Efficient Implementations of Some Tweakable Enciphering Schemes in Reconfigurable Hardware

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

Computer Science Departament,
Centro de Investigación y Estudios Avanzados del IPN,
Av. Instituto Politécnico Nacional No. 2508, México D.F.

Abstract. We present optimized FPGA implementations of three tweakable enciphering schemes, namely, HCH, HCTR and EME using AES-128 as the underlying block cipher. We report performance timings and hardware resources occupied by these three modes when using a fully pipelined AES core and a sequential AES design. Our experimental results suggest that in terms of area HCTR, HCH and HCHfp (a variant of HCH) require more area than EME. However, HCTR performs the best in terms of speed followed by HCHfp, EME and HCH.

1 Introduction

A tweakable enciphering scheme (TES) is a specific kind of block-cipher mode of operation which provides a strong pseudorandom permutation (SPRP). A fully defined TES for arbitrary length messages using a block cipher was first presented in [9]. In [9] it was also stated that a possible application area for such encryption schemes could be low level disc encryption, where the encryption/decryption algorithm resides on the disc controller which has access to the disc sectors but has no knowledge of the disk's high level partitions such as directories, files, etc. Furthermore, it was suggested in [9] that sector addresses could be used as tweaks. Because of the specific nature of this application, a length preserving enciphering scheme is required and under this scenario, a SPRP 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 [9], EME [10], EME* [7] falls under the Encrypt-Mask-Encrypt group. PEP [3], TET [8], HEH [17] falls under the Hash-ECB-Hash type and XCB [15], HCTR [19], HCH [4] falls under the Hash-Counter-Hash type. Although about nine different constructions of different TES have been proposed, we are not aware of any work reporting experimental performance data of any of these schemes. A comparative performance comparison of these modes is very necessary given the current efforts of IEEE security in storage working group [12] towards standardization of TES.

K. Srinathan, C. Pandu Rangan, M. Yung (Eds.): Indocrypt 2007, LNCS 4859, pp. 414–424, 2007.

A speculative performance comparison of the EME*, XCB, HCH and TET modes of operation in hardware is provided in [8]. This comparison assumes the same hardware implementation setting reported in [1], where a fully-parallel $GF(2^n)$ field multiplier capable of performing one multiplication in one clock cycle was implemented at a hardware cost in area of about three times the cost associated with one AES round function. The AES core was 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.

In this paper we present performance data for hardware implementation of three TES. Our implementations are optimized for the application of low level disc encryption. The modes we select for our comparative study are EME, HCH and HCTR. Also we provide performance data for a variant of HCH which is called HCHfp, which is particularly useful for disk encryption. We use AES-128 as the underlying block-cipher, and 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 three modes. Due to lack of space in this paper we discuss in detail the construction and implementation of HCH only, but present performance data of all the modes we implemented. The full implementation details of the three modes will appear in the full version of the paper.

Notations. In the rest of the paper by $E_K(\,)$ we shall mean a n bit block cipher call with key K. 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 of the field $GF(2^n)$. If X and Y are n bit strings then by $X \oplus Y$ and XY we shall mean addition and multiplication in the field respectively. By xX we would represent the multiplication of X by the polynomial x.

2 The Schemes

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

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

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, the counter mode is defined as

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

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 $(P_1, \ldots, P_m)$.

Fig. 1. 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 $r \leq n$, and l is the length of the message.

Where $S_i = S \oplus \mathsf{bin}_n(i)$. The complete encryption and decryption algorithm of HCH is given in Fig. 1.

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 [5] a modification of HCH is also proposed which is called HCHfp. HCHfp can only be used in those applications where the message length is fixed. This construction simplifies the general HCH construction and requires one less block-cipher call, but it requires two separate keys for the hash and the block-cipher. As in HCHfp the hash key is not dependent on the tweak so pre-computation for calculating the hash is also possible. 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 can be found in [5].

All variants of HCH are provably secure and the authors guarantee that the advantage of any computationally 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 ciphertext blocks the adversary has access to, and δ denotes the advantage of an adversary to distinguish the underlying block-cipher from a random permutation.

The structure of HCTR is similar to that of HCH with some important differences. 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 utilizes 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 and it requires three less block cipher calls than HCH and 2 less block cipher calls than HCHfp but it needs four more multiplications than both HCH and HCHfp. A full description of HCTR can be found in [19].

EME stands for ECB-Mask-ECB (EME)[10]. 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. EME requires $2m + 2$ block cipher calls for encrypting a 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 disc encryption EME appears to be sufficient, as generally disk sectors lengths are less than 2KB and their lengths are multiples of 128 bits. EME has a security bound of $O(\sigma^2)/2^n + \delta$. A full description of EME can be found in [10].

3 Design Decisions

For implementing all three schemes we chose the underlying block cipher as AES-128. As mentioned earlier the designs that we present here are directed towards the application of disk sector encryption. 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 the block length of the block cipher.

The speed of a low level disk encryption algorithm must meet the current possible data rates of disc controllers. With emerging technologies like serial ATA and Native Command Queuing (NCQ) the modern day discs can provide data rates around 3Giga-bits per second[18]. Thus, the design objective should be to achieve an encryption/decryption speed which matches this data rate.

The modes HCH and HCTR 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 design must meet the desired speed. Out of many possible designs reported in the literature [13,6,2,11] we decided to design the AES core so that a 10-stage pipeline architecture could be used to implement two different functionalities: the counter mode, and the encryption of one single block that we will call in the rest of this paper as 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 one encrypted block in each subsequent clock-cycle. It is worth mentioning that in the literature, several ultra fast designs with up to 70 pipeline stages have been reported [13], 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, such pipeline designs are not suitable for our target application.

The main building block needed for implementing the polynomial hash of the HCH and HCTR modes is an efficient multiplier in $GF(2^{128})$. 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 [16]. This time efficient multiplier occupies about 2 times the hardware resources required by one single AES round. Because of this, the total hardware area required by HCTR and HCH are significantly more than EME (which does not require multipliers). A more compact multiplier selection would yield significantly lower speeds which violates the design objective of optimizing for speed.

The specifications of both the HCTR 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 in software are discussed in [14,1]. These techniques can be 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.

We implemented the schemes on a FPGA device which operates at lower frequencies than true VLSI circuits. Thus the throughput that we obtained probably can be much improved if we use the same design strategies on a different technology. Our target device was a XILINX Virtex 4, xc4v1x100-12FF1148.

4 The Design Overviews

In this Section we give a carefully analysis of the data dependencies of HCH and explain how we exploit the parallelism present in the algorithm. Similar analysis for HCTR and EME can be found in the extended version of this paper.

In the analysis which follows we assume the message to be of 512 bytes (32 AES blocks). Also, 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.

Referring to the Algorithm of Fig. 1 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 $Z = R^{31}P_2 \oplus \ldots \oplus RP_{32}$. 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 block cipher 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 2 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.

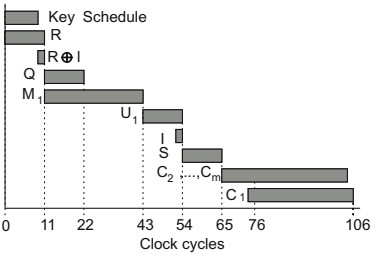

Fig. 2. HCH Time Diagram

A similar analysis can be done in case of HCTR and EME. Exploiting the parallelism present in these algorithms to the full extent we obtain that for HCTR a valid output will be ready after the cycle 55 and a whole disk sector will be ready in the cycle 88. For EME the first block of valid output would be produced after 75 clock cycles and the whole sector would be ready after 106 clock-cycles.

5 Implementation

Due to lack of space, in this Section we only discuss the design details of the basic control unit of HCH. The other implementation details along with the details for HCTR and EME implementation will appear in the full version of this paper, but we shall provide performance data for all the modes in Section 6.

Fig. 3 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.

Additionally, a hash function is also required as one of the main building blocks. The architecture operation is synchronized through a control unit that performs the adequate sequence of operations in order to obtain a valid output.

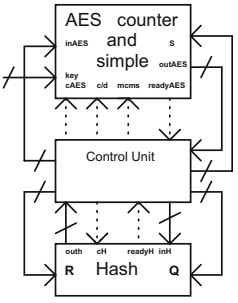

Fig. 3. HCH General Architecture

The HCH control unit architecture is shown in Fig. 4. 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 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

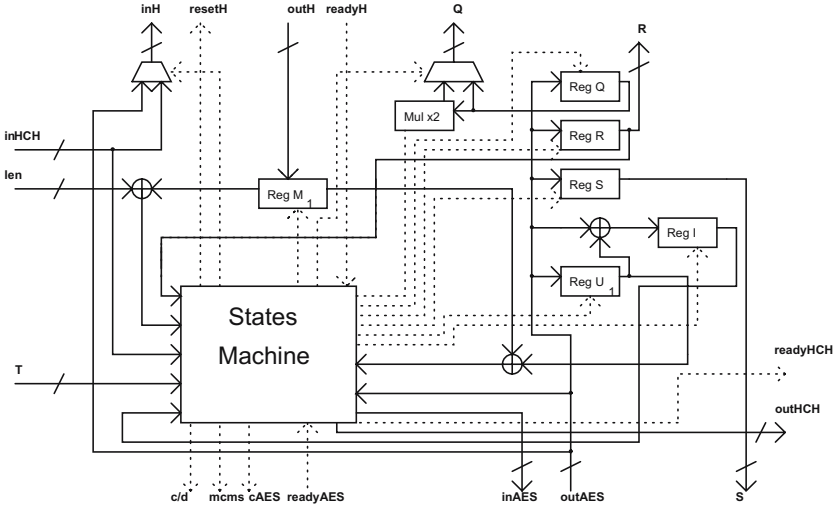

Fig. 4. HCH Control Unit Architecture

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 uses 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 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.

6 Results

In this section we provide the performance results obtained from our implementations. We will measure the performances based on the following criteria: time taken for encrypting 32 blocks of data, the latency, i.e., the time required to produce the first block of output, 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, $TPA = [(slices + 128 \cdot BRAMS) \cdot time]^{-1}$. For a given design, a high TPA indicates high efficiency, i.e., a good performance/area tradeoff.

In Table 1 we show the performance of the basic building blocks of the architectures, i.e, the performance of one AES round and one multiplier. Table 1 shows that considering the B-RAMs and slices the size of our multiplier circuit is about two times the size of a AES round. The critical path delay of the AES round is more than the multiplier, so the AES round determines the critical path in all the implementations.

Table 1. Performance of AES round and multiplier

Design	**Slices**	**B-RAM**	**Critical Path**(nS)
AES round	1215	8	10.998
multiplier	3223	-	9.85

Table 2 gives the performance of the full AES (both a sequential and pipelined architecture), it also shows the performance of the two different hash functions for HCH and HCTR. Column 5 of Table 2 shows the throughput. Throughput does not carry the usual meaning in case of the hash functions, as they produce a 128 bit output irrespective of the input size. By throughput of the hash function we mean the number of bits they can process per unit time. The sequential

Table 2. Performance of the AES and Hash implementations

Method	Slices	B-RAM	Frequency (MHz)	Throughput (GBits/s)
AES-Sequential	1301	18	81.967	1.049
AES-Pipeline	6368	85	83.88	10.736
Hash-HCTR	3986	-	101.08	12.15
Hash-HCH	4014	-	101.45	12.98

AES gives significantly poor throughput and both hash functions have better throughput than the AES-pipeline.

In Table 3 we show the experimental results for the four modes of operations implemented using a pipelined AES core. As we implemented both encryption and decryption functionalities in the same circuit and due to the symmetry of the algorithms the timings for encryption and decryption operations are the same. Note that the number of clock-cycles reported in Table 3 are one more than those estimated in Section 4, this is because in the true implementations one clock cycle is lost due to the initial reset operation. From Table 3 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. The most costly mode in terms of area is HCH since it requires 6 registers in contrast to the three registers required by the HCTR. Additionally, HCH has more possible inputs for its AES building block. In terms of speed, the fastest mode is HCTR since it only utilizes one AES block cipher call in sequential mode, whereas HCH requires a total of four such calls (although only three have consequences in terms of clock cycles since the other one is masked with the computation of the hash function). HCHfp is better than both EME and HCH in terms of speed.

Table 3. Hardware costs of the HCTR, HCH and EME modes with an underlying AES full pipeline core: The time and clock-cycles are the time required to encrypt 32 blocks

Mode	Slices	B-RAM	Frequency (MHz)	Clock Cycles Cycles	Time (μS)	Latency (μS)	Throughput GBits/Sec	TPA
HCH	13755	85	65.939	107	1.622	1.167	2.46	24.42
HCHfp	12970	85	66.500	96	1.443	0.992	2.83	29.05
HCTR	12068	85	79.65	89	1.117	0.703	3.66	39.81
EME	10120	87	67.835	107	1.576	1.120	2.64	29.85

In Table 4 we show the four modes of operation when using a sequential implementation of the AES core. In a sequential architecture, EME is the most inefficient mode in terms of latency due to the two costly block cipher passes that require eleven clock cycles per block. Hence, a significant increment in the required number of clock cycles is observed for the EME mode. This situation does not occur in HCTR or in HCH 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.

Table 4. Hardware costs of the HCTR, HCH and EME modes with an underlying sequential 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	15.73
HCTR	7006	18	77.737	388	4.991	0.820	21.53
EME	5053	20	65.922	716	10.861	0.377	12.09

Discussion: As we stated in Section 3 the design objective would be to match the data rates of modern day disk controllers which are of the order of 3Gbits/sec. Table 4 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 3 we see that the most efficient mode in terms of speed is HCTR followed by HCHfp, EME and HCH. 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 guarantees that HCTR provides is quite weak as it have a cubic security bound. Thus, among the different modes that we implemented, and in view of all these constraints, HCHfp should probably be the most preferred mode.

7 Conclusion

We presented optimized implementation of three TES. To our knowledge this is the first work to report real performance data of any TES on hardware. There are many other TES schemes which needs to be implemented and then a true performance comparison would be possible. This performance comparison will of course help in selection of the best mode. This work can be seen as a first step towards this objective.

References

1. Bo Yang, R.K., Mishra, S.: A high speed architecture for galois/counter mode of operation (gcm). Cryptology ePrint Archive, Report 2005/146 (2005), `http://eprint.iacr.org/`
2. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)
3. Chakraborty, D., Sarkar, P.: A New Mode of Encryption Providing a Tweakable Strong Pseudo-random Permutation. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 293–309. Springer, Heidelberg (2006)

4. Chakraborty, D., Sarkar, P.: HCH: A New Tweakable Enciphering Scheme Using the Hash-Encrypt-Hash Approach. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 287–302. Springer, Heidelberg (2006)
5. Chakraborty, D., Sarkar, P.: HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. Cryptology ePrint Archive, Report 2007/028 (2007), http://eprint.iacr.org/
6. Good, T., Benaissa, M.: AES on FPGA from the Fastest to the Smallest. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 427–440. Springer, Heidelberg (2005)
7. Halevi, S.: EME*: Extending EME to handle arbitrary-length messages with associated data. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 315–327. Springer, Heidelberg (2004)
8. Halevi, S.: TET: A wide-block tweakable mode based on Naor-Reingold. Cryptology ePrint Archive, Report 2007/014 (2007), http://eprint.iacr.org/
9. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
10. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
11. Hsiao, S.F., Chen, M.C.: 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 (2005)
12. IEEE Security in Storage Working Group (SISWG). PRP modes comparison IEEE p1619.2. IEEE Computer Society (March 2007), Available at http://siswg.org/
13. Jarvinen, K., Tommiska, M., Skytta, J.: Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. Information Security, IEE Proceedings 152(1), 3–12 (2005)
14. McGrew, D., Viega, J.: 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
15. McGrew, D.A., Fluhrer, S.R.: The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278 (2004), http://eprint.iacr.org/
16. Rodríguez-Henríquez, F., Koç, Ç.: On fully parallel karatsuba multipliers for GF(2^m). In: International Conference on Computer Science and Technology CST 2003, pp. 405–410. Acta Press (May 2003)
17. Sarkar, P.: Improving upon the TET mode of operation. Cryptology ePrint Archive, Report 2007/317 (2007), http://eprint.iacr.org/
18. Seagate Technology. Internal 3.5-inch (sata) data sheet, Available at: http://www.seagate.com/docs/pdf/datasheet/disc/ds_internal_sata.pdf
19. Wang, P., Feng, D., Wu, W.: HCTR: A variable-input-length enciphering mode. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 175–188. Springer, Heidelberg (2005)