised 4 Nov. 2009; accepted 23 Dec. 2009; published online 2 Mar. 2010.
Recommended for acceptance by G. Constantinides.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-03-0117.
Digital Object Identifier no. 10.1109/TC.2010.64.

Let us consider an n bit block cipher E with a key space $\mathcal{K}$ defined as a function $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$. Thus, E is capable of encrypting n bit strings. A TES is a type of block cipher *mode of operation* that allows us to extend the domain of a block cipher to (ideally) arbitrary long messages, i.e., it is a specific way to use multiple block cipher calls to encrypt messages which are longer than n bits, while satisfying the length-preserving property mentioned above. In addition to this, a TES behaves as a strong pseudorandom permutation, which is a desired feature to provide adequate security for low-level disk encryption. This means that in spite of being length preserving, a TES achieves CCA security in the sense that no efficient adversary can distinguish outputs of a TES from random strings and also cannot create valid ciphertexts (i.e., if the adversary introduces a change in any single bit of a valid ciphertext, then the corresponding plaintext on decryption will look completely random). As discussed earlier, an encryption algorithm for low-level disk encryption cannot maintain associated data like initialization vectors or nonces. However, without such state information, two disk sectors that happen to have the same data would get encrypted into the same ciphertext, which is not desirable. To overcome this difficulty, in addition to the plaintext and the key, a TES takes in an additional input parameter called tweak. A tweak is a public quantity, which was introduced to increase variability of block cipher outputs in [24]. The tweak in a TES serves the same purpose of increasing variability in the ciphertext. In [16], it was proposed that the tweak does not need to be stored explicitly but the sector addresses can themselves act as tweaks.

We stress that a TES is not the only possible way to extend the domain of applicability of a block cipher to make it suitable to encrypt arbitrarily long messages. In fact, any scheme able to do so is called a mode of operation of a block cipher. There exist numerous kinds of block cipher modes of operations designed for different functionalities, such as privacy-only modes, authenticated encryption, authenticated encryption with associated data, etc. [6]. None of these modes satisfy the requirements of low-level disk encryption. For example, privacy-only modes, such as the Counter mode or the Cipher Block Chaining (CBC) mode, are not CCA secure. On the other hand, authenticated encryption modes are not length preserving. A TES is a specific type of mode which is length preserving, provides security of that of a strong pseudorandom permutation, and introduces variability in ciphertexts by the use of tweaks.

1.1 The Known Constructions of TES

A fully defined TES for arbitrary length messages using a block cipher was first presented in [16]. In [16], it was also first stated that a possible application area for such type of encryption schemes could be low-level disk encryption. In the last few years, there have been numerous proposals for TES. These proposals fall into three basic categories: Encrypt-Mask-Encrypt type, Hash-ECB-Hash type, and Hash-Counter-Hash type. CMC [16], EME [17], and EME* [14] fall under the Encrypt-Mask-Encrypt group. PEP [7], TET [15], and HEH [38] fall under the Hash-ECB-Hash type; and XCB [31], HCTR [45], HCH [8], and ABL [33] fall under the Hash-Counter-Hash type.

All the TESs that have been designed to date use a block cipher as the basic primitive, and in addition, some schemes utilize a universal hash function which is a Wegman-Carter type polynomial hash. The constructions of the Hash-Counter-Hash and Hash-Encrypt-Hash type invoke two polynomial hash functions and a layer of encryption in-between. The Encrypt-Mask-Encrypt structure consists of two layers of encryption with a light weight masking layer in-between. So, the main computational overhead of the Encrypt-Mask-Encrypt architecture is given by the block cipher calls, whereas for the other two classes of constructions, both block cipher calls and finite field multiplications amount for a significant portion of the total computational cost. From nature of the constructions as described above, one can conclude that the Encrypt-Mask-Encrypt type schemes use approximately two block cipher calls per message block, while the two other two types require one block cipher call and two $GF(2^n)$ field multiplications per message block.

1.2 Implementation Aspects of TES

Surprisingly, in spite of being a very active research area that up to today has reported about 10 different TES constructions along with an active standardization effort that has been carried on in [19], little experimental data of these schemes are known. Instead, mostly heuristic efficiency arguments have been provided so far to compare the various modes. For example, from the point of view of efficiency, it was argued in [8], [32] that the Encrypt-Mask-Encrypt type constructions would be slower than the other 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 algorithmic dependencies. It is noticed that this is not the case for a hardware design, where block cipher invocations can have different timing costs depending on the way the cipher core is implemented (using either a sequential or a pipelined architecture) and the data dependencies associated with the particular algorithm under analysis. The same arguments hold for the multipliers as one can have an efficient fully parallel multiplier which can compute the product of 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 [15]. This comparison assumes the same hardware implementation setting reported in [46], where a fully parallel $GF(2^n)$ field multiplier able to compute a product in one clock cycle was implemented at a hardware cost in area of about three times the cost associated with one AES round function, and where the AES core was implemented through the computation of 10 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 implemented in hardware is very necessary. The recent standardization activities for such modes by the IEEE working group on

storage security [19] also demand performance data for the many proposed schemes.

As mentioned above, it is important to study the best possible strategies for exploiting the inherent parallelism of the TES in hardware implementations. In this regard, reconfigurable hardware devices offer shorter design cycles due to their ability of providing fast and accurate functionality testing. These characteristics have made FPGA platforms a popular choice for the prototyping development of cryptographic algorithms. In particular, block cipher algorithms have been profusely implemented on reconfigurable hardware platforms in the last few years [4], [9], [10], [12], [13], [21], [25], [37], [43].

Nevertheless, the relatively high size and power consumption shown by FPGA devices have been the most important drawback of that platform toward an eventual substitution of the more traditional Application-Specific Integrated Circuit (ASIC) technology. In recent years, however, FPGA manufacturers have significantly reduced the gap that still exists between FPGA and ASIC technology, paving the way for the utilization of FPGA devices not only as prototype tools but also as key components of embedded systems or even, becoming the system itself [20], [42], [44].

1.3 Our Contribution

The main contribution of this paper is thus to present a detailed algorithm analysis of six TESs with the aim of exploring achievable lower limits in area and computation time, when implemented in realistic hardware platform scenarios. The modes we chose are HCH, HCTR, XCB, EME, TET, and HEH. The modes that we left out in this study are CMC, PEP, EME*, and ABL. 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.

For all the implementations, we use AES-128 as the underlying block cipher. Whenever required, we utilize a fully parallel Karatsuba multiplier to compute the hash functions. We carefully analyze and present our design decisions, and finally, report hardware performance data of the six modes. Our implementations show that in terms of hardware resources, HCTR, HCH, HEH, TET, and XCB require more area than EME. HEH performs the best in terms of speed followed by HCTR, HCH, EME, TET, HCH, and XCB.[1]

2 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. A tweakable enciphering scheme is a function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\mathcal{K}$, $\mathcal{T}$, and $\mathcal{M}$ are nonempty sets representing the key space, tweak space, and message space, respectively. We shall often write $\mathbf{E}_K^T(.)$ instead of $\mathbf{E}(K,T,.)$. Moreover, for any tweakable enciphering scheme $\mathbf{E}_K^T(.)$, there exists its inverse $\mathbf{D} = \mathbf{E}^{-1}$, where $X = \mathbf{D}_K^T(Y)$ if and only if $\mathbf{E}_K^T(X) = Y$. Ideally, the message space $\mathcal{M} = \cup_{i \geq 1}\{0,1\}^i$, i.e., the message space consists of binary strings of all possible sizes. A TES is generally constructed using a block cipher and for specific applications like disk encryption, the message space is defined as $\mathcal{M} = \{0,1\}^{mn}$, where n is the block length of the underlying block cipher utilized by the scheme and m is the number of plaintext blocks. In this paper, we chose AES-128 as the underlying block cipher and the length of the message is the same as that of the size of a typical disk sector (which is 512 bytes), so for this paper, n and m can be replaced by 128 and 32, respectively.

By $X\|Y$, we shall mean the concatenation of two binary strings X and Y and $\mathrm{bin}_n(|X|)$ will denote the n-bit binary representation of $|X|$, which denotes the length 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 $GF(2^n)$. We also assume that the field $GF(2^n)$, with $n = 128$, is generated by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. If X and Y are n bit strings, then by $X \oplus Y$ and XY, we shall mean field addition and field multiplication, respectively. 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 the polynomial $P(x)$ defined above. By xX, we would represent the field multiplication of X by the monomial x.

3 The Schemes

In this section, we give a short description of the various algorithms whose performance is discussed in this paper, namely, HCH, HCTR, XCB, EME, TET, and HEH. A pictorial high-level description of the six modes is provided in Fig. 1, whereas their corresponding encryption algorithm description is given in Fig. 2. For a more detailed explanation regarding the constructions of the different modes, we refer the reader to the respective papers where each mode was proposed.

Most of the algorithms in their original proposals are stated in the highest possible generality, i.e., the designers of some of the proposals have tried to incorporate messages and tweaks of arbitrary lengths. However, we argue that this generality is not required, in practice, for the disk encryption problem, as the message, in this case, is always of fixed length (512 bytes), which is a multiple of the cipher block length (128 bits). The tweak, which is the sector address, is also of fixed length and it can be restricted to one block. Hence, the description of the algorithms provided in this section assume above length restrictions for both the plaintexts and the tweaks. In some proposals specifically in HCH and HEH, the authors have shown that the general algorithm can be modified for the specific application of disk encryption and have thus proposed modifications named HCHfp and HEHfp over the general algorithms. Thus, in the discussion that follows HCH and HEH will mean the HCHfp and HEHfp versions. Moreover, only the encryption algorithms of the various schemes are described in Fig. 2. In each algorithm described in that figure, we

1. Some parts of the results reported here have been published in [26]. This paper is a significantly enhanced version of [26] which contains results on new modes and with different implementation scenarios not present in [26].

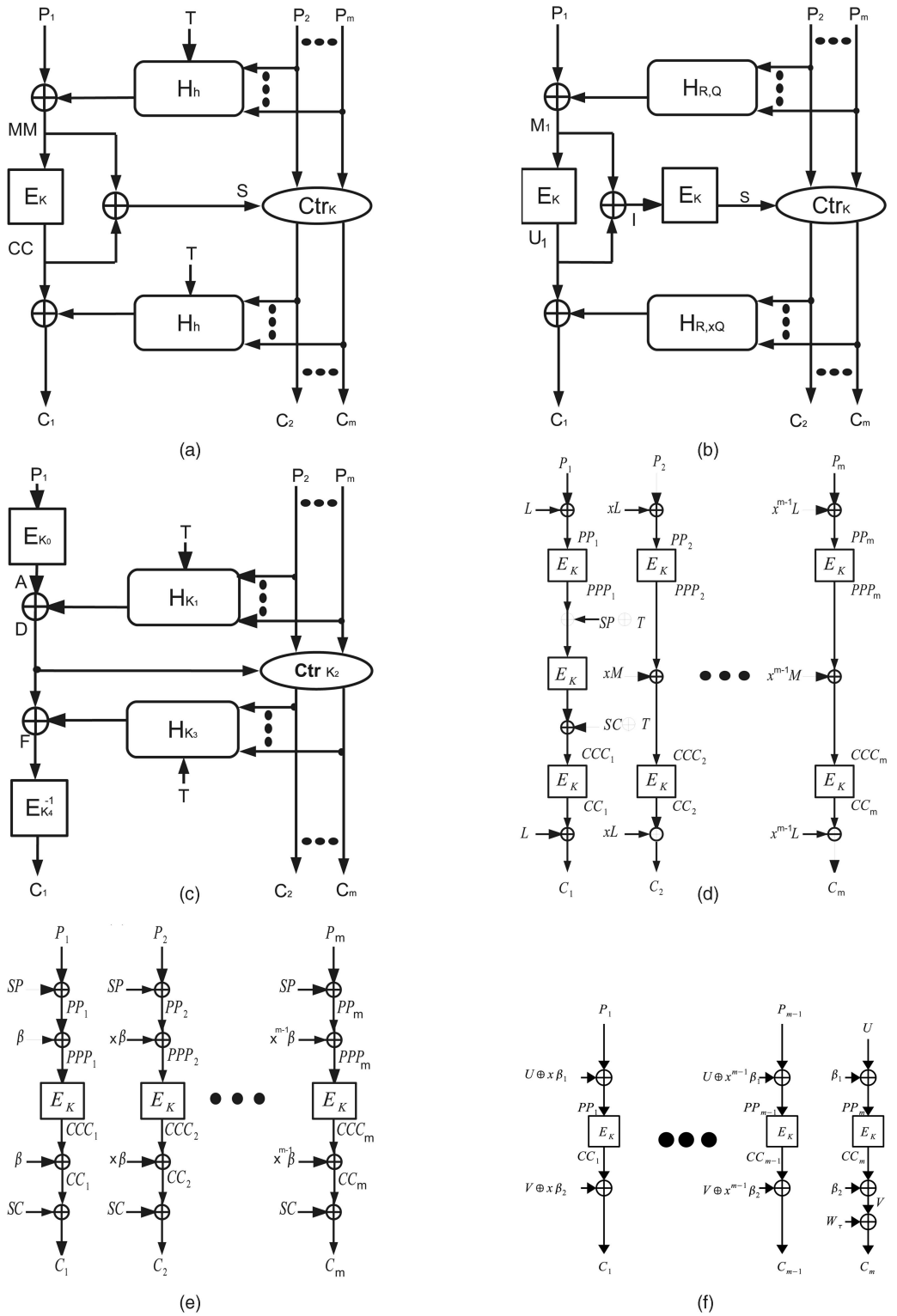


Fig. 1. High-level block diagram of the schemes: (a) Encryption using HCTR. Here, K is the key for the block cipher $E_K()$ and h is the key for the universal hash function $H_h()$. (b) Encryption using HCH. Here, $R = E_K(T)$ and $Q = E_K(R \oplus \text{bin}_n(l))$, where l is the total length of the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

use the notation: **TES_name**.Encrypt_K^T, where the superscript T stands for the tweak, whereas the subscript denotes the key or keys required by the scheme. Both the tweak T and the keys are input parameters of the algorithm. Additionally, all the schemes discussed here take as input m message blocks denoted by $P_1, P_2, \ldots, P_m$. In the description given in Fig. 2, we assume that $P_i (i = 1, \ldots, m)$ and T are n bit strings.

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

Algorithm HCH.Encrypt$^T_K(P_1,\ldots,P_m)$
1. $R \leftarrow E_K(T)$; $Q \leftarrow E_K(R \oplus \mathsf{bin}_n(mn))$;
2. $M_1 \leftarrow P_1 \oplus H_{R,Q}(P_2||\ldots||P_{m-1}||P_m)$;
3. $U_1 \leftarrow E_K(M_1)$; $I \leftarrow M_1 \oplus U_1$; $S \leftarrow E_K(I)$;
4. $(C_2,\ldots,C_{m-1},C_m) \leftarrow \mathsf{Ctr}_{K,S}(P_2,\ldots,P_{m-1},P_m)$;
5. $C_1 \leftarrow U_1 \oplus H_{R,xQ}(C_2||\ldots,C_{m-1}||C_m)$;
6. return $(C_1,\ldots,C_m)$;

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

Algorithm EME.Encrypt$^T_K(P_1,P_2,\ldots,P_m)$
1. $L \leftarrow xE_K(0^n)$
2. $(PPP_1,\ldots,PPP_m) \leftarrow$F-ECB$(P_1,P_2,\ldots,P_m;L)$
3. $MP \leftarrow (PPP_1 \oplus PPP_2 \oplus \ldots \oplus PPP_m) \oplus T$
4. $MC \leftarrow E_K(MP)$; $M \leftarrow MP \oplus MC$
5. **for** $i \leftarrow 2$ to m **do**
6. $\quad CCC_i \leftarrow PPP_i \oplus x^{i-1}M$
7. **end for**
8. $CCC_1 \leftarrow MC \oplus (CCC_2 \oplus CCC_3 \oplus \ldots \oplus CCC_m) \oplus T$
9. $(C_1,C_2,\ldots,C_m) \leftarrow$L-ECB$(CCC_1,\ldots,CCC_m;L)$
10. **return** $(C_1,C_2,\ldots,C_m)$;

Algorithm TET.Encrypt$^T_{K_1,K_2,\tau,\sigma}(P_1,P_2,\ldots,P_m)$
1. $L \leftarrow \mathsf{bin}_n(mn)$;
2. $X \leftarrow E_{K_1}(L)$;
3. $\beta \leftarrow E_{K_1}(T \oplus xX)$;
4. $SP \rightarrow 0^n$; $SC \leftarrow 0^n$;
5. **for** $i \leftarrow 1$ to m
6. $\quad SP \leftarrow (SP \oplus P_i)\tau$;
7. **end for**
8. $SP \leftarrow SP\sigma^{-1}$
9. **for** $i \leftarrow 1$ to m
10. $\quad PP_i \leftarrow P_i \oplus SP$;
11. $\quad PPP_i \leftarrow PP_i \oplus x^{i-1}\beta$;
12. $\quad CCC_i \leftarrow E_{K_2}(PPP_i)$;
13. $\quad CC_i \leftarrow CCC_i \oplus x^{i-1}\beta$
14. $\quad SC \leftarrow (SC \oplus CC_i)\tau$
15. **end for**
16. $SC \leftarrow SC\sigma^{-1}$;
17. **for** $i \leftarrow 1$ to m
18. $\quad C_i \leftarrow CC_i \oplus SC$;
19. **Return** $(C_1,\ldots,C_m)$;

Algorithm HEH.Encrypt$^T_{K,\tau}(P_1,P_2,\ldots,P_m)$
1. $\beta_1 \leftarrow E_K(T)$; $\beta_2 \leftarrow x\beta_1$;
2. $U \leftarrow P_1$;
3. **for** $i = 1$ to m,
4. $\quad U \leftarrow U\tau \oplus P_i$
5. **end for**
6. $Q \leftarrow \beta_1$;
7. **for** $i = 1$ to $m-1$
8. $\quad Q \leftarrow xQ$;
9. $\quad PP_i \leftarrow P_i \oplus U \oplus Q$;
10. $\quad CC_i \leftarrow E_K(PP_i)$;
11. **end for**
12. $PP_m \leftarrow U \oplus \beta_1$; $CC_m \leftarrow E_K(PP_m)$;
13. $V \leftarrow CC_m \oplus \beta_2$; $Q \leftarrow x\beta_2$;
14. $C_1 \leftarrow CC_1 \oplus Q \oplus V$; $W \leftarrow CC_1$;
15. **for** $i \leftarrow 2$ to $m-1$
16. $\quad Q \leftarrow xQ$;$C_i \leftarrow CC_i \oplus Q \oplus V$;
17. $\quad W \leftarrow W\tau \oplus C_i$
18. **end for**
19. $C_m \leftarrow V \oplus W\tau$
20. **Return** $(C_1,\ldots,C_m)$;

Fig. 2. The encryption algorithms.

3.1 Hash-Counter-Hash

We begin with the description of the Hash-Counter-Hash type algorithms. As explained above, in this work, we consider three of such schemes, namely, HCTR, HCH, and XCB. This class of TES constructions utilizes a variant of the Wegman-Carter polynomial hash combined with a counter mode of operation. The first scheme described in Fig. 2 is the encryption procedure using HCTR. Lines 1 and 5 of the HCTR algorithm compute the $H_h(.)$ hash function. For a plaintext string, $X = X_1\|X_2\|\cdots\|X_p$, where $X_i \in \{0,1\}^n$, $H_h(X)$ is defined as

$$H_h(X) = X_1h^{p+1} \oplus X_2h^p \oplus \cdots \oplus X_ph^2 \oplus \mathrm{bin}_n(|X|)h, \quad (1)$$

where h is an n-bit hash key. In addition to the two hash layers, HCTR performs a counter mode operation in line 4. Given an n-bit string S, a key K and m blocks of plaintext/ciphertext $A_1, A_2, \ldots, A_m$ ($A_i \in \{0,1\}^n$), the counter mode is defined as

$$\mathrm{Ctr}_{K,S}(A_1,\ldots,A_m) = (A_1 \oplus E_K(S \oplus \mathrm{bin}_n(1)),\ldots, A_m \oplus E_K(S \oplus \mathrm{bin}_n(m))). \quad (2)$$

The HCH algorithm is very similar to that of HCTR. The main difference is the way the tweak is handled and an extra encryption before the counter mode. The structural difference can be easily observed by comparing the two schemes in Fig. 1. HCH also uses a polynomial hash similar to that of HCTR. For $X = X_1\|\cdots\|X_p$, where $X_i \in \{0,1\}^n$, the hash function $H_{R,Q}(\cdot)$ is defined as

$$H_{R,Q}(X) = Q \oplus X_1R^{p-1} \oplus X_2R^{p-2} \oplus \cdots \oplus X_{p-1}R^2 \oplus X_pR. \quad (3)$$

The counter mode of HCH is same as that described in (2). In [8], the authors describe another version of HCH called HCHfp which is specifically meant for fixed-length messages.

The XCB scheme uses the hash $\boldsymbol{h}_h(X,T)$, where X is same as defined above and T is an n bit string. The hash is defined as

$$\boldsymbol{h}_h(X,T) = X_1h^{p+2} \oplus X_2h^{p+1} \oplus \cdots X_ph^3 \oplus Th^2 \oplus \left(\mathrm{bin}_{\frac{n}{2}}(|P|)\|\mathrm{bin}_{\frac{n}{2}}(|T|)\right)h. \quad (4)$$

TABLE 1
Summary of the Characteristics of the TES Described

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

We consider a fixed-length m-block message for all the schemes.

The counter mode in XCB is a bit different from the one in (2). Let us define a function $incr_i(S)$, where i is a positive integer and S an n-bit string defined as $S = S_l \| S_r$, where S_r is 32 bits long and S_l is $(n-32)$ bits long. Then, we define

$$incr_i(S) = S_l \| (S_r + i (\mathrm{mod}\ 2^{32})).$$

The counter mode of line 6 of the XCB algorithm is defined as

$$\overline{\mathrm{Ctr}}_{K,S}(A_1, \ldots, A_m) = (A_1 \oplus E_K(incr_1(S)), \ldots, A_m \oplus E_K(incr_m(S)).$$

3.2 Encrypt-Mask-Encrypt

The candidate for the Encrypt-Mask-Encrypt category considered here is called EME which stands for ECB-Mask-ECB. As the name suggests, the EME algorithm consists of two ECB layers with a layer of masking in-between. The description of the EME algorithm is provided in Fig. 2. This TES uses the variants of ECB described in the algorithm in Fig. 2 as F-ECB and L-ECB. These functions are defined as follows:

$$\text{F-ECB}(X_1, \ldots, X_m : L) = (E_K(X_1 \oplus L), E_K(X_2 \oplus xL), \ldots, E_K(X_m \oplus x^{m-1}L))$$
$$\text{L-ECB}(X_1, \ldots, X_m : L) = (E_K(X_1) \oplus L, E_K(X_2) \oplus xL, \ldots, E_K(X_m) \oplus x^{m-1}L).$$

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 2,048 bytes (2 KB) of data. This message length restriction was removed in a construction called EME* [14] 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 2 KB and their lengths are multiples of 128 bits.

3.3 Hash-Encrypt-Hash

The other two TES modes, TET and HEH, fall under the category Hash-Encrypt-Hash. As described in the corresponding algorithms of Fig. 2, these two schemes also use polynomial hash functions similar to the ones included in HCH and HCTR. TET uses two layers of blockwise invertible universal hash functions with an ECB layer of encryption between them. To compute the hash function, TET requires a hash key τ which must meet certain properties. To ensure invertibility of the hash function, $\tau \in GF(2^n)$ must be such that for an m block message, $\sigma = \sum_{i=1}^{m} \tau^i \neq 0$. 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 from a master key. The encryption algorithm also requires the value of σ^{-1}. This makes TET rather complicated for applications which require encryption of variable length messages. In the algorithm in Fig. 2, we only present a fixed-length version of the TET algorithm. Also, we assume a fixed value of τ and that σ^{-1} has been precomputed offline.

HEH is a significant improvement over TET, where the requirement of the inverse computation has been completely removed. The algorithm for HEH, as described in Fig. 2, is also meant for fixed-length messages. The specific version of HEH that we present in Fig. 2 has been named by the authors in [38] as HEHfp.

We summarize in Table 1 the characteristics of the various modes considered. Table 1 shows the number of basic operations required by each TES, and also the number of keys associated to them. The security provided by each mode is similar in nature, i.e., every mode depicted in Table 1 has a quadratic security bound.[2]

4 Design Decisions

The speed of a low-level disk encryption algorithm must meet the current data rates of commercial disk controllers. With emerging technologies like serial ATA and Native Command Queuing (NCQ), modern day disk controllers can provide data rates around three Giga-bits per second [40]. Thus, our design objective is to achieve an encryption/decryption speed which matches this data rate.

For implementing all six schemes, we chose the underlying block cipher as AES-128. As explained in the previous section, the modes HCH, HCTR, XCB, HEH, and TET use two major building blocks, namely, a polynomial universal hash and the block cipher. EME does not need a hash function in its operation. Since AES-128 was our selection for the underlying block cipher, proper design decisions for the AES structure must meet the target speed. Out of many possible

2. Note that in the original paper [45] where HCTR was proposed, the author showed a cubic bound which was improved in [5]. The original paper [31], where XCB was proposed did not have a security proof. The security proof of XCB was recently reported in [32].

designs reported in the literature [4], [12], [13], [18], [25], 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* encryption.

This decision was taken based on the fact that the structure of the AES algorithm admits to a natural 10-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 fast designs with up to 70 pipeline stages have been reported [21], but such designs would increase the latency, i.e., the total delay before a single block of ciphertext can be produced. As the message lengths in the target application are particularly small (as mentioned, 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, TET, and HEH modes is an efficient multiplier in the field $GF(2^{128})$. Out of many possible choices, we selected a fully parallel Karatsuba multiplier which can multiply two 128-bit strings in a single clock cycle at a subquadratic computational cost [36]. This multiplier occupies about 1.4 times the hardware resources required by our implementation of one single AES round.[3] Because of this, the total hardware area of EME (which does not require multipliers) is significantly less than that required by the other modes studied here. 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, HEH, and HCHfp algorithms imply that one multiplicand is always fixed, thus allowing the usage of precomputed lookup tables that can significantly speed up the multiplication operation. Techniques to speed up multiplication by lookup tables for the software platform scenario are discussed in [22], [30], [34], [41], [46]. These techniques can also be, to a certain extent, utilized in hardware implementations. However, there is a trade-off in the amount of speed that can be obtained by means of precomputation and the amount of data that needs to be stored in tables. Significantly higher speeds can be obtained if one stores larger tables. This speedup thus comes with an additional cost of area and also the potentially devastating penalty of secure storage. Moreover, if precomputation is used in a hardware design, then the key must 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 a fast but large multiplier is justified in the scenario under analysis.[4] Regarding storage of key-related materials, we make an exception to this design decision in the case of TET. TET requires computation of an inverse, which is a particularly expensive arithmetic operation. In case of TET, we store the hash key τ along with the precomputed 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.

3. For specific experimental details, see Table 4.
4. A similar design decision was taken in [30], [39].

We prototyped our design in the Xilinx Virtex 4 device xc4v1x100-12FF1148. The choice of our target device from the Virtex 4 family was guided by the need for high throughput. The studies presented here are prototypical in nature, i.e., through their reconfigurable hardware implementations, we aim to show the feasibility of the designs. We do not recommend the use of FPGA devices in a real application. The Virtex 4 family of devices is quite costly to date, and thus, it would not be feasible to put such devices in a commercial circuit. But we hope that these designs when translated to ASICs will provide more throughput, and we also anticipate that the manufactured cost will go significantly down, as it is expected that there would be an increasing need for mass production of such circuits.

Although the main target device for the reported design is a Xilinx Virtex 4 FPGA, we also performed experiments of the same designs on other FPGA devices. Specifically, we observed that our designs show a reasonable performance when implemented on cheaper FPGAs such as the Xilinx Virtex 2 Pro family. As expected, the implementation of our designs on the Xilinx Virtex 5 family gives even better throughput than the ones obtained in Virtex 4. For comparison purposes, we also report the performance obtained by the sequential versions of our designs when implemented on the economical Xilinx Spartan 3e family.

5 The Design Overviews

In this section, we give a careful algorithm analysis of the modes' data dependencies and explain how the parallelism present in the algorithms can be exploited. From our design decisions of using a 10-stage pipeline design for the AES, and a single clock cycle Karatsuba multiplier along with the Horner's rule for computing the polynomial hash functions, we came up with the following observations:

1. Computation of a hash involving m blocks can be accomplished in exactly m clock cycles.
2. n AES calls which can be computed in parallel (for example, as in counter mode or in ECB mode) can be completed in $n + 10$ cycles, as it will take 10 cycles for the AES pipeline to fill out, and thereafter, at each clock cycle, a cipher block will be produced.
3. AES calls which cannot be parallelized will be completed after 11 clock cycles.

Using these basic observations and the algorithm descriptions in Fig. 2, we show in Fig. 3 the number of clock cycles required by the six TES modes considered in this work. For example, the clock cycle count for HCH is done as follows.

Referring to the Algorithm **HCH**.Encrypt in Fig. 2, the algorithm starts with the computation of the parameter R in Step 1. Notice that we are forced to calculate R in single mode, which implies that 11 clock cycles will be required for computing this parameter. Simultaneously, the 10 AES round keys can be calculated by executing concurrently the AES key schedule algorithm. The hash function of Step 2 can be written as (see (3))

$$\begin{aligned} M_1 &= P_1 \oplus H_{R,Q}(P_2 \| \dots \| P_{32}) \\ &= Q \oplus P_1 \oplus P_2 R^{m-1} \oplus P_3 R^{m-2} \cdots \oplus P_{31} R^2 \oplus P_{32} R \\ &= Q \oplus P_1 \oplus Z, \end{aligned}$$

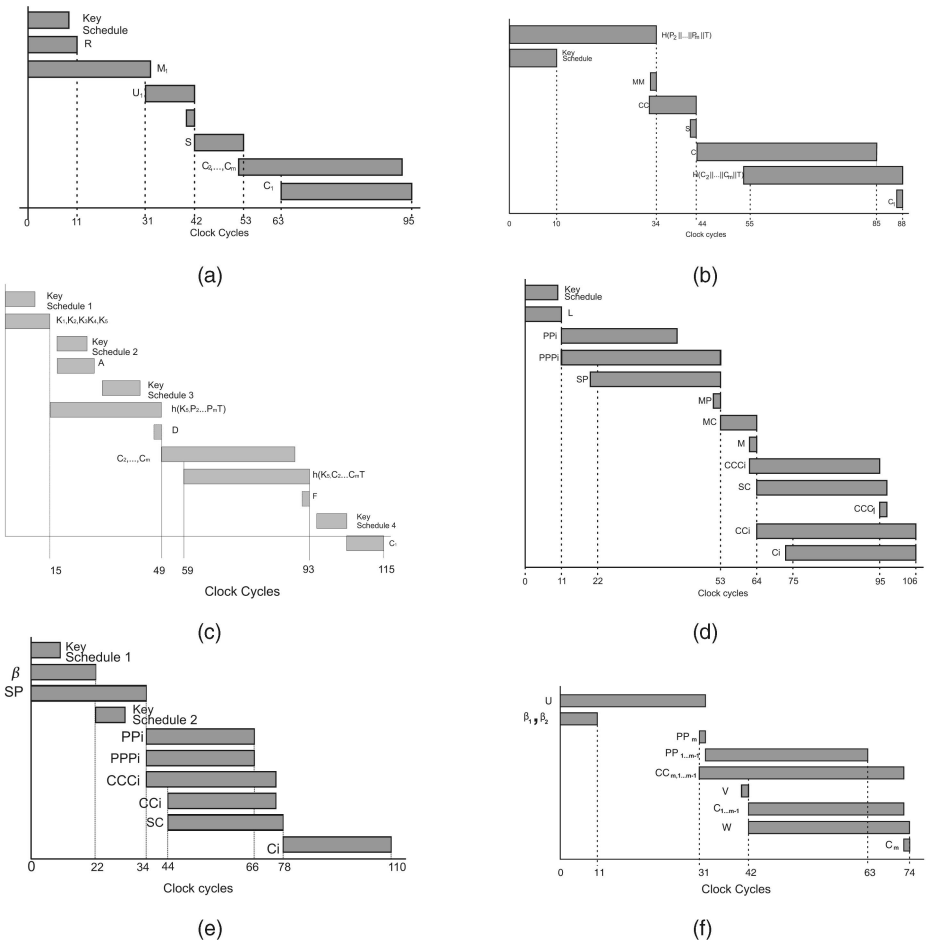


Fig. 3. Timing diagrams of the six TES modes of operation. (a) HCH. (b) HCTR. (c) XCB. (d) EME. (e) TET. (f) HEH.

where Z is the result of hashing the last 31 blocks of the message. It is noticed that Z and Q can be computed in parallel. For computing Z, 31 multiplications are required, whereas obtaining the value of Q takes 11 clock cycles. So, the computation of the hash at step 3 takes 31 clock cycles. Then, the computation of Step 4 requires two single-mode encryptions, 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 block cipher calls which can be pipelined. So, computation of step 5 requires a total of $31 + 10 = 41$ clock cycles. Hence, the first cipher block C_2 is produced 11 clock cycles after the counter mode starts. The second hash function computation of Step 7 can start as soon as C_2 is available in the clock cycle 75. This implies that the computation of this hash function can be completed at the same time that the last cipher block (C_m) of Step 5 is produced.

We list in Table 2 the total number of clock cycles required by each of the six modes to encrypt a whole disk sector of 32 blocks, along with the number of clock cycles that would be required to produce the first cipher block. 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 one clock cycle multiplier). Precomputation in the multiplier or other hash and/or multiplication strategies may bring down the cost of computing the hash function in HCH, HCTR, TET, and HEH.

6 Architecture of HCH

As a representative design example, we shall discuss the architecture of HCH in detail. The architectures for the other modes are quite similar and will not be discussed here. Fig. 4 shows a general overview of the architecture of the HCH mode of operation coprocessor. As can be seen, the AES block must be implemented both in counter and single 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 [27].

TABLE 2
The Number of Clock Cycles Required for Various Constructions

Scheme	Total clock cycles to encrypt a sector	Total Clock cycles to output first block
HCH	106	75
HCHfp	95	64
HCTR	88	55
XCB	115	59
EME	106	75
TET	110	79
HEH	74	42

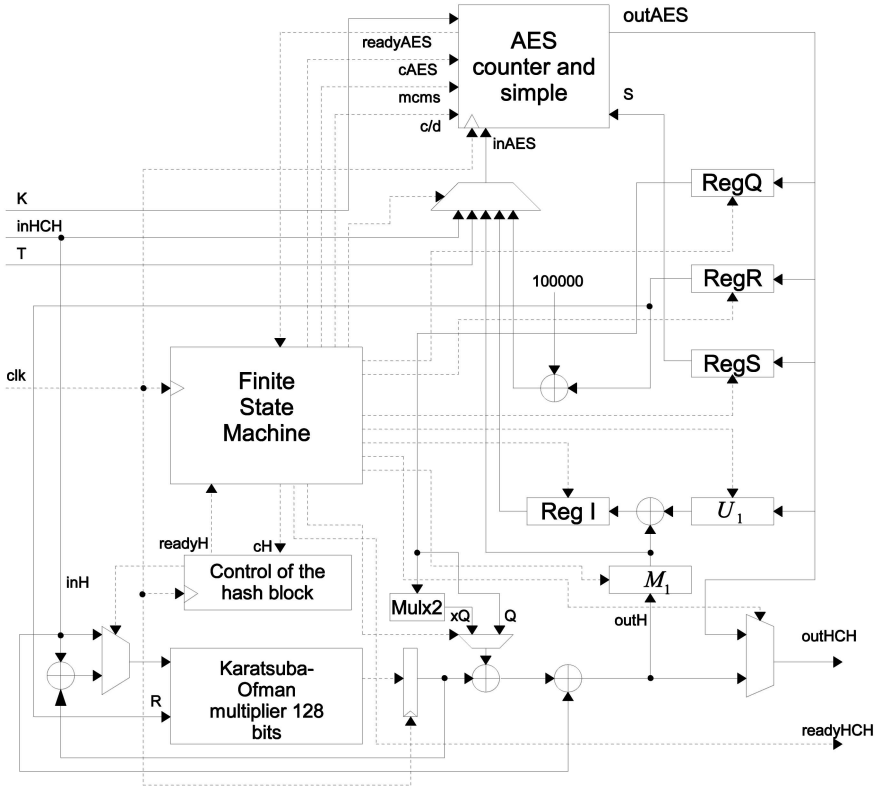


Fig. 4. HCH coprocessor architecture.

The HCH control unit is shown in Fig. 4. It controls the AES block by means of four 1-bit signals, namely: **cAES** that initializes a new counter mode cycle, the **c/d** signal that selects between encryption or decryption mode, the **mcms** signal that indicates whether one single block must be processed or rather, multiple blocks by means of the counter mode. Finally, **readyAES**, which 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, and **S** that receives the initialization parameter required by the counter mode (see (2)). The hash function block is controlled by means of two signals: **cH** that resets the accumulator register and the block counting that have been already processed, and **readyH** that indicates when the hash function computation has been accomplished. The data input/output of this block is carried out by the **inHCH** and **outHCH** busses, respectively. The parameters **R** and **Q** (see Step 1 of Algorithm **HCH**.Encrypt in Fig. 2) are calculated by the AES block and send through the busses to the hash function.

The HCH control unit implements an eight-state finite-state machine that executes the HCH sequence of operations. At each state, a 14-bit control word is continuously updated so that the required operations by the HCH algorithm are performed in the correct sequence. 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. Since the hash function input **inHCH** can come from either the system input or from the output of the AES counter mode, a multiplexer block is needed in order to select the correct input source for the hash block. Furthermore, the decryption algorithm of HCH (see [27, Fig. 2]) requires the computation of xQ, i.e., the field multiplication of the parameter Q by the monomial x. We compute **xQ** in the finite field GF(2^{128}) by means of an *xtimes* operation, which was implemented as described in [35].

We do not present the details of the other designs as they follow the same strategy, but in Table 3, we show the extra hardware resources required by each architecture of the 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, a fact that will become more explicit from the experimental results presented in the next section.

7 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 of data; *latency*, i.e., the time needed to produce the first output block; the size of the

TABLE 3
Hardware Resources Utilized by the TES

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

TABLE 4
Performance of an Encryption/Decryption AES Round and a 128-Bit Fully-Parallel Karatsuba Multiplier

Design	Slices	B-RAM	Critical Path(ηs)
Full AES round	1215	8	11.00
Encryption-only AES round	298	8	6.71
multiplier	3223	-	9.85

TABLE 5
Performance of the AES and Hash Implementations

Method	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Throughput (Gbps)
Full-core AES-Sequential	1301	18	81.97	10	1.05
Full-core AES-Pipeline	6368	85	83.88	1	10.74
Encryption-Only AES-Pipeline	2468	85	149.00	1	19.07
Hash function	4014	-	101.45	1	12.99

TABLE 6
Hardware Costs of the Modes with an Underlying Full 10-Stage Pipelined 128-Bit AES Core When Processing One Sector of 32 AES Blocks: Virtex 4 Implementation

Mode	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Time (μS)	Latency (μS)	Throughput GBits/Sec
HCTR	12068	85	79.65	89	1.117	0.703	**3.665**
HCH	13622	85	65.94	107	1.623	0.801	2.524
HCHfp	12970	85	66.50	96	1.443	0.990	2.837
XCB	13418	85	54.02	116	2.147	1.114	1.907
EME	10120	87	67.84	107	1.577	1.123	2.597
TET	12072	87	60.51	111	1.834	1.301	2.232
HEH	11545	85	72.44	75	1.035	0.591	**3.956**

circuit in slices; the number of B-RAMs used and the *throughput*. The rest of this section is organized as follows: In Section 7.1, we report the area and timing performance obtained in the implementation of the main building blocks required for the TES modes. Then, Section 7.2 reports the performance achieved by the TES modes analyzed in this paper. The comparison was carried out using a fully pipelined AES core, a sequential AES core, and a pipelined encryption-only AES core. Finally, in Section 7.3, we give a speculative comparison of our EME design against some of the best software implementations reported for the AES cipher.

7.1 Main Building Blocks

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 4 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 Karatsuba multiplier.[5]

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

Considering the utilization of B-RAMs and slices, the size of one full AES round in our design is about 40 percent smaller than that of the Karatsuba multiplier. However, the critical path delay associated to an encryption/decryption AES round is longer than that of the multiplier block by about 10 percent.

The two building blocks shown in Table 4 were used for implementing a full AES core (i.e., a key generation/encryption/decryption core) in sequential and pipelined architectures; an encryption-only pipeline AES core and a hash function for the HCH, HCTR, TET, XCB, and HEH modes. Table 5 summarizes the performance obtained in the implementation of those blocks. Note that the sequential AES core gives significantly poor throughput, while the hash function has better throughput than the full AES pipeline but smaller throughput than the encryption-only AES core.

7.2 Performance Comparison of the Six TES Modes

In Table 6, 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 6 is one more than those estimated in Section 5. This is because in our actual implementation, one clock cycle is wasted on the initial reset operation. The critical path of the designs shown in Table 6 is mainly determined by the AES core, which, as shown in Table 5, has a longer path than the hash function utilized in all five HCTR, HCH, HEH, TET, and XCB modes.

From Table 6, it is evident that EME is the most economical mode in terms of area resources, mainly due to the fact that

TABLE 7
Hardware Costs of the Modes with an Underlying Sequential 128-Bit AES Core When Processing One Sector of 32 AES Blocks: Virtex 4 Implementation

Mode	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Time (μS)	Throughput (Gbits/sec)
HCTR	6000	18	76.46	398	5.205	**0.786**
HCH	6805	18	63.36	416	6.565	0.624
HCHfp	6481	18	63.84	405	6.339	0.645
XCB	6706	18	51.86	455	8.773	0.466
EME	2958	20	80.00	716	8.950	0.457
TET	6518	20	58.08	421	7.248	0.565
HEH	6128	18	69.36	384	5.536	**0.739**

this mode does not utilize 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 six modes analyzed, HEH is both the fastest and the mode that shows the lowest latency to produce the first block. XCB is the second most expensive mode in terms of hardware resource utilization, and the slowest. HCH requires more hardware resources than HCTR. TET is slower than HCH even assuming that its parameter σ^{-1} has been previously precomputed.

In terms of speed, the fastest mode is HEH since it only utilizes one AES block cipher call in sequential mode that can be computed concurrently with the hash function computation, whereas HCTR requires one AES call in the single mode which cannot be parallelized and HCH requires a total of four such calls (although only three have consequences in terms of clock cycles since the fourth one is masked with the computation of the hash function). Under this scenario, ordered from the fastest to the slowest, we have HEH, HCTR, HCHfp, EME, HCH, TET, and XCB.

In Table 7, 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 require 11 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 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 fully parallel multiplier block able to produce a result in one clock cycle. Using a sequential AES core, the best throughput is obtained by HCTR and HEH in that order.

While performing a sector encryption, all modes considered here, except for XCB, require AES calls in encryption mode only. Hence, we just need an encryption-only AES core for performing a sector encryption in those modes. It was this observation that motivated us to investigate the performance of all modes except XCB using an encryption-only AES underlying block cipher. The results of this experiment are summarized in Table 8. The fastest throughput in this scenario is achieved by EME (the only mode that does not use a Karatsuba multiplier). In fact, in this case, EME essentially achieves the same maximum clock frequency as that of the encryption-only AES core (see Table 5). EME is closely followed by HEH, and TET turns out to be the slowest.

Although our designs were optimized for their implementation on the Xilinx Virtex 4 device xc4vlx100-12ff1148, in order to provide the reader with a better comparison spectrum, we performed additional simulations on some other FPGA families such as the state-of-the-art Xilinx Virtex 5 and cheaper FPGA families like Virtex 2 pro and Spartan 3E. The obtained place-and-route simulation results are summarized in Table 9.[6]

Let us recall that Tables 6, 7, and 8 present the Virtex 4 implementation results achieved by the TES modes when using a full pipelined, sequential, and encryption-only AES core designs, respectively. The first three portions in Table 9 show the performances obtained by those three designs when implemented on the Virtex 5 device xc5vlx110-3ff1153. Due to the technology advantage of the Virtex 5 architecture, in this experiment, we got much better performances in terms of both area and throughput. One thing to notice is that there are both similarities and discrepancies between the results obtained in these two family of devices. For example, in both experiments, HCTR and HEH are the fastest TES and XCB the slowest for the full pipelined architecture. Moreover, we also tested the full pipelined architecture on the Virtex 2 pro device xcvp40-6ff1152 (the results of this experiment are shown in the fourth portion of Table 9), obtaining roughly the same trends among the modes already observed in the Virtex 4 and Virtex 5 implementations. On the other hand, in the case of the encryption-only AES core design, HCTR outperformed EME in the Virtex 5 implementation, whereas EME was the clear winner in the Virtex 4 device simulation (see Table 8). Finally, we implemented the two fastest TESs under the sequential core scenario in the Spartan 3E device xc3s1600e-5fg484.

7.3 Speculative Comparison Against Software Solutions

According to Table 1, 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 thirty-two 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 these estimations, we show in Table 10 the speedup that our EME reconfigurable hardware implementation achieves compared with the best software implementations reported in the open literature.

The fastest software AES encryption implementation published up to date was recently presented in [22]. That design requires about 129.28 clock cycles for encrypting one 128-bit block when implemented in an Intel Core i7 processor running at 2.67 GHz, with measurements using 576-byte plaintexts. This implies that the clock cycle count for encrypting one whole sector using EME is lower bounded by $66 \times 129.28 = 8{,}532.48$. The corresponding throughput is computed as $2.67 \times 4{,}096/8{,}532.48 = 1.282$ Gbps. Other competitive software AES designs are also included in Table 10, such as [1], [2], [23], [28], [29]. As shown in Table 10, the reconfigurable hardware EME design presented here outperforms the best software solutions by a factor of at least 4.45 speedup.

6. We note that the architecture of Xilinx Virtex 5 is substantially different than the ones of previous generations. Specifically, each Virtex-5 slice contains four 6-input 2-output LUTs and four flip-flops, whereas slices in previous families have two 4-input 1-output LUTs and two flip-flops.

TABLE 8
Hardware Costs of the HCH, HCTR, EME, TET, and HEH Modes with an Underlying Encryption-Only 10-Stage Pipelined 128-Bit AES Core When Processing One Sector of 32 AES Blocks: Virtex 4 Implementation

Mode	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Time (μS)	Latency (μS)	Throughput GBits/Sec
HCTR	6996	85	98.58	89	0.902	0.557	4.540
HCHfp	7513	85	95.80	98	1.022	0.678	4.000
EME	4200	87	149.09	107	0.717	0.496	**5.710**
TET	7165	87	93.04	111	1.193	0.849	3.430
HEH	7494	85	93.70	75	0.800	0.458	**5.117**

TABLE 9
Performance of the Modes in Other FPGA Families of Devices

	Mode	Slices	B-RAM	Frequency (MHz)	Clock Cycles	Time (μS)	Latency (μS)	Throughput GBits/Sec
Full 10-Stage pipelined AES-128 core in Virtex 5	HCTR	5865	85	107.93	89	0.824	0.519	**4.965**
	HCH	5667	85	91.24	107	1.172	0.832	3.452
	HCHfp	5640	85	92.38	96	1.039	0.692	3.941
	XCB	5854	85	92.69	116	1.251	0.647	3.272
	EME	4518	87	90.19	107	1.186	0.831	3.450
	TET	5554	87	89.91	111	1.234	0.878	3.317
	HEH	5139	85	89.65	75	0.836	0.479	**4.896**
Sequential AES in Virtex 5	HCTR	2735	18	103.75	398	3.836		**1.06**
	HCH	2986	18	87.59	416	4.749		0.862
	HCHfp	2976	18	88.68	405	4.566		0.896
	XCB	3057	18	88.97	455	5.114		0.800
	EME	1274	20	107.00	716	6.691		0.612
	TET	2968	20	86.50	421	4.867		0.841
	HEH	2940	18	86.07	384	4.461		**0.918**
Encryption only AES-128 in Virtex 5	HCTR	3192	85	147.44	89	0.603	0.379	**6.785**
	HCHfp	3324	85	118.30	96	0.811	0.642	5.047
	EME	2005	87	173.80	107	0.615	0.431	**6.653**
	TET	2979	87	125.30	111	8.885	0.630	4.623
	HEH	3695	85	120.54	75	0.622	0.356	6.583
Full 10-Stage pipelined AES-128 core in Virtex 2 pro	HCTR	10946	85	73.06	89	1.218	0.752	**3.360**
	HCH	11718	85	50.30	107	2.127	1.490	1.920
	HCHfp	11227	85	63.52	96	1.511	1.180	2.710
	XCB	11685	85	50.11	116	2.315	1.177	1.769
	EME	9423	87	62.57	107	1.710	1.182	2.390
	TET	11163	87	50.58	111	2.194	1.540	1.860
	HEH	11539	85	72.44	85	1.173	0.579	**3.490**
Sequential AES-128 core in Spartan 3E	HCTR	5783	18	32.05	398	12.410		0.330
	HEH	6254	18	30.72	384	12.500		0.327

TABLE 10
A Performance Comparison of the EME Mode of Operation Using Several AES Encryption Cores Implemented in Software versus Our EME Reconfigurable Hardware Design

Design	Processor	Cycles/Sector	Throughput (Gbps)	fold Speedup
EME with AES in [22]	Intel Core i7 @2.67GHz	8532.48	1.282	**4.45**
EME with AES in [2]	Intel C2-Q @2.4GHz	11161.92	0.881	6.48
EME with AES in [29]	Intel C2-D @2.4GHz	13411.20	0.733	7.79
EME with AES in [28]	Athlon-64@2GHz	11214.72	0.730	7.82
EME with AES in [1]	Intel C2-Q @2.4GHz	14520.00	0.677	8.43
EME with AES in [23]	Athlon-64@2GHz	13136.64	0.624	9.15
EME here	Virtex IV@149MHz	107.00	5.710	1.00

8 Discussion

As we stated in Section 4, the design objective was to match the data rates of modern day disk controllers which are of the order of 3 Gbits/sec. Table 7 shows that using a sequential design, it is not possible to achieve such data rates, although 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 6, we see that while using an encryption/decryption pipeline AES core, the most efficient mode in terms of speed is HEH followed by HCTR, 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 HEH and HCTR are the best modes to use for this application.

In the case of the Virtex 5 family of devices, authors in [3] reported a fast and efficient AES design with an associated critical path smaller than the one corresponding to the Karatsuba multiplier used in this work. Since EME is the only TES mode that does not require a field multiplier block, one can conclude that if that AES design and technology is adopted, then EME will probably emerge as the fastest of the TES modes of operation studied here.

From Table 8, we see that the encryption operation of all the modes considered here except XCB can be significantly improved if an 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 the 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 likely to be performed more frequently.

Since the TESs are length-preserving encryption schemes, i.e., they are permutations, one can process data by encrypting-then-decrypting or by decrypting-then-encrypting without affecting the security guarantees provided by the modes. However, from the practical perspective, 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.

9 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 10).

The two most important building blocks utilized in our architectures are: a single AES-128 core designed with a 10-stage pipelined architecture and a fully parallel Karatsuba multiplier. From this setting, we think that the clock cycle count associated to each of the six TES algorithms studied in this paper, as illustrated in Fig. 3, cannot be further reduced. We also observe that according to our implementation results, the most influential factor in determining both area and time performances of the TES modes is the full AES pipelined core.

Although 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 is yet available for practical scenarios. In this paper, we presented optimized hardware implementation of six TESs. 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 discuss the achieved performance and hardware costs. 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.

From the different experiments conducted in this work, we conclude that the HEH scheme outperformed the other five TESs in most scenarios except when considering what we called the *encryption-only AES core scenario*, where EME emerged as the best of all the modes studied here. On the other side of our spectrum, the XCB was the TES that consistently performed in the last place with respect to all of them, area, timing, and the throughput per area metrics.

As a future work, we would like to explore the performance of the TESs when using as underlying block cipher other schemes different than AES. In this respect, the AES final candidate scheme Serpent would be of special interest, since along with Rijndael, this was the AES final candidate that performed better in hardware.

Acknowledgments

The authors thank the anonymous reviewers for their valuable suggestions. D. Chakraborty and F. Rodríguez-Henríquez acknowledge support from CONACyT through the CONACyT project numbers 90775 and 60240, respectively.

References

[1] D.J. Bernstein, "A State of the Art Message Authentication Code," http://cr.yp.to/mac.html, Feb. 2005.

[2] D.J. Bernstein and P. Schwabe, "New AES Software Speed Records," *Proc. Progress in Cryptology—Int'l Conf. Cryptology in India (INDOCRYPT '08)*, D.R. Chowdhury, V. Rijmen, and A. Das, eds., pp. 322-336, 2008.

[3] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," *Proc. Progress in Cryptology—Int'l Conf. Cryptology in Africa (AFRICACRYPT '08)*, S. Vaudenay, ed., pp. 16-26, 2008.

[4] D. Canright, "A Very Compact S-Box for AES," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '05)*, J.R. Rao and B. Sunar, eds., pp. 441-455, 2005.

[5] D. Chakraborty and M. Nandi, "An Improved Security Bound for HCTR," *Proc. Int'l Workshop Fast Software Encryption—(FSE '08)*, K. Nyberg, ed., pp. 289-302, 2008.

[6] D. Chakraborty and F. Rodríguez-Henríquez, "Block Cipher Modes of Operation from a Hardware Implementation Perspective," *Cryptographic Engineering*, Ç.K. Koç, ed., pp. 321-363, Springer, 2009.

[7] D. Chakraborty and P. Sarkar, "A New Mode of Encryption Providing a Tweakable Strong Pseudo Random Permutation," *Proc. Int'l Workshop Fast Software Encryption—(FSE '06)*, M.J.B. Robshaw, ed., pp. 293-309, 2006.

[8] D. Chakraborty and P. Sarkar, "HCH: A New Tweakable Enciphering Scheme Using the Hash-Counter-Hash Approach," *IEEE Trans. Information Theory,* vol. 54, no. 4, pp. 1683-1699, Apr. 2008.

[9] F. Charot, E. Yahya, and C. Wagner, "Efficient Modular-Pipelined AES Implementation in Counter Mode on ALTERA FPGA," *Proc. Int'l Conf. Field Programmable Logic and Application—(FPL '03),* pp. 282-291, 2003.

[10] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '03),* C.D. Walter, Ç.K. Koç, and C. Paar, eds., pp. 319-333, 2003.

[11] N. Ferguson, "AES-CBC + Elephant Diffuser: A Disk Encryption Algorithm for Windows Vista," Microsoft White Paper, http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a 0be4bbb36e/BitLockerCipher200608.pdf, 2006.

[12] Y. Fu, L. Hao, and X. Zhang, "Design of an Extremely High Performance Counter Mode AES Reconfigurable Processor," *Proc. Second Int'l Conf. Embedded Software and Systems (ICESS '05),* pp. 262-268, 2005.

[13] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '05),* J.R. Rao and B. Sunar, eds., pp. 427-440, 2005.

[14] S. Halevi, "EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data," *Proc. Progress in Cryptology—Int'l Conf. Cryptology in India (INDOCRYPT '04),* A. Canteaut and K. Viswanathan, eds., pp. 315-327, 2004.

[15] S. Halevi, "Invertible Universal Hashing and the TET Encryption Mode," *Proc. Advances in Cryptology—Ann. Int'l Cryptology Conf. (CRYPTO '07),* A. Menezes, ed., pp. 412-429, 2007.

[16] S. Halevi and P. Rogaway, "A Tweakable Enciphering Mode," *Proc. Advances in Cryptology—Ann. Int'l Cryptology Conf. (CRYPTO '03),* pp. 482-499, 2003.

[17] S. Halevi and P. Rogaway, "A Parallelizable Enciphering Mode," *Proc. Topics in Cryptology—The Cryptographers' Track at RSA Conf. (CT-RSA '04),* T. Okamoto, ed., pp. 292-304, 2004.

[18] S.F. Hsiao and M.C. Chen, "Efficient Substructure Sharing Methods for Optimising the Inner-Product Operations in Rijndael Advanced Encryption Standard," *IEE Proc. Computer and Digital Technology,* vol. 152, no. 5, pp. 653-665, Sept. 2005.

[19] IEEE Security in Storage Working Group (SISWG), PRP Modes Comparison IEEE p1619.2, IEEE Computer Society, http://siswg.org/, Nov. 2008.

[20] Y. Inoguchi, "Outline of the Ultra Fine Grained Parallel Processing by FPGA," *Proc. Seventh Int'l Conf. High Performance Computing and Grid in Asia Pacific Region (HPCAsia '04),* pp. 434-441, July 2004.

[21] K. Järvinen, M. Tommiska, and J. Skyttä, "Comparative Survey of High-Performance Cryptographic Algorithm Implementations on FPGAs," *IEE Proc. Information Security,* vol. 152, no. 1, pp. 3-12, Oct. 2005.

[22] E. Käsper and P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '09),* C. Clavier and K. Gaj, eds., pp. 1-17, 2009.

[23] H. Lipmaa, Fast Implementations: Complete AES (Rijndael) Library, http://home.cyber.ee/helger/implementations/, Oct. 2006.

[24] M. Liskov, R.L. Rivest, and D. Wagner, "Tweakable Block Ciphers," *Proc. Advances in Cryptology—Ann. Int'l Cryptology Conf. (CRYPTO '02),* pp. 31-46, 2002.

[25] E. López-Trejo, F. Rodríguez-Henríquez, and A. Díaz-Pérez, "An Efficient FPGA Implementation of CCM Mode Using AES," *Proc. Int'l Conf. Information Security and Cryptology—(ICISC '05),* pp. 208-215, Dec. 2005.

[26] C. Mancillas-López, D. Chakraborty, and F. Rodríguez-Henríquez, "Efficient Implementations of Some Tweakable Enciphering Schemes in Reconfigurable Hardware," *Proc. Progress in Cryptology—Int'l Conf. Cryptology in India (INDOCRYPT '07),* pp. 414-424, 2007.

[27] C. Mancillas-Lopez, D. Chakraborty, and F. Rodriguez-Henriquez, "Reconfigurable Hardware Implementations of Tweakable Enciphering Schemes," Report 2007/437, Cryptology ePrint Archive, http://eprint.iacr.org/, 2007.

[28] M. Matsui, "How Far Can We Go on the x64 Processors?" *Proc. Int'l Workshop Fast Software Encryption—(FSE '06),* M.J.B. Robshaw, ed., pp. 341-358, 2006.

[29] M. Matsui and J. Nakajima, "On the Power of Bitslice Implementation on Intel Core2 Processor," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '07),* P. Paillier and I. Verbauwhede, eds., pp. 121-134, 2007.

[30] D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM), Submission to NIST Modes of Operation Process," http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec. pdf, Jan. 2004.

[31] D.A. McGrew and S.R. Fluhrer, "The Extended Codebook (XCB) Mode of Operation," Report 2004/278, Cryptology ePrint Archive, http://eprint.iacr.org/, 2004.

[32] D.A. McGrew and S.R. Fluhrer, "The Security of the Extended Codebook (XCB) Mode of Operation," *Proc. Ann. Workshop Selected Areas in Cryptography,* C.M. Adams, A. Miri, and M.J. Wiener, eds., pp. 311-327, 2007.

[33] D.A. McGrew and J. Viega, "Arbitrary Block Length Mode," http://grouper.ieee.org/groups/1619/email/pdf00005.pdf, 2004.

[34] D.A. McGrew and J. Viega, "The Security and Performance of the Galois/Counter Mode (GCM) of Operation," *Proc. Progress in Cryptology—Int'l Conf. Cryptology in India (INDOCRYPT '04),* A. Canteaut and K. Viswanathan, eds., pp. 343-355, 2004.

[35] P. Rogaway, M. Bellare, and J. Black, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," *ACM Trans. Information and System Security,* vol. 6, pp. 365-403, 2003.

[36] F. Rodríguez-Henríquez and Ç. K. Koç, "On Fully Parallel Karatsuba Multipliers for GF(2^m)," *Proc. Int'l Conf. Computer Science and Technology (CST '03),* pp. 405-410, May 2003.

[37] G.P. Saggese, A. Mazzeo, N. Mazzocca, and A.G.M. Strollo, "An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm," *Proc. Int'l Conf. Field Programmable Logic and Application—(FPL '03),* P.Y.K. Cheung, G.A. Constantinides, and J.T. de Sousa, eds., pp. 292-302, 2003.

[38] P. Sarkar, "Improving upon the TET Mode of Operation," *Proc. Int'l Conf. Information Security and Cryptology—(ICISC '07),* K.-H. Nam and G. Rhee, eds., pp. 180-192, 2007.

[39] A. Satoh, T. Sugawara, and T. Aoki, "High-Performance Hardware Architectures for Galois Counter Mode," *IEEE Trans. Computers,* vol. 54, no. 7, pp. 917-930, July 2009.

[40] Seagate Technology, "Internal 3.5-Inch (SATA) Data Sheet," http://www.seagate.com/www/en-us/products, 2010.

[41] V. Shoup, "On Fast and Provably Secure Message Authentication Based on Universal Hashing," *Proc. Advances in Cryptology—Ann. Int'l Cryptology Conf. (CRYPTO '96),* N. Koblitz, ed., pp. 313-328, 1996.

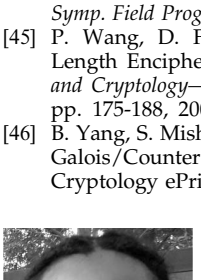
[42] K. Siozios, G. Koutroumpezis, K. Tatas, D. Soudris, and A. Thanailakis, "DAGGER: A Novel Generic Methodology for FPGA Bitstream Generation and Its Software Tool Implementation," *Proc. 19th Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* 2005.

[43] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems—(CHES '03),* C.D. Walter, Ç.K. Koç, and C. Paar, eds., pp. 334-350, 2003.

[44] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90nm Low-Power FPGA for Battery-Powered Applications," *Proc. Int'l Symp. Field Programmable Gate Arrays—(FPGA '06),* pp. 3-11, 2006.

[45] P. Wang, D. Feng, and W. Wu, "HCTR: A Variable Input-Length Enciphering Mode," *Proc. Int'l Conf. Information Security and Cryptology—(CISC '05),* D. Feng, D. Lin, and M. Yung, eds., pp. 175-188, 2005.

[46] B. Yang, S. Mishra, and R. Karri, "A High Speed Architecture for Galois/Counter Mode of Operation (GCM)," Report 2005/146, Cryptology ePrint Archive, http://eprint.iacr.org/, 2010.



Cuauhtemoc Mancillas-López received the BE degree in electronic and communications engineering from ESIME-Instituto Politécnico Nacional (IPN), Mexico, in 2004, and the MSc degree in computer science from CINVESTAV-IPN, Mexico, where he is currently working toward the PhD degree at the Computer Science Department. His current research interests include design and analysis of provably secure symmetric encryption schemes, efficient software/hardware implementations of cryptographic primitives, and computational arithmetic.

Debrup Chakraborty received the BE degree in mechanical engineering from Jadavpur University, Kolkata, India, in 1997, and the MTech and PhD degrees in computer science from the Indian Statistical Institute, Kolkata, in 1999 and 2005, respectively. Currently, he is a researcher in the Computer Science Department of Centro de Investigaciones y Estudios Avanzados del IPN, Mexico City, Mexico. His current research interests includes design and analysis of provably secure symmetric encryption schemes, efficient software/hardware implementations of cryptographic primitives, pattern recognition, and neural networks.

Francisco Rodríguez-Henríquez received the BSc degree in electrical engineering from the University of Puebla, México, in 1989, the MSc degree in electrical and computer engineering from the National Institute of Astrophysics, Optics and Electronics (INAOE), Mexico, in 1992, and the PhD degree in electrical and computer engineering from Oregon State University, in 2000. Currently, he is a professor (CINVESTAV-3B researcher) in the Computer Science Department of CINVESTAV-IPN, Mexico City, Mexico, which he joined in 2002. His major research interests are in cryptography, finite field arithmetic, and hardware implementation of cryptographic algorithms.