Low-Cost Concurrent Error Detection for GCM and CCM

Xiaofei Guo and Ramesh Karri

New York University
xg243@nyu.edu, rkarri@nyu.edu

Abstract. In many applications, encryption alone does not provide enough security. To enhance security, dedicated authenticated encryption (AE) mode are invented. Galios Counter Mode (GCM) and Counter with CBC-MAC mode (CCM) are the AE modes recommended by the National Institute of Standards and Technology. To support high data rates, AE modes are usually implemented in hardware. However, natural faults reduce its reliability and may undermine both its encryption and authentication capability. We present a low-cost concurrent error detection (CED) scheme for 7 AE architectures. The proposed technique explores idle cycles of the AE mode architectures. Experimental results shows that the performance overhead can be lower than 100% for all architectures depending on the workload. FPGA implementation results show that the hardware overhead in the 0.1-23.3% range and the power overhead is in the 0.2-23.2% range. ASIC implementation results show that the hardware overhead in the 0.1-22.8% range and the power overhead is in the 0.3-12.6% range. The underlying block cipher and hash module need not have CED built in. Thus, it allows system designers to integrate block cipher and hash function intellectual property from different vendors.

1 Introduction

Symmetric key encryption is widely used for confidential communications. The most widely used symmetric key encryption is Advanced Encryption Standard [36]. However, if an attacker can intercept, tamper with, and send ciphertexts to the receiver, symmetric key encryption can fail catastrophically, allowing the attacker to partially or completely decrypt messages [4, 6, 14, 40]. These attacks require the attacker to only see an error message from the receiver.

A simple countermeasure for these attacks is to use message authentication code (MAC) such as HMAC on ciphertexts [9]. Encrypting the message and then generating a MAC is provably secure as long as the encryption and MAC have certain properties [25]. A designer should also consider other critical implementation issues; i.e., the message space, the padding, the key setup, the nonce requirement, parallelizability, whether it is online[1], preprocessing, provable security, and intellectual property restrictions. Intellectual property restriction is a very important consideration, because companies which own encryption and authentication primitives can combine them freely for authenticated encryption. For these reasons, specialized block cipher modes of operation, called Authenticated Encryption (AE) mo- des, are invented. AE modes handle both the encryption and authentication.

To satisfy the security and speed requirements of various information disciplines, e.g., networking, telecommunications, database systems, and mobile applications, many cryptographic systems are implemented as cryptographic accelerators. The complexity of these hardware implementations is raising concerns regarding their security and reliability. As

[1] Whether one needs to know the message size before encrypting it.

VLSI technology keeps scaling, the chips will have more soft errors [33], process variation [12], and aging [3]. Cryptographic chips are sensitive to faults in hardware [7]. Radiation, heat, incorrect voltages, and atypical clock rates all cause cryptographic devices to malfunction [22].

In order to protect cryptographic hardware from fau- lts, concurrent error detection (CED) has been developed. There are four different types of CED, i.e., hardware, time, information, and hybrid redundancies. Hardware redundancy duplicates the function and detects faults by comparing the outputs of two copies. Time redundancy computes the same input twice using the same function and compares the results. Information redundancy techniques are based on error detecting codes (EDC). A few check bits are generated from the input message; then they propagate along with the input message and are finally validated when the output message is generated.

For AES, information redundancy techniques such as parity and robust code are proposed. Parity for AES has been studied extensively [10, 15, 29, 32, 34, 43]. Their hardware overheads are in the range of 8-26%. Robust code provides uniform fault coverage for different fault distributions [23]. However, to ensure the uniform fault coverage, its hardware overhead is over 70%. a generic hybrid redundancy scheme uses decryption to check encryption and vice versa [24]. Although this scheme has near optimal fault coverage, it requires both encryption and decryption to be on the same chip. Although this is a reasonable assumption for standalone AES chips, it is not the case if the cipher uses block cipher counter mode (CTR), cipher block chaining mode (CBC), or cipher feedback mode (CFB) [17]. Another example hybrid redundancy merges the encryption and decryption datapath and time multiplexing the encryption and decryption operations [39]. But such time multiplexing is difficult and requires many registers to store results if the latency of round operations are different.

For Galois Field (GF) multipliers, a parity CED for GF multipliers with arbitrary irreducible polynomials is developed with a hardware overhead of 10.29% [8]. The REcomputing using Shifted Operand (RESO) is applied to polynomial basis multipliers [27]. However, this technique can only be used for irreducible polynomials with all ones or equally spaced ones. Thus, it is not applicable to Galois Field (GF) multiplier-adders (GHASH).

All these schemes focuses on protecting individual cryptographic primitives. Although one can use separate CEDs for AES and GHASH, a straightforward combination of CED-enabled AES and CED-enabled GHASH may not yield optimal throughput and hardware utilization. Moreover, because it is desirable that AE modes remains free of intellectual property restrictions, one should be able to use AES and GHASH from different IP vendors interchangeably to build AE modes without compromising reliability. Therefore, primitive specific CED techniques are restrictive and hence not appropriate for system-level design such as AE modes. As more functionalities are integrated into cryptographic modules, system-level CED trade-offs need to be made. Our contributions are:

- A low-cost CED techniques for AE modes; i.e., GCM and CCM. The proposed technique is independent of the implementation of the underlying intellectual property so that the designer can combine modules from different vendors.
- We improve the CED using redundant cycles to achi- eve 100% fault coverage, and the performance overhead is in the range of 0% to 100% depending on the workload.

This paper is organized as follows: Section 2 gives the background of GCM and CCM. Section 3 analyzes 6 GCM architectures and the proposed CEDs. Section 4 shows the fault coverage and implementation results. Section 5 concludes the paper.

Table 1: Authenticated Encryption Modes.

Arch.	Encryption	Authentication
GCM	CTR	Multiply-add unit
CCM	CTR	CBC-MAC

2 Authenticated Encryption (AE) Mode

National Institute of Standards and Technology (NIST) standardized several AE modes based on block ciphers, i.e., Galois Counter Mode (GCM), and Counter Mode with CBC MAC (CCM) [2]. Both of them are based on block ciphers. Other AE modes that are submitted to NIST includes EAX, Carter-Wegman + CTR mode (CWC), and Offset Codebook Mode (OCB). Alternative AE mode that uses permutation are also proposed [11]. AE mode is an active research field and there are other initiatives for new proposals [1].

GCM is online, parallelizable, and patent-free. GCM generates ciphertexts and an authentication tag simultaneously. It uses CTR mode [35] and a hash function over $GF(2^{128})$ to generate the tag. GCM mainly consists of AES modules and Galois Field (GF) multiplier-adders (GHASH). GHASH uses GF multiplier and GF adder. Many security standards have adopted GCM with AES as the underlying building block cipher and GHASH as the hash function, including in-car systems [5], the IEEE 802.1AE for frame data encryption in the ethernet [19], the IEEE P1619.1 for hard disk encryption [21], and the RFC 4106 for payload encryption in IPSec [41]. CCM is patent-free. It is not online, which means you have to know the size of your message before you start encrypting it. CCM with AES is adopted in IEEE 802.15.4 standard to provide link-layer security in wireless network [20]. EAX is similar to CCM. It is patent-free. Unlike CCM, it is online. OCB has a patent on it [37]. CWC is similar to GCM but it is much slower [26]. NIST recommended two AE modes, i.e., GCM and CCM. Therefore we focus on GCM and CCM.

2.1 GCM

An example of GCM[2] authenticated encryption is shown in Fig. 1. GCM receives a secret key K, a 96-bit initial vector IV, m-block authentication data A_1, A_2,..., A_{m-1}, A_m^*, and n-block plaintext P_1, P_2,..., P_{n-1}, P_n^*. Each block has 128 bits. When a final block is shorter than 128 bits, the rest of the block is padded with zeros. The IV is concatenated with 31 zeros and a one as the initial counter value[3]. The v and u bits in the following equations represent the bit width of the last blocks A_m^* and P_n^*, respectively. The encryption result of IV is XORed with the hash result, and the first t bits of the result become the authenticated tag T. The authenticated encryption operation is:

$$
\begin{aligned}
H &= Enc(K, 0^{128}) \\
Y_0 &= IV||0^{31}1 \begin{cases} IV||0^{31}1 & \text{if len}(IV) = 96 \\ GHASH(H, \{\}, IV) & \text{otherwise.} \end{cases} \\
C_0 &= Enc(K, Y_0) \\
Y_i &= Y_{i-1} + 1 \qquad i = 1, ..., n \\
C_i &= P_i \oplus Enc(K, Y_i) \qquad i = 1, ..., n{-}1 \\
C_n^* &= MSB_u(P_n \oplus Enc(K, Y_i)) \\
T &= MSB_t(GHASH(H, A, C) \oplus C_0).
\end{aligned} \tag{1}
$$

[2] A comprehensive description of GCM is in [30].

[3] Concatenation is denoted as ||

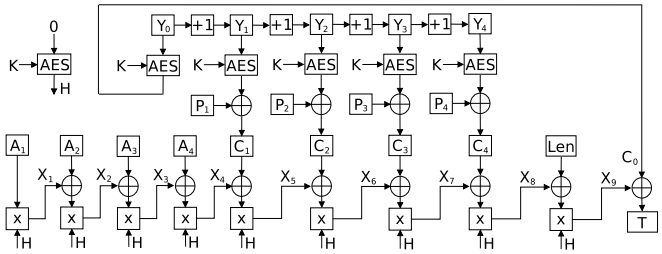


Fig. 1: GCM with authentication blocks $m = 4$ and encryption blocks $n = 4$.

The hash function $GHASH()$ defined over the field repeats multiplication and addition is:

$$X_i = \begin{cases} 0 & i{=}0 \\ (X_{i-1} \oplus A_i) \cdot H & i{=}1, ..., m{-}1 \\ (X_{m-1} \oplus (A_m^* || 0^{128-v}) \cdot H & i{=}m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & i{=}m{+}1, ..., m{+}\ n{-}1 \\ (X_{m+n-1} \oplus (C_n^* || 0^{128-u})) \cdot H & i{=}m{+}n \\ (X_{m+n} \oplus (len(A)||len(C))) \cdot H & i{=}m{+}n{+}1 \end{cases} \tag{2}$$

The function len() returns a 64-bit string describing the number of bits in its input, with the least significant bit on the right. The final hash value X_{m+n+1} is:

$$X_{m+n+1} = GHASH(H, A, C) \tag{3}$$

The irreducible polynomial in $GF(2^{128})$ used in GCM is:

$$g(x) = x^{128} + x^7 + x^2 + x + 1 \tag{4}$$

In decryption, the authenticated data A and the ciphertext C are used to recompute an authenticated tag $T^{'}$:

$$\begin{aligned} H &= Enc(K, 0^{128}) \\ Y_0 &= IV||0^{31}1 \\ C_0 &= Enc(K, Y_0) \\ Y_i &= Y_{i-1} + 1 \qquad\qquad i = 1, ..., n \\ P_i &= C_i \oplus Enc(K, Y_i) \qquad i = 1, ..., n \\ T^{'} &= MSB_t(GHASH(H, A, C) \oplus C_0). \end{aligned} \tag{5}$$

Then $T^{'}$ is compared with the original tag T to verify the data authenticity.

2.2 CCM

CCM receives a secret key K, a 128-bit initial vector IV, m-block authentication data A_1, A_2 ,..., A_{m-1}, A_m, and n-block plaintext P_1, P_2, ..., P_{n-1}, P_n. Each block has 128 bits. When a final block is shorter than 128 bits, the rest of the block is padded with zeros. The v and u bits in the following equations represent the bit width of the last blocks A_m and P_n, respectively. The encryption result of IV is XORed with the hash result, and the first t bits of the result become the authenticated tag T. The CCM authenticated encryption

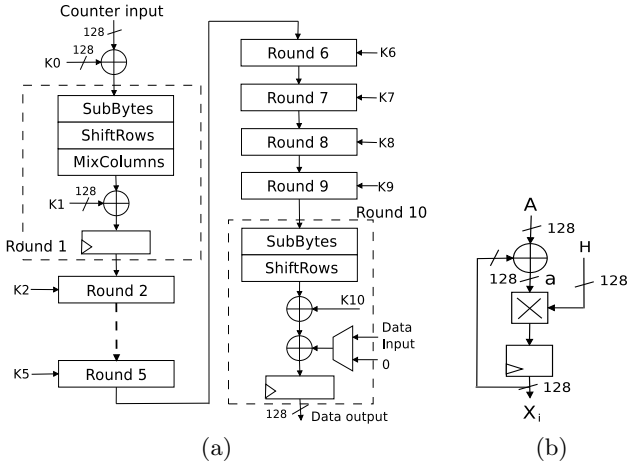


Fig. 2: (a) Fully pipelined AES architecture. (b) GHASH architecture with parallel GF multiplier-adder.

operation is:

$$\begin{aligned} Y_0 &= IV \\ C_0 &= P_0 \oplus Enc(K, Y_0) \\ Y_i &= Y_{i-1} + 1 \qquad\qquad i = 1, ..., n \\ C_i &= P_i \oplus Enc(K, Y_i) \qquad i = 1, ..., n{-}1 \\ C_n^* &= MSB_u(P_n \oplus Enc(K, Y_i)) \\ T &= MSB_t(MAC(H, A, C) \oplus C_0) \end{aligned} \tag{6}$$

The $MAC()$ function uses CBC MAC mode defined as:

$$X_{i+1} = \begin{cases} Enc(K, A_i) & i{=}0 \\ Enc(K, X_i \oplus A_i) & i{=}1, ..., m{-}1 \\ Enc(K, X_i \oplus A_m^* || 0^{128-v}) & i{=}m \\ Enc(K, X_i \oplus C_i) & i{=}m{+}1, ..., m{+}n{-}1 \\ Enc(K, X_i \oplus C_n^* || 0^{128-u}) & i{=}m{+}n \end{cases} \tag{7}$$

2.3 AES primitives

We consider 128-bit AES as specified by NIST [36]. AES encrypts a 128-bit plaintext into a 128-bit ciphertext with a user key using 10 nearly identical rounds plus an initial special round (round 0). We use 3 different AES architectures with different throughput and area requirements: (a) Fully pipelined (AES-P, shown in Fig. 2(a)), (b) iterative (AES-I) [38], (c) pipelined loop (AES-PL) [38]. The throughput of the 3 techniques are 1, 10, 4 cycles, respectively[4].

2.4 GHASH primitives (GF multiplier-adders)

GCM uses a 128-bit $GF(2^{128})$ multiplier-adder for the hash function GHASH defined in equation (2). GHASH uses a customized irreducible polynomial on $GF(2^{128})$, defined in (4).

[4] Because once the AES architecture and GHASH architecture in GCM (or CBC MAC architecture in CCM) is fixed, the clock frequency will be fixed. We compare the performance overhead of the CED and its corresponding non-CED implementation using clock cycles.

Table 2: GCM hardware architectures with different AES and GHASH modules. a. The throughput of the block multiplier-adder is 2 if $i = 1$ and is $\lceil \frac{i-2}{4} \rceil + 3$ if $i > 1$.

Arch.	AES	GHASH
GCM 1 [45]	1-clock AES-P & 10-clock AES-I	1-clock mult-add.
GCM 2 [38]	1-clock AES-P	1-clock mult-add.
GCM 3 [38]	10-clock AES-I	8-clock mult-add.
GCM 4 [38]	4-clock AES-PL	4-clock mult-add.
GCM 5 [38]	10-clock AES-I × 4	4-clock mult-add.
GCM 6 [38]	1-clock AES-P × 4	4-block mult-add.[a]

We use 4 different GF multiplier-adders: (a) parallel multiplier-adder (shown in Fig. 2(b)), (b) 8-clock sequential multiplier-adder [38], (c) 4-clock sequential multiplier-adder [38], (d) block multiplier-adder [38]. A parallel multiplier-adder is shown in Fig. 2(b). It multiplies the 128-bit input A with the 128-bit constant H and sums up the 128-bit result in the register X to calculate the intermediate hash value X_i in one clock cycle. The parallel multiplier-adder, the 8-clock sequential multiplier-adder, and the 4-clock sequential multiplier-adder have a throughput of 1, 8, and 4 cycles, respectively. The block multiplier-adder can process 4 128-bit data blocks simultaneously. Let $i = m + n + 1$, in which m is the number of authentication data blocks and n is the number of encryption data blocks. The throughput of the block multiplier-adder is 2 if $i = 1$ and is $\lceil \frac{i-2}{4} \rceil + 3$ if $i > 1$.

3 GCM and CCM architectures

Table 2 presents GCM architectures. Various AES and GHASH modules are used in each architecture for different throughput and area requirements. GCM architecture 1 uses one iterative AES-I module, one pipelined AES-P module, and GHASH with 1-clock parallel GF multiplier-adder. GCM architecture 2 uses 1 fully pipeli- ned AES-P and GHASH with 1-clock GF multiplier-adder. GCM architecture 3 uses 1 iterative AES-I and GHASH with 8-clock sequential GF multiplier-adder. GCM architecture 4 uses 1 pipelined-loop AES and GH- ASH with 4-clock sequential GF multiplier-adder. GCM architecture 5 uses 4 iterative AES-I and GHASH with 4-clock sequential multiplier-adder. GCM architecture 6 uses 4 fully pipelined AES-P and GHASH with one 4-block multiplier-adder.

3.1 CED Architectures for GCM

Although one can use separate CEDs for AES and GH- ASH, a straightforward combination of CED-enabled AES and CED-enabled GHASH may not yield optimal throughput and area. Moreover, because it is desirable that GCM remains free of intellectual property restrictions, one needs to be able to use AES and GHASH from different IP vendors interchangeably to build GCM architectures without compromising reliability.

3.2 CED for GCM Architecture 1

3.2.1 Baseline Architecture Fig. 3 shows a high-performance GCM architecture. This architecture uses a fully pipelined AES (AES-P), an iterative AES (AES-I), and GHASH with a parallel multi- plier-adder. The data input register (DI) is a 128-bit register. Assuming m=4 and n=4, we analyze the data dependency in Fig. 4(a). The numbers on the left or right side of the bidirectional arrows indicate the time in clock cycles. The normal computation

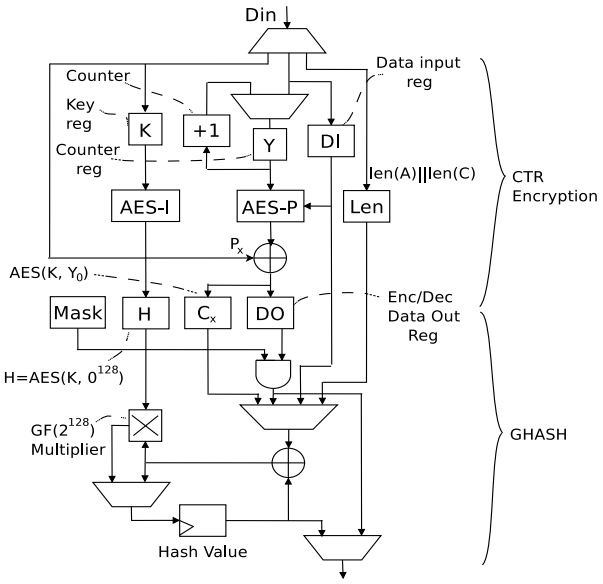


Fig. 3: GCM architecture 1 with 10-clock AES-I, 1-clock AES-P and 1-clock GHASH.

results are indicated by the the oval (white), e.g., from cycle 1 to 10, AES-I takes K as the input at cycle 1 and generates H at cycle 10. The CED checks are indicated by the rectangle (green) which we will discuss in Section 3.2.2. Generally, a single secret key is used for all packets processed in a given secure session. This secret key is determined during session initiation. Hence, K and H are ready before packets are transmitted. GCM starts computing the intermediate hash value $X_1{-}X_4$ when it receives a packet header as additional authenticated data $A_1{-}A_4$. GHASH takes 4 cycles to generate X_4, and waits until ciphertext C_1 is generated. Then, Y_0 is generated by padding 32 zeros to IV, and this takes 0 cycles. After Y_0 is generated, GCM encryption starts to compute the key stream $(C_0, C_1, ...C_4)$. Because the data input register is 128-bit and packet data are coming as 128-bit blocks, the authentication data blocks are processed before the plaintexts blocks, so GCM encryption cannot start earlier. Once AES-P has ramped up, the ciphertext C_i $(1 \leq i \leq 4)$ is generated every cycle. When C_1 is generated, GHASH restarts computing until the end of the GCM operation.

3.2.2 Idle Cycle-based CED Architecture We analyze the CED checks in Fig. 4(a). Our idea is to use the idle hardware during the GCM operation to perform CED checks. AES-I is idle after computing H. Then it can use 10 cycles to check one encryption. The CED check starts at cycle 11. GHASH uses an extra 4 cycles to recompute X_4 when AES-P fills its pipeline. AES-P has 2 idle cycles before GHASH generates the final hash value. So it uses 2 cycles to check 2 extra encryptions; e.g., C_1 and C_2.

In Figs. 5(a) and 5(b), we analyze the number of CED checks for AES and GHASH when we vary m and n values. In both figures, we analyze for $m = 1, 4, 8, 10, 12, 16$ and $n = 4, 8, 12, 16$. The results show that the GCM can check 3 to 6 encryptions and 1 to 10 hashes. We formulate the number of encryption and hash checks for various m and n values in Table 5.

We define ending time of computation to the ending time of recomputation as the detection latency. Because AES-I is only used to compute H once, in AES-I CED from cycle 10 to 20, the detection latency is 10 cycles. AES-P first computes C_0. In the AES-P CED round, the detection latency is 1 to 5 cycles. For GHASH, the detection latency is 4 cycles,

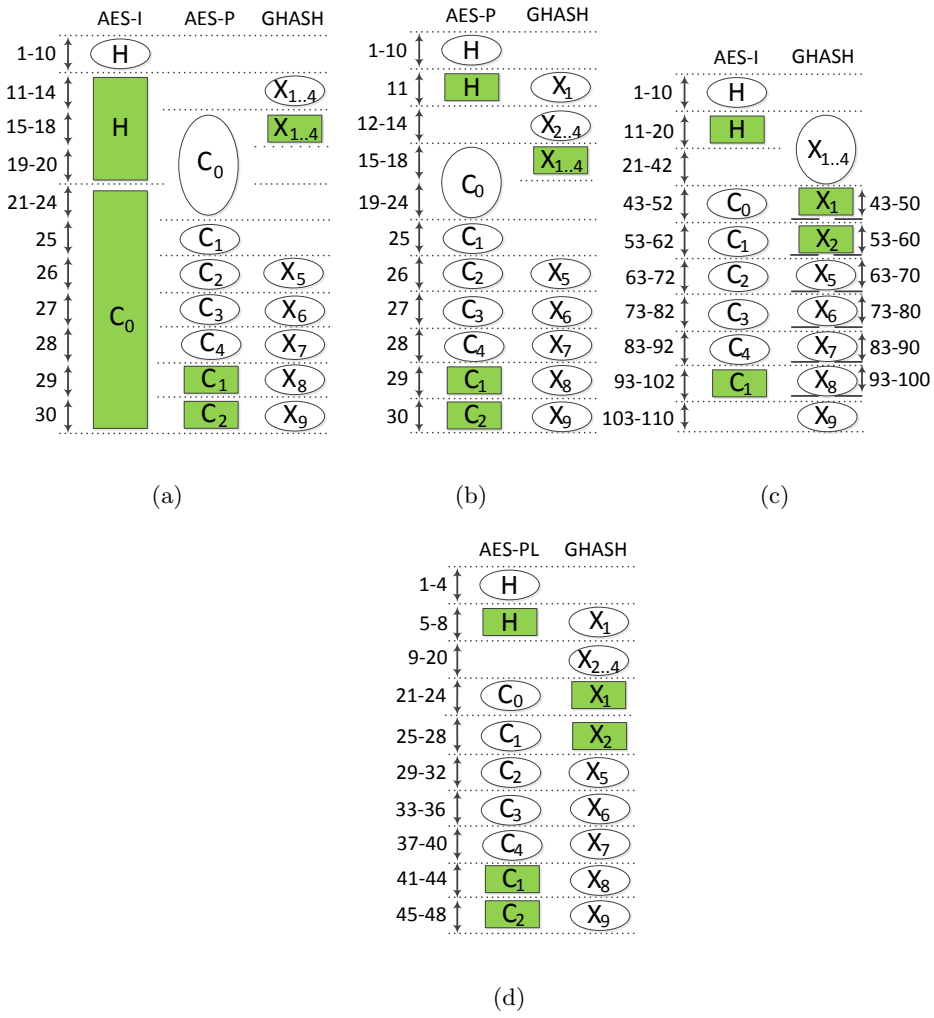


Fig. 4: The data dependencies and CED checks (m=4, n=4) of GCM architecture (a) 1, (b) 2, (c) 3, (d) 4.

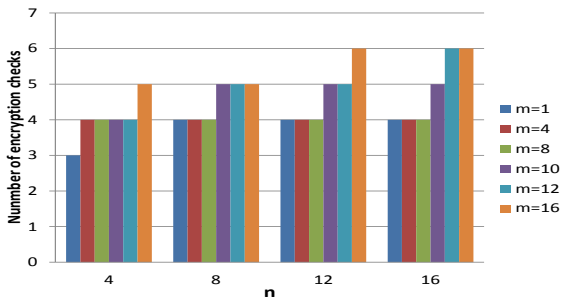
because the CED is used immediately after the normal computation. The detection latency of different m and n values is shown in Table 3.

3.3 CED for GCM Architecture 2 to 4

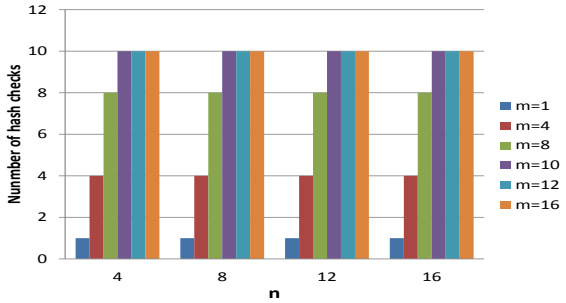
GCM architecture 2 is similar to GCM architecture 1, with the exception that there is only one AES-P module in the GCM [38]. The data dependency and the CED checks are shown in Fig. 4(b). The detection latency for AES is between 1 and 6 cycles. The detection latency for GHASH is between 1 and 4 cycles.

GCM architecture 3 uses an iterative AES and a 8-clock sequential multiplier-adder. The data dependency and CED scheduling are shown in Fig. 4(c). The detection latency of AES is between 10 to 60 cycles. GHASH recomputes the one authentication data block right after the normal computation. Therefore, the detection latency is between 8 to 32 cycles.

GCM architecture 4 uses a pipelined-loop AES and a 4-clock sequential multiplier-adder. The data dependency along with the CED checks are shown in Fig. 4(d). The detection



(a)



(b)

Fig. 5: (a) Number of encryption checks and (b) number of hash checks as a function of m, n in CED for GCM architecture 1.

latency of AES is between 4 to 24 cycles. The detection latency of GHASH is between 4 and 24 cycles.

3.4 CED for GCM Architecture 5

3.4.1 Baseline Architecture This architecture uses 4 10-clock AES-I, and a 4-clock GHASH with a sequential GF multiplier-adder. The hardware is shown in Fig. 6(a). The data dependency are shown in Fig. 6(b). The GCM first uses an AES-I to compute H with the user key K in 10 cycles. After Y_0 is generated, Y_1–Y_4 is generated by the counter (CTR) in cycle 27 simultaneously. Then, it takes 10 cycles for the 4 AES modules to generate X_1–X_4. After that, 4 AES ciphertexts is generated every 10 cycles, and every intermediate hash value X_i is generated every 4 cycles. To generate all ciphertexts, it takes 20 cycles. The first four intermediate hash values are generated in 16 cycles. Then it wait for the ciphertext to be generated, Then it takes 20 cycles to generate the final hash value from cycle 37 to 56.

3.4.2 Idle Cycle-based CED Architecture The CED checks are shown in Fig. 6(b). Because there are 3 idle AES modules when H is being computed, we use 1 idle AES to compute H and check the results. When C_0–C_3 are computed, 2 hash values can be checked in 8 cycles, After C_0–C_4 are generated, there are still 20 cycles before the GCM operation finishes. So we use 20 cycles to check all 5 encryption results C_0–C_4. From cycle 37 to 46, C_0, C_1, and C_2 are recomputed. From cycle 47 to 56, C_3 and C_4 are computed. Therefore, we use different AES modules to recompute the respective data to detect permanent faults. The detection latency is 20 cycles. Because GHASH needs to wait for the counter to generate input for AES, the detection latency is 5 cycles.

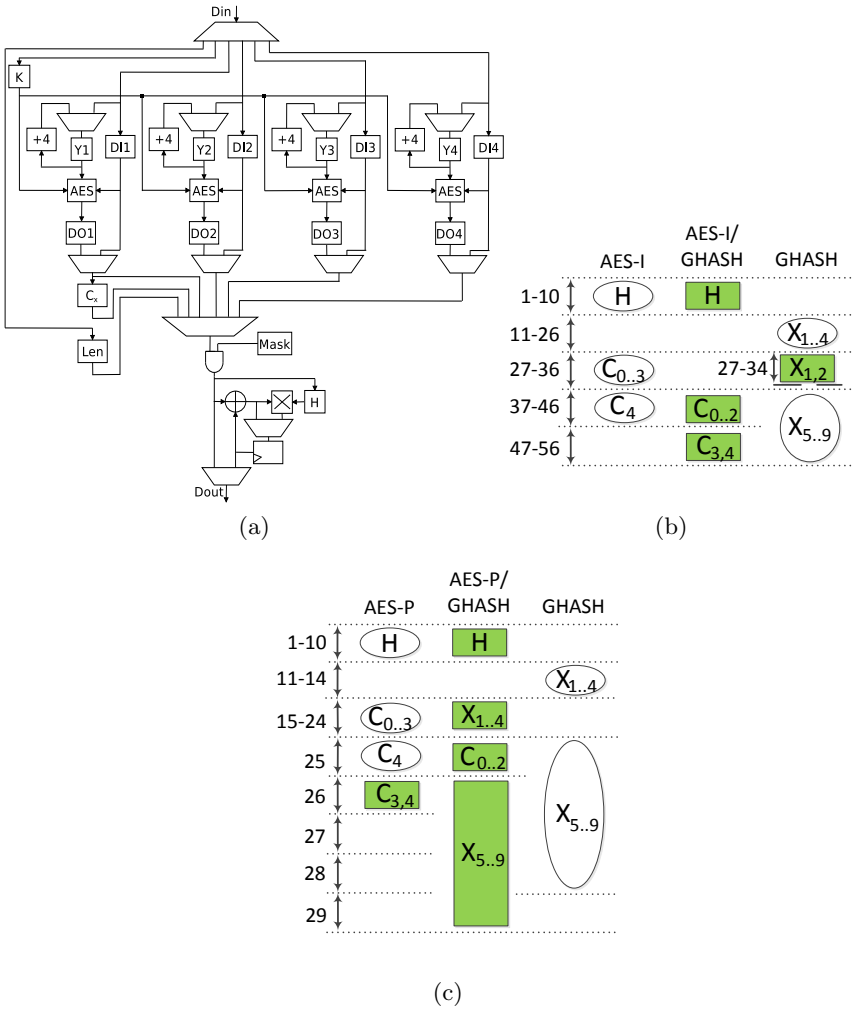


Fig. 6: (a) GCM architecture 5. The data dependencies and CED checks (m=4, n=4) of GCM architecture (b) 5, (c) 6.

Thus, the miminum detection latency of AES is 0 cycles. When AES computes the ciphertexts, it needs to use 4 AES to compute 4 ciphertexts, then one AES will be used to compute the last ciphertext. When this AES computes the last ciphertext, three other AES can be used to recompute 3 ciphertexts. In the end, 2 AES will finish computing the last 2 ciphertexts. Thus, the detection latency is at most 20 cycles. The GHASH recomputes the one authentication data blocks right after the normal computation. Therefore, the detection latency is between 4 and 24 cycles.

3.5 CED for GCM Architecture 6

3.5.1 Baseline Architecture This architecture is similar to GCM architecture 5. It uses 4 pipelined AES-P and a 4-block multiplier-adder to achieve higher throughput. The data dependencies are shown in Fig. 6(c).

3.5.2 Idle Cycle-based CED Architecture Because there are 4 AES-P modules and 4 parallel multi- plier-adders, it can detect permanent faults if we let different modules to

Table 3: Detection latency in cycles with number of authentication blocks m and number of encryption blocks n.

Arch.	Encryption		Hash
GCM 1	AES-I	AES-P	$1 \sim m$
	10	$1 \sim (n+2)$	
GCM 2	$1 \sim (n+2)$		$1 \sim m$
GCM 3	$10 \times (1 \sim (n+2))$		$8 \sim 8 \times m$
GCM 4	$4 \times (1 \sim (n+2))$		$4 \sim 4 \times m$
GCM 5	$0 \sim \lceil 10 \times (n+1)/4 \rceil$		$4 \times (4 \sim m) + 1$
GCM 6	$0 \sim \lceil (n+1)/4 \rceil$		$0 \sim m$

check each other. When the one AES-P is computing H, one of other AES-P is used to compute H. The detection latency is 0 cycles. After AES-P computes the first 4 ciphertexts in 10 cycles, an AES-P computes the last ciphertext and the other AES-P can be used to recompute three ciphertexts in the next cycle. Two other ciphertexts will be recomputed in another cycle. So the detection latency is 2 cycles. When one GHASH computes H^2 and H^4, one of the other GHASH can do the same computation. Thus, the minimum detection latency is 0 cycles. When GHASH recompute the authentication data, it needs to wait for the counter to generate input for the AES-P. So the detection latency is 2 cycles. Every intermediate hash value X_i $(0 \leq i \leq 4)$ is generated every cycle.

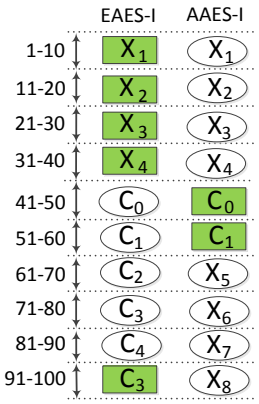
3.6 Application of the Proposed CED to CCM Mode

Counter with CBC-MAC (CCM) is the other confidentiality and authentication mode [42]. CCM with AES is adopted in IEEE 802.15.4 standard to provide link-layer security in wireless network [20].

3.6.1 Baseline Architecture Similar to GCM, CCM also uses AES counter mode for the encryption (EAES-I). CCM also uses AES cipher block chaining (CBC) mode for authentication (AAES-I). In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. Therefore, each ciphertext block is dependent on all plaintext blocks processed up to that point. We implemented CCM with two iterative AES-I as described in [28] for 4 authentication blocks and 4 plaintexts. The data dependencies are shown in Fig. 7(a).

The authentication AES-I first computes the hash from authentication data A_1−A_4 in 40 cycles, and it waits for C_1 to be generated. The encryption AES-I encrypts the plaintexts C_0 −C_4 in 50 cycles. After C_1 is generated, the authentication AES-I starts to compute hashes from the ciphertexts. Thus, it takes 50 cycles to compute the complete hash value.

3.6.2 Idle Cycle-based CED for CCM Architecture We analyze the data dependency in Fig. 7(a). The encryption AES-I is idle at the beginning of the operation, thus, we use it to recompute the hash value X_1−X_4 and these values are checked every cycle. Once the encryption AES-I starts to encrypt plaintexts, the authentication AES-I waits for C_1 to be generated. Therefore, we use it to check C_0 and C_1 simultaneously with encryption AES-I. When the authentication AES-I computes the last hash value X_8, the encryption AES-I is idle and is used to check another encryption C_3. The ASIC implementation result shows that the CED utilizes 60,694 gates which is 3.5% hardware overhead. The CED has 23.2% power overhead. For the FPGA implementation, the CED utilizes 1,017 slices which is 1.23% hardware overhead. The power overhead is 1.4%.



(a)

Fig. 7: Data dependency and CED checks of CCM architecture

4 Experimental Results

In this section, we discuss two improvements for the proposed CED. We also analyze the fault coverage and implementation results.

4.1 Fault Coverage

We analyze the fault coverage of the proposed CED for transient faults and permanent faults as shown in Table 4. The proposed CED can detect transient faults in all the GCM architectures. GCM 1, 5, and 6 are able to detect permanent faults in the encryption module because it has more than one encryption engines and they can check each other. GCM 6 can also detect permanent faults in the authentication (GHASH) module because it has 4 multipliers in the block multiplier and adder architecture. CCM can detect permanent faults in the authentication (AES) module

Recomputing with Permuted Operands (REPO) uses a permutation to recompute the inputs, and its fault coverage is around 99.99997% for both transient and permanent fault [17]. It also provides provable security against fault attacks [48]. Therefore, we can add additional registers to the AES and use them to store the round input and output of the normal computation. During the CED recomputation, we will apply the permuted input to the AES and compare the permuted output with those previously stored. By adding REPO, the fault coverage of the proposed scheme can detect permanent fault and the fault coverage is the same as transient fault. The finite field multiplier can also be protected using normal basis REPO [47].

Since the proposed CED utilizes the idle cycles in the algorithm, the fault coverage for transient faults with various durations are shown in Table 4. We vary the number of fault duration D from 1 to $4 \times L$. L is the latency of the module, e.g., the latency of AES-I is 10 cycles. Thus, D_1 represents the faults the last from 1 to $L = 10$ cycles in the GCM operations. Based on our analysis, the faults coverages stays the same when faults last longer than $4L$ cycles, because there are no more computations to affect in the architecture. The fault coverage of AES increases as the fault duration increases for all architectures. The only exception is the AES-I in architecture 1. It has 0% fault coverage if the fault duration reached $2L$. Because there are only two computations in AES-I in the first architecture, if the faults persist during these entire two computations, the two computation generates the

Table 4: Comparisons of fault detection capability against transient and permanent faults. † Encryption module (AES). ◇ Authentication module (GHASH for GCM and AES for CCM). ⋆ The proposed CED can detect permanent faults if REPO is integrated.

Arch.	Transient		Permanent	
	Enc†	Auth◇	Enc†	Auth◇
GCM 1	√	√	√	
GCM 2	√	√	√⋆	
GCM 3	√	√	√⋆	
GCM 4	√	√	√⋆	
GCM 5	√	√	√	
GCM 6	√	√	√	√
CCM	√	√	√	√

Table 5: Number of encryption and hash checks for different m and n values

Arch.	Encryption	Hash
GCM 1	AES-I $\lfloor (m+n+12)/10 \rfloor$ AES-P 2	$\begin{cases} m \text{ if } m < 11 \\ 11 \text{ if } m \geq 11 \end{cases}$
GCM 2	3	$\begin{cases} m \text{ if } m < 11 \\ 11 \text{ if } m \geq 11 \end{cases}$
GCM 3	$\begin{cases} 1 \text{ if } m = 1 \\ 2 \text{ if } m \geq 2 \end{cases}$	$\begin{cases} 1 \text{ if } m = 1 \\ 2 \text{ if } m \geq 2 \end{cases}$
GCM 4	3	$\begin{cases} m \text{ if } m < 3 \\ 3 \text{ if } m \geq 3 \end{cases}$
GCM 5	$\lfloor \frac{8(n+1)}{5} \rfloor - (n-3)$ $-(n-3)\%4$	$\begin{cases} m \text{ if } m < 3 \\ 2 \text{ if } m \geq 3 \end{cases}$
GCM 6	$n+2$	$\begin{cases} m+2 \text{ if } m < 31 \\ 32 \text{ if } m \geq 30 \end{cases}$

same faulty results. For most architectures, the fault coverages of the GHASH increase from 22.2% to 42.9% and then drop back to 33.3%. Because we are only checking the GHASH computation before the ciphertexts starts to generate, the fault coverage will not increase to 50%.

Redundant cycle can be used to compute extra hashes, and it improves the fault coverage of both the AES and GHASH modules. Although this will reduce the throughput, the fault coverage increases significantly. Fig. 8 shows the relationships between number of redundant cycles (x-axis), fault coverage (y-axis), and performance overhead (y-axis) of AES and GHASH. P. O. stands for performance overhead. Even when we increase the fault coverage to 100% by adding redundant cycles, the performance overhead is not 100%. As shown in Fig. 8(a), if we add five additional cycles for GCM 1, the throughput overhead will be 16.7% and the fault coverage will increase to 100% because the GCM can recompute all the encryptions and hashes. The relationship between fault coverage and performance of GCM 2 is similar to GCM 1 as illustrated in Fig. 8(b). In Fig. 8(c), GCM 3 needs 56 redundant cycles to reach 100% fault coverage. Although the number of redundancy cycles required is much larger than GCM 1 and GCM 2, the performance overhead is only 54.5%, much less than 100%. We can observe similar results for GCM 4 and GCM 5 in Fig. 8(d)

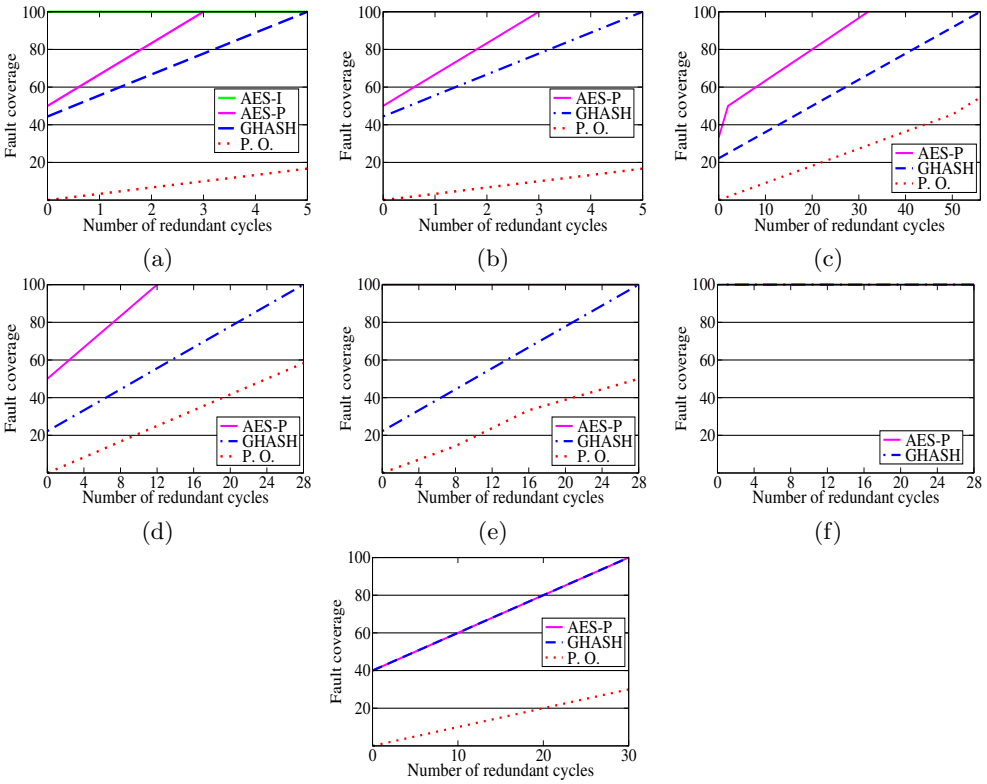


Fig. 8: The relationships between number of redundant cycles (x-axis), fault coverage (y-axis), and performance overhead (y-axis) of AES and GHASH. (a) GCM 1, (b) GCM 2, (c) GCM 3, (d) GCM 4, (e) GCM 5, (f) GCM 6, (g) CCM. P.O. means performance overhead.

Table 6: Fault coverage with four different fault duration in cycles: D_1 ($1 \leq D \leq L$), D_2 ($L<D\leq 2L$), D_3 ($2L<D\leq 3L$), and D_4 ($3L<D\leq 4L$).

Fault duration		D_1	D_2	D_3	D_4
GCM 1	AES-P	40%	100%	100%	100%
	AES-I	40%	0%	0%	0%
	GHASH	44.4%	42.9%	40%	33.3%
GCM 2	AES-P	40%	100%	100%	100%
	GHASH	22.2%	42.9%	40%	33.3%
GCM 3	AES-I	20%	50%	100%	100%
	GHASH	22.2%	42.9%	40%	33.3%
GCM 4	AES-PL	40%	100%	100%	100%
	GHASH	33.3%	42.9%	40%	33.3%
GCM 5	AES-I	100%	100%	100%	100%
	GHASH	22.2%	42.9%	40%	33.3%
GCM 6	AES-PL	100%	100%	100%	100%
	GHASH	50%	100%	100%	100%

Table 7: Number of redundant cycles required to achieve 100% fault coverage when $n \geq 4$.

Arch.	AES	GHASH
GCM 1	0(AES-I) $n-2-\lfloor\frac{n-12}{10}\rfloor$(AES-P)	$n+1$
GCM 2	$n-1$	$n+1$
GCM 3	$10n-8$	$8(n+3)$
GCM 4	$4n-1$	$4(n+3)$
GCM 5	0	$4(n+3)$
GCM 6	0	$6+\lceil(n-1)/2\rceil$
CCM	$10(n-1)$	$10(n-1)$

and 8(d), respectively. For GCM 6, there is no redundant cycles required to achieve 100% fault coverage as demonstrated in Fig. 8(f).

In reality, the size of the packet varies in different applications. Therefore, it is important to provide high performance for a large variety of packet sizes. Moreover, the authentication data in the packets are normally header information, and they are not likely to vary. The data which requires encryption vary significantly. In Fig. 9, we analyze the performance overhead as the number of encryption bytes increases while maintaining 100% fault coverage. We vary the number of encryption bytes from 0 to 8 kilo bytes (kB) while fixing the authentication blocks to four (1kB). Most AE modes has less than 50% performance overhead while the number of bytes are below 1kB. GCM 1 and GCM2 have less than 60% performance overhead when the number of encryption bytes reaches 5kB. Most AE modes reach more than 80% performance overhead after encrypting 8kB encryption bytes except GCM 1, 2, and 3. We can conclude that the performance overhead of GCM 1, 2, and 3 scales better than the other AE architectures.

The center of applied internet data analysis provides recent internet traffic data from various cities such as Chicago and San Jose [46]. For example, in the Chicago area, the mean packet size for IPv4 and IPv6 ranges from 406B to 1.11KB between 2008 and 2014. In San Jose, it ranges from 359B to 942B between 2008 and 2014. Data from other areas show similar results. Since the packet size is below 1KB most of the time, the performance overhead are in the range of 25% to 65% depends on the AE architectures.

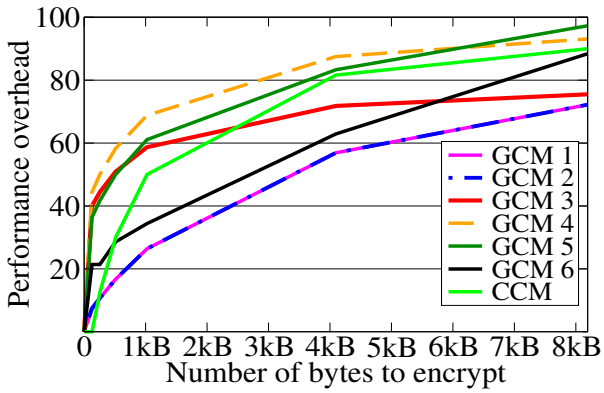


Fig. 9: The performance overhead as the number of encryption bytes increases while maintaining 100% fault coverage.

Table 8: Comparisons of implementation of CEDs on FreePDK 45nm ASIC library. ⋄ overhead in percentage.

Arch.	Original					With CED				
	Gate	Freq. (GHz)	Thro. (Gbps)	Eff.(Gbps /gate)	Pwr (mW)	Gate (%)⋄	Freq. (GHz)	Thro. (Gbps)	Eff.(Gbps /gate)	Pwr (mW) (%)⋄
GCM 1	267,099	1.0	128	0.48	23.67	271,210(2.7)	1.0	128	0.47	28.07(18.6)
GCM 2	264,071	1.0	128	0.48	21.80	274,996(3.0)	1.0	128	0.47	25.63(17.6)
GCM 3	41,145	1.36	17.4	0.42	5.20	50,748(23.3)	1.36	17.4	0.34	6.26(20.4)
GCM 4	125,785	0.37	11.84	0.09	5.14	131,407(4.5)	0.37	11.84	0.09	5.67(10.2)
GCM 5	128,643	1.35	69.1	0.54	15.19	138706(7.8)	1.35	69.1	0.50	16.48(8.5)
GCM 6	1,007,471	0.94	481.3	0.48	139.91	1,020,311(1.3)	0.94	481.3	0.42	148.33(0.6)
CCM	58,614	1.36	17.4	0.30	6.12	60,694(3.5)	1.36	17.4	0.29	7.54(23.2)

4.2 Implementation Results

We implement CCM, and six GCM architectures on ASIC and FPGA, shown in Table 8 and 9, respectively. We implemented these authenticated encryption modes in 45nm FreePDK ASIC library as well as Xilinx Virtex-5 (xc5vlx330t-2ff1738) platform We use pipelined distributed memories for S-boxes and inverse S-boxes similar to [16]. For both ASIC and FPGA, we use ModelSim SE 6.5c [31] to generate the VCD file so that we can evaluate the dynamic power consumption more accurately. For ASIC, we use Cadence RTL Compiler (RC10.1.306) for synthesis and power evaluation [13]. The metrics include (1) the area (the number of gate instances), (2) are overhead (ratio of number of gate instances for the ones with CED over the ones without CED), (3) maximum clock frequency, (4) throughput, (5) efficiency (raitio of (4) over (1)), (6) static power consumption, (7) dynamic power consumption, and (8) dynamic power consumption overhead. For FPGA, we use Xilinx ISE 10.1 for synthesis and Xilinx XPower for power evaluation [44]. the metrics include (1) slice utilization (the number of occupied slices), (2) slice overhead (ratio of number of slices for the ones with CED over the ones without CED), (3) maximum clock frequency, (4) throughput, (5) efficiency (raitio of (4) over (1)), (6) static power consumption, (7) dynamic power consumption, and (8) dynamic power consumption overhead.

In general, the CED implementations have slightly higher hardware overhead than the ones without CED on both ASIC and FPGA. The power consumption of the CED is slightly larger than the one without the CED. First, let us take a look at the ASIC implementation results. Our scheme only adds comparators at the output of AES and GHASH modules as

Table 9: Comparisons of implementation of CEDs on Xilinx Virtex-5 FPGA (xc5vlx330t).

Arch.	Original					With CED				
	Slice	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)	Pwr (W)	Slice (overhead)	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)	Pwr (W) (overhead)
GCM 1	5,645	100.3	12.9	2.29	4.19	6,060(7.4%)	100.3	12.9	2.13	4.46(6.4%)
GCM 2	5,469	100.2	12.9	2.36	3.65	6,191(13.2%)	100.2	12.9	2.08	3.88(6.3%)
GCM 3	926	98.6	1.26	1.36	3.52	1,137(22.8%)	98.6	1.26	1.10	3.57(1.4%)
GCM 4	2,956	50	1.6	0.54	3.97	3,091(0.1%)	50	1.6	0.52	4.39(10.6%)
GCM 5	1,955	111.1	5.7	2.92	3.75	2,353(20.3%)	50	5.7	2.92	4.1(9.3%)
GCM 6	19,951	50.5	25.7	1.29	3.97	24,252(21.6%)	50.5	25.7	1.29	3.98(0.3%)
CCM	830	160.3	2.05	2.45	3.64	1,017(1.23%)	160.3	2.05	2.02	3.69(1.4%)

well as redundant flip-flops. The comparator is not in the critical path of the design, thus, it does not change the clock frequency of the designs. It is note that the results are synthesis results and we may expect the CED implementation have slightly higher delay because the extra flip-flops that stores the intermediate value may increase the fanout. Because our CED checks are only performed when the modules are idle, they do not use extra cycles to affect the throughput of the designs. If we look at the gate utilization, GCM 1 has slightly higher slice utilization than GCM 2 because GCM 1 has an extra iterative AES. GCM3 is the smallest among all the GCM architectures and it also has the lowest throughput. The CED power overhead of GCM 1, 2, and 3 are in the 17.6-20.4% range. GCM 4 only has 10.2% power overhead because it has larger hardware which consumes more static power for both the ones with CED and without CED. Similarly, the GCM 6, which is the largest among all GCM architectures, has only 0.6% power overhead. We can observe similar results on the FPGA. The static power of FPGA is much larger than the static power of ASIC because of extra the switches and interconnects on the FPGA. Therefore, the power overhead on the FPGA is smaller.

5 Conclusion

This paper presented an idle cycle-based CED scheme that explores the data dependency of AE modes. It is applicable to various GCM and CCM architectures. Moreover, this technique does not assume the CED capability of the underlying cryptographic primitives. Therefore, it allows the designers to integrate IPs from different vendors. The fault coverage of the proposed CED scheme is between 20% to 100% for AES module and is between 22.2% to 44.4% for GHASH for transient faults. The fault coverage can increase to 100% without adding 100% performance overhead. With those improvements, the fault coverage can increase to 100%. Finally, we extend the proposed CED to CCM. Our experimental results show that the fault coverage is 100% for both transient fault and permanent fault with 7.2% hardware overhead. We also analyze the power consumption as well as the area utilization. The proposed technique shows that a system-level trade-off needs to be made when designing CED for complex cryptographic designs.

6 Acknowledgments

This material is based upon work supported by the NSF CNS program under grant 0831349 and the Center for Interdisciplinary Studies in Security and Privacy (CRISSP). The authors would like to thank Rui Yang for his help with the implementations. The final publication is available at Springer via http://dx.doi.org/10.1007/s10836-014-5494-0

References

1. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness. http://competitions.cr.yp.to/caesar.html
2. Information technology-security techniques-authenticated encryption. 19772 2009. ISO/IEC. Retrieved March 12 (2013)
3. Agarwal, M., B. C. Paul, M.Z., Mitra, S.: Circuit Failure Prediction and Its Application to Transistor Aging. VTS pp. 277–286 (2007)
4. Albrecht, M., Paterson, K., Watson, G.: Plaintext Recovery Attacks against SSH. IEEE Security and Privacy pp. 16–26 (2009)
5. Bar-El, H.: Intra-Vehicle Information Security Framework. In Proc. the ESCAR Conference (2009)
6. Bardou, R., Focardi, R., Kawamoto, Y., Simionato, L., Steel, G., Tsay, J.K.: Efficient Padding Oracle Attacks on Cryptographic Hardware. CRYPTO pp. 608–625 (2012)
7. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE **100**(11), 3056–3076 (2012)
8. Bayat-Sarmadi, S., Hasan, M.A.: On Concurrent Detection of Errors in Polynomial Basis Multiplication. IEEE Trans. VLSI **15**(4), 413–426 (2007)
9. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication pp. 1–15 (1996)
10. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. IEEE Trans. Computers **52**(4), 492–505 (2003)
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. http://eprint.iacr.org/2011/499.pdf
12. Borkar, S.: Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation. MICRO **25**(6), 10–16 (2005)
13. Cadence: Encounter rtl compiler. http://www. cadence.com/products/ld/rtl_compiler/pages/default.aspx
14. Canvel, B., Hiltgen, A., Vaudenay, S., Vuagnoux, M.: Password Interception in a SSL/TLS Channel. CRYPTO pp. 583–599 (2003)
15. Chih-Hsu, Y., Bing-Fei, W.: Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard. IEEE Trans. Computers **55**(6), 730–731 (2006)
16. Guo, X., Karri, R.: Invariance-based Concurrent Error Detection for Advanced Encryption Standard. DAC pp. 573–578 (2012)
17. Guo, X., Karri, R.: Recomputing with permuted operands: A concurrent error detection approach. IEEE Trans. CAD **32**(10), 1595–1608 (2013)
18. Henzen, L., Fichtner, W.: FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications. ESSCIRC pp. 202–205 (2010)
19. IEEE: 802.1AE-media access control (MAC) security, draft 3.5. http://www.ieee802.org/1/pages/ 802.1ae.html (2005)
20. IEEE: IEEE Std. 802.15.4-2006 (2006)
21. IEEE: P1619.1/d12astandard for authenticated encryption with length expansion for storage devices. http://grouper.ieee.org/groups/ 1619/email/bin00084.bin (2006)
22. Karaklajić, D., Schmidt, J.M., Verbauwhede, I.: Hardware designer's guide to fault attacks. IEEE Trans. VLSI **21**(12), 2295–2306 (2013)
23. Karpovsky, M., Kulikowski, K.J., Taubin, A.: Robust Protection Against Fault-Injection Attacks of Smart Cards Implementing the Advanced Encryption Standard. DNS pp. 93–101 (2004)
24. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes of fault based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans. CAD **21**(12), 1509–1517 (2002)
25. Katz, J., Lindell, Y.: Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC (2007)
26. Kohno, T., Viega, J., Whiting, D.: The CWC Authenticated Encryption (associated data) Mode. http://eprint.iacr.org/ (2003)
27. Lee, C.Y., Chiou, C.W., Lin, J.M.: Concurrent Error Detection in a Polynomial Basis Multiplier over GF(2^m). Journal of Electronic Testing: Theory and Applications **22**(2), 143–150 (2006)

28. López-Trejo, E., Rodríguez-Henríquez, F., Díaz-Pérez, A.: An Efficient FPGA Implementation of CCM Mode using AES (2005)
29. M. Mozaffari-Kermani and A. Reyhani-Masoleh: Concurrent structure-independent fault detection schemes for the Advanced Encryption Standard. IEEE Trans. Computers **59**(5), 608–622 (2010)
30. McGrew, D., Viega, J.: The Galios/Counter Mode of Operation (Full Version)). http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/-gcm/gcm-revised-spec.pdf (2005)
31. Mentor Graphics: Modelsim. http://model.com/
32. Mozaffari-Kermani, M., Reyhani-Masoleh, A.: A lightweight high-performance fault detection scheme for the Advanced Encryption Standard using composite field. IEEE Trans. VLSI **19**(1), 85–91 (2011)
33. Mukherjee, S.S., Emer, J., Reinhardt, S.K.: The Soft Error Problem: An Architectural Perspective. HPCA pp. 243–247 (2005)
34. Natale, G.D., Flottes, M.L., Rouzeyre, B.: A Novel Parity Bit Scheme for SBox in AES Circuits. DDECS pp. 1–5 (2007)
35. National Institute of Standards and Technology (NIST): Recommendation for Block Cipher Modes of Operation - Methods and Techniques. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf (2001)
36. National Institute of Stardards and Technology (NIST): Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (2001)
37. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/-proposedmodes/ocb/ocb-spec.pdf (2001)
38. Satoh, A., Sugawara, T., Aoki, T.: High-Performance Hardware Architectures for Galois Counter Mode. IEEE Trans. Computers **58**(7), 917–930 (2009)
39. Satoh, A., Sugawara, T., Homma, N., Aoki, T.: High-performance concurrent error detection scheme for AES hardware. In Proc. CHES pp. 100–112 (2008)
40. Vaudenay, S.: Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... EUROCRYPT pp. 534–546 (2002)
41. Viega, J., McGrew, D.: The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (EPS). http://www.faqs.org/rfcs/rfc4106.htm (2005)
42. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ccm /ccm.pdf (2002)
43. Wu, K., Karri, R., Kuznetsov, G., Goessel, M.: Low Cost Concurrent Error Detection for the Advanced Encryption Standard. ITC pp. 1242–1248 (2004)
44. Xilinx: http://www.xilinx.com/
45. Yang, B., Mishra, S., Karri, R.: High Speed Architecture for Galois/Counter Mode of Operation (GCM). Cryptography ePrint Archive: Report 2005/146 (2005)
46. http://www.caida.org/data/passive/trace_stats/
47. Guo, X., Mukhopadhyay, D. Jin, C., Karri, R.: NREPO: Normal Basis Recomputing with Permuted Operands Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on pp. 118–123 (2014)
48. Guo, X., Mukhopadhyay, D., Karri, R.: Provably Secure Concurrent Error Detection Against Differential Fault Analysis Cryptology ePrint Archive, Report 2012/552 (2012)