J Cryptogr Eng (2011) 1:201–218
DOI 10.1007/s13389-011-0013-z

REGULAR PAPER

High performance GHASH and impacts of a class of unconventional bases

Nicolas Méloni · Christophe Negre · M. Anwar Hasan

Received: 5 April 2011 / Accepted: 6 July 2011 / Published online: 30 August 2011

Abstract This work presents a new method to compute the GHASH function involved in the Galois/Counter Mode of operation for block ciphers. If $X = X_1 \dots X_n$ is a bit string made of n blocks of 128 bits each, then the GHASH function essentially computes $X_1 H^n + X_2 H^{n-1} + \cdots + X_n H$, where H is the hash key and an element of the binary field $\mathbb{F}_{2^{128}}$. This operation is usually computed using n successive multiply-and-add operations over $\mathbb{F}_{2^{128}}$. Our proposed method replaces all but a fixed number of those multiplications by additions on the field. This is achieved using the characteristic polynomial of H. We present both how to use this polynomial to speed up the GHASH function and how to efficiently compute it for each session that uses a new H. We also show that the proposed technique can be parallelized to compute GHASH even faster. In order to completely eliminate the need for a field multiplication, we investigate a different set of bases for the field element representation and report their architectural and possible security impacts.

Keywords Galois/Counter mode · GHASH function · Characteristic polynomial · Basis representation

1 Introduction

The Galois/Counter mode (GCM) is one of the modes of operation recommended by the National Institute of Standards and Technology (NIST) [12]. As an authenticated encryption mode, it generates both a ciphertext and an authentication tag for each session. The ciphertext is obtained using the counter mode of encryption (CTR) with a 128-bit block cipher algorithm (usually AES). The authentication part is performed using a universal hashing (GHASH) based on multiplication in the binary field $\mathbb{F}_{2^{128}}$ [10]. More precisely, if $X = X_1 \dots X_n$ is a bit stream divided into 128-bit blocks, the authentication process computes $X_1 H^n + X_2 H^{n-1} + \cdots + X_n H$, where H depends on the encryption key.

High speed GCM designs are usually based on both fast implementations of the block cipher [2,4,6,20] and bit-parallel multiplier over $\mathbb{F}_{2^{128}}$ [1,3,8,13]. One can refer to [7,17–19] for various implementations of the AES-GCM mode of encryption.

In practice, the $\mathbb{F}_{2^{128}}$ multiplier of bit parallel type operates faster than the block cipher. However, it is possible to speed up the block cipher computation by taking advantage of the natural parallelism of the CTR mode. In that case, the $\mathbb{F}_{2^{128}}$ multiplier's critical path of the GHASH function becomes the bottleneck. To overcome this problem, one can try to parallelize the GHASH computation process as suggested in [10]. Such solutions imply a proportional increase in the number of multipliers.

This work was done while N. Méloni was at the University of Waterloo, Canada.

N. Méloni
Université de Toulon, Toulon, France
e-mail: meloni.nicolas@gmail.com

C. Negre
ECE Department, University of Waterloo, Waterloo, Canada

C. Negre
LIRMM, Université Montpellier 2, Montpellier, France

C. Negre (✉)
Team DALI, Université de Perpignan, Perpignan, France
e-mail: christophe.negre@univ-perp.fr

M. A. Hasan
Department of Electrical and Computer Engineering,
University of Waterloo, Waterloo, Canada
e-mail: ahasan@sisr.uwaterloo.ca

In this paper, we propose to speed up the GHASH function for long messages. This is achieved by first computing the characteristic polynomial of H and then performing the computation of the GHASH function modulo this polynomial. Effectively, the multiplication in $\mathbb{F}_{2^m}$ is replaced by m parallel field additions. We thus replace the delay of a field multiplication which, in general, is equal to $O(\log(m))$ by a delay of an addition $O(1)$. We also present a parallelized version of the characteristic polynomial approach, which further reduces the GHASH computation time. This was part of a preliminary version of this work presented at the conference ACNS 2010 [11].

We then present a variant of the method based on characteristic polynomial. In this variant, we represent the field elements with a polynomial basis where the generating element of this basis is H or $\sqrt[m]{H}$. This results in a slightly modified GHASH where no $\mathbb{F}_{2^m}$ multiplier is required. Due to a possible security flaw induced by the H dependent representation, we provide a security evaluation for this approach.

The remainder of this paper is organized as follows: Sect. 2 briefly describes the GHASH function and its implementations using multiplication and addition over of the binary field $\mathbb{F}_{2^m}$. In Sect. 3, we describe our approach based on characteristic polynomial for GHASH computation. Later on, we present a parallelization of this approach. In Sect. 4, we study variants of this approach using H-dependent field representation. We compare in Sect. 5, the proposed architectures with best known methods. In Sect. 6, we present how to efficiently compute a characteristic polynomial over $\mathbb{F}_{2^{128}}$. Finally, a few concluding remarks are made in Sect. 7.

2 GHASH authentication function

GCM performs two main operations: encryption and authentication. Encryption is done using a block cipher with a block of size m used in the counter mode. The authentication consists of evaluating the ciphertext, considered as a polynomial in $\mathbb{F}_{2^m}[T]$, in a key dependant element $H \in \mathbb{F}_{2^m}$. In practice the recommended block cipher is AES, which has a block size of $m = 128$ bits. One can find a full description of the encryption process in [10]. Here, we only give some necessary details for the authentication part.

Let $X = X_1 X_2 \dots X_n$ be a bit string divided into block of size m bits (X_n might be padded with 0's) and H be the m-bit hash sub-key. The GHASH function computes a hash value of the message X as

$$\text{GHASH}_H(X) = X_1 H^n + X_2 H^{n-1} + \cdots + X_n H.$$

In practice, X is obtained as the concatenation of the ciphertext, the authentication data and their combined length, and H is generated by applying the block cipher to the block with all bits being zero.

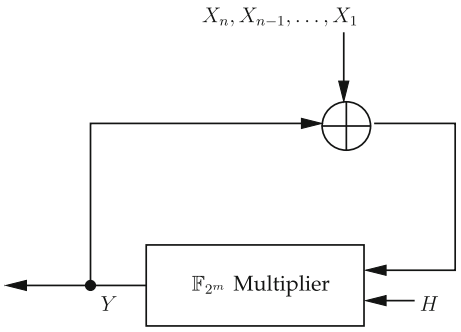

Fig. 1 Single multiply-and-add unit based GHASH architecture

Remark 1 Let us consider a confidential communication between Alice and Bob involving GCM-AES. They first agree on a secret key K and then they use the key during the whole session: this means that they will encrypt several GCM packets with the same key. Consequently, any precomputation involving $H = AES(K, 0)$ can be done once at the beginning of the session and even shared by Alice and Bob and then used as required during the encryption of the different packets. Consequently, in the sequel, if such a precomputation is required in GHASH computations, we do not count it in the complexity evaluation.

2.1 Multiply-and-add architectures for GHASH function

The computation of GHASH(X) can be performed through a sequence of multiplication by H and an addition of X_i in $\mathbb{F}_{2^m}$. This method was proposed by the authors of GCM in [10]. This method is given in Algorithm 1.

Algorithm 1 GHASH using multiply-and-add

Require: X, H
Ensure: $\text{GHASH}_H(X)$
  $Y \leftarrow 0$
  for $i = 1$ **to** n **do**
    $Y \leftarrow (Y + X_i) \times H$
  end for
  return Y

One can verify that Algorithm 1 computes $X_1 H^n + X_2 H^{n-1} + \cdots + X_n H$. Note that Algorithm 1 requires n multiplications and n additions on the field $\mathbb{F}_{2^m}$. Assuming that we have only one multiplier and one adder, these operations would be performed in sequence. The resulting architecture is shown in Fig. 1.

In the GCM specification, m is 128 and the binary field $\mathbb{F}_{2^{128}}$ is defined as $\mathbb{F}_{2^{128}} = \mathbb{F}_2[t]/(t^{128} + t^7 + t^2 + t + 1)$ which is the set of residues of polynomials with coefficients in $\mathbb{F}_2$ modulo $t^{128} + t^7 + t^2 + t + 1$. An element of this field is represented as a vector of length 128 with coefficients in $\mathbb{F}_2$.

Table 1 Complexities of various multipliers over $\mathbb{F}_{2^m}$ where D_X (resp. D_A) is the delay of a two-input XOR (resp. AND) gate

Multiplier	#AND	#XOR	Gate delay (D_M)
CRT [1]	$O(m^{1.6})$	$O(m^{1.6})$	$O(m)D_X + 4m^{0.4}D_A$
Karat. [13]	$O(m^{1.58})$	$O(m^{1.58})$	$(3\log_2(m)+1)D_X + D_A$
F-H [3]	$O(m^{1.58})$	$O(m^{1.58})$	$(2\log_2(m)+1)D_X + D_A$
Mast. [8]	$O(m^2)$	$O(m^2)$	$\log_2(m)D_X + D_A$

The delay of an addition is exactly that of a bitwise XOR operation denoted D_X. In Table 1, we list a number of high-speed bit-parallel field multiplication methods and their space and time complexities in terms of gate counts and gate delays.

Let n be the length of the message to be treated (considered as a sequence of m-bit blocks). The architecture of Fig. 1 performs, at each step of the algorithm, one field multiplication and one block bitwise XOR operation. Denoting D_M as the delay of a field multiplication, which could be any one listed in Table 1, the total delay of a GHASH computation is $\mathcal{D} = n(D_M + D_X)$. The space complexity is equal to $S_M + mS_X$ where S_M denotes the space complexity of the field multiplier and S_X denotes the space complexity of one XOR gate.

2.2 Parallel multiply-and-add architectures for GHASH function

As noted in [10] and later by Satoh [18], the multiply-and-add method for GHASH computation can be parallelized. We assume that we have r field multipliers and r field adders. For simplicity, we also assume that n is a multiple of r (we pad zeros to X if needed). Then we can decompose X into r separate sets $P_1, P_2, \ldots, P_r$ such that

$$\text{GHASH}_H(X) = P_1H^r + P_2H^{r-1} + \cdots + P_rH, \qquad (1)$$

where for $1 \le i \le r$ we have used $P_i = X_i(H^r)^{\frac{n}{r}-1} + X_{i+r}(H^r)^{\frac{n}{r}-2} + \cdots + X_{n-r+i}(H^r)^0$. Then all P_i, for $1 \le i \le r$, can be computed in parallel through $\frac{n}{r} - 1$ iterations, each consisting of multiplication by H^r and an addition. The delay of one multiply-and-add is $D_M + D_X$, thus the time to compute all the P_i is $(\frac{n}{r} - 1)(D_M + D_X)$. With one additional delay of D_M, one can compute P_iH^{r-i+1} for $i = 1, \ldots, r$, assuming that the r elements $H, H^2, \ldots, H^r$ are known. Finally $\text{GHASH}_H(X)$ is obtained by adding the r elements P_iH^{r-i+1}, which incurs a delay of $\lceil \log_2(r) \rceil D_X$, assuming additions are done in a binary tree fashion. The resulting architecture is shown in Fig. 2. The delay of the full computation of $\text{GHASH}_H(X)$ is thus $\mathcal{D} = (\frac{n}{r} - 1)(D_M + D_X) + D_M + (\lceil \log_2 r \rceil)D_X$, and the space complexity is equal to $rS_M + (2r-1)mS_X$.

3 Computing GHASH using a characteristic polynomial

As mentioned earlier in the computation of $\text{GHASH}_H(X)$, H is an element of $\mathbb{F}_{2^m}$. This means that there exists a polynomial $\chi_H(T) = \prod_{i=0}^{m-1}(T + H^{2^i})$ of degree m, which has its coefficients in $\mathbb{F}_2$ and satisfies $\chi_H(H) = 0$. If H does not lie in any of the proper subfields of $\mathbb{F}_{2^m}$, then χ_H is an irreducible polynomial and is known as the minimal polynomial of H. When H lies in a subfield of $\mathbb{F}_{2^m}$, which happens with a small probability of 2^{-64} for the case of GHASH where $m = 128$, polynomial χ_H is a multiple of the minimal polynomial of H. Whether χ_H is irreducible or not, we will refer to it as characteristic polynomial. Below we show how to use such a polynomial to reduce the number of field multiplications involved in the computation of $\text{GHASH}_H(X)$.

3.1 GHASH architecture based on reduction modulo χ_H

Let $H \in \mathbb{F}_{2^m}$, then the characteristic polynomial of H as defined above is a polynomial in $\mathbb{F}_2[T]$ of degree m. From its definition we also have $\chi_H(H) = 0$. Let us now write $\chi_H = \sum_{i=0}^{m} c_iT^i$. Since $\chi_H(H) = 0$ and $c_m = 1$, then we have $H^m = \sum_{i=0}^{m-1} c_iH^i$. Using this expression of H^m, we

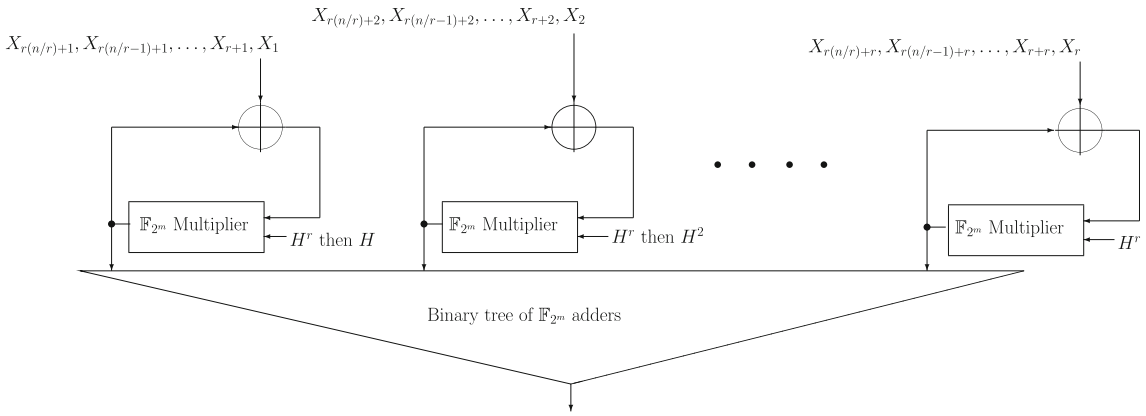

Fig. 2 Parallel multiply-and-add GHASH architecture

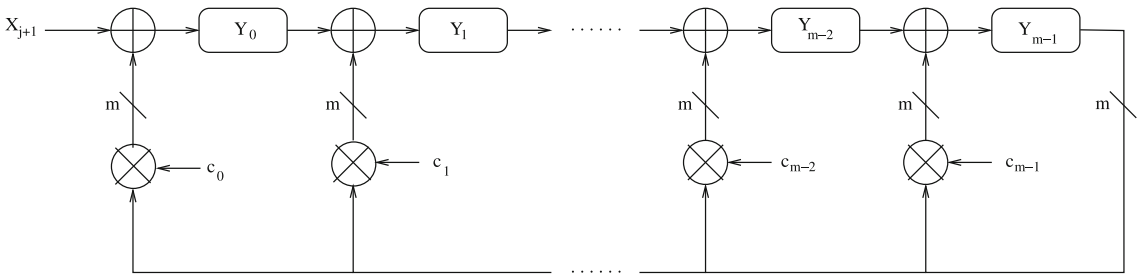

Fig. 3 Implementation of the Polynomial Reduction Unit (PRU)

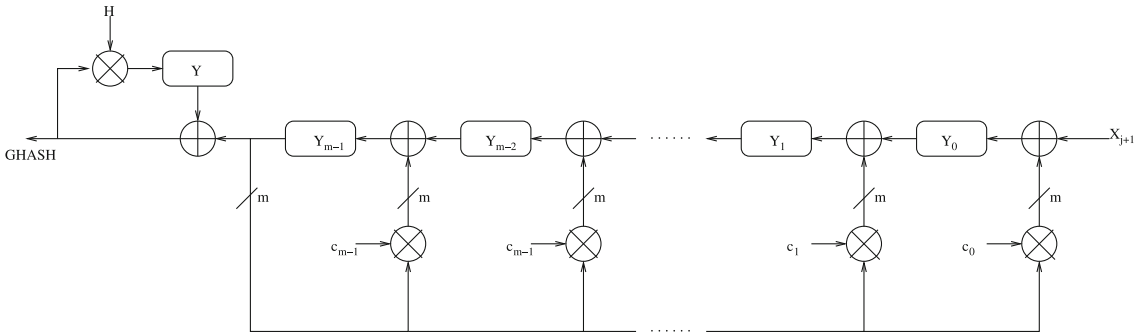

Fig. 4 Implementation of the GHASH function using a PRU

can reduce a polynomial in H of degree m to a polynomial of degree $m-1$. Indeed, let $P = X_1H^m + X_2H^{m-1} + \cdots + X_mH$ with $X_i \in \mathbb{F}_{2^m}$, then

$$P \mod \chi_H = (X_2 + c_{m-1}X_1)H^{m-1} + (X_3 + c_{m-2}X_1)H^{m-2} + \cdots + (X_m + c_1X_1)H + c_0X_1.$$

It is important to note that all the c_is are in $\mathbb{F}_2$, which means that computing $(X_{m-i+1} + c_iX_i)$ is just an addition over $\mathbb{F}_{2^m}$ when $c_i = 1$. This reduction modulo χ_H of a polynomial in H of degree m can be performed in the circuit shown in Fig. 3. We call this circuit a polynomial reduction unit (PRU).

If we consider now a polynomial $P = X_1^n + X_2X^{n-1} + \cdots + X^nH$ in H of degree n, this polynomial can be decomposed as follows

$$P = (\cdots((X_1H^m + X_2X^{m-1} + \cdots + X_{m+1})H + X_{m+2})H + \cdots + X_{n-1})H + X_n)H.$$

We can reduce P modulo χ_H by performing sequentially $n - m$ reductions of polynomial of degree m as follows

$$P = ((\cdots((X_1H^m + X_2X^{m-1} + \cdots + X_{m+1} \mod \chi_H)H + X_{m+2} \mod \chi_H)H + \cdots + X_{n-1} \mod \chi_H)H + X_n \mod \chi_H)H \mod \chi_H.$$

We can perform this reduction using the PRU of Fig. 3. The PRU is originally loaded with $X_1, \ldots, X_m$ and then it is sequentially updated and fed with $X_{m+1}, \ldots, X_n$. The output of the polynomial reduction unit (PRU) is a polynomial of degree $m - 1$ in H. The output of the PRU can then be sent to a multiply-and-add unit in order to compute the desired GHASH value, which is an element of $\mathbb{F}_{2^m}$. This is the approach used in the architecture shown in Fig. 4. Specifically, in this architecture, the PRU is updated $n - m$ times through the feedback connection. Then, during the next m clock cycles, the feedback connections are not active; rather the contents of the PRU registers (i.e., $Y_i, i = m-1, m-2, \ldots, Y_0$) sequentially enter the multiply-and-add unit of the architecture. The switching of the Y_{m-1} content either going to the feedback network of the PRU during the reduction phase or entering the multiply-and-add unit during the final m clock cycles can be realized with a 1:2 demultiplexer. For the sake of simplicity, the demultiplexer is not explicitly shown in Fig. 4.

In Algorithm 2 below, we give the operations performed in the circuit shown in Fig. 4. The evaluation of X of length n m-bit blocks in H can be computed using at most $m-1$ field multiplications, replacing each of the remaining $n-m$ multiplications by a reduction of a polynomial of degree m modulo χ_H. Replacing multiplication by H by reduction modulo χ_H is advantageous with regard the computation delay since the PRU has a smaller delay than a multiplier.

Complexity The PRU part of the architecture in Fig. 4 has a space complexity of $m^2S_X + m^2S_A$, where S_X and S_A denote the space requirement of a XOR gate and an AND

Algorithm 2 GHASH$_H(X)$ using χ_H

Require: $X = X_1 X_2 \dots X_n$ and $\chi_H(T) = \sum_{i=0}^{m} c_i T^i$
Ensure: GHASH$_H(X) = X_1 H^n + X_2 H^{n-1} + \cdots + X_n H$
  $Y_{m-1} \dots Y_0 \leftarrow X_1 \dots X_m$
  $temp \leftarrow 0$
  for $j = m$ **to** $n-1$ **do**
    $C \leftarrow Y_{m-1}$
    $Y_i \leftarrow Y_{i-1} + c_i C, 1 \le i \le m-1$
    $Y_0 \leftarrow X_{j+1} + c_0 C$
  end for
  for $i = m-1$ down **to** 1 **do**
    $temp \leftarrow (temp + Y_i) \times H$
  end for
  return $(temp + Y_0)$

gate, respectively. Its critical path delay is equal to $D_X + D_A$. There are $n - m$ reductions of degree m polynomial which are performed sequentially to obtain the full reduction. Consequently, the total time delay for the reduction modulo χ_H is $(n-m)(D_X + D_A)$. The final $m-1$ field multiplications are performed through a multiply-and-add circuit. This contributes $S_M + mS_X$ to the space complexity and $(m-1)(D_M + D_X)$ to the time complexity. Consequently, the total time delay of the architecture shown in Fig. 4 is equal to

$$\mathcal{D} = (n-m)(D_X + D_A) + (m-1)(D_M + D_X)$$

and the space complexity is equal to

$$\mathcal{S} = (m^2 + m)S_X + m^2 S_A + S_M.$$

For long messages, the value of n is expected to be much longer than that of m, e.g., if the size of X is 1MBytes, then $n \cong 2^{16}$ and the reduction modulo χ_H will dominate in terms of computation.

We recall that the characteristic polynomial χ_H can be precomputed and used during a full GCM session. Consequently, the corresponding complexity can be treated separately in the GHASH computation.

3.2 Delay reduction with multiple PRUs

In order to simplify notations, let $n = 2n'$ be an even integer. Then one can write

$$\begin{aligned} P &= X_1 H^n + X_2 H^{n-1} + \cdots + X_n H \qquad (2)\\ &= (X_1 (H^2)^{n'-1} + X_3 (H^2)^{n'-2} + \cdots + X_{n-1}) H^2 \\ &\quad + (X_2 (H^2)^{n'-1} + X_4 (H^2)^{n'-2} + \cdots + X_n) H \\ &\equiv P_1 H^2 + P_2 H. \qquad (3) \end{aligned}$$

Now assume that we have two PRUs whose feedback connections are defined by the characteristic polynomial of H^2 over $\mathbb{F}_2$. Then these PRUs can process even and odd numbered blocks of X in parallel in $\frac{n}{2} - m$ steps. Referring to (3), the outputs of PRUs are $P_1 \mod \chi_{H^2}$ and $P_2 \mod \chi_{H^2}$, that we call partial GHASH. One can then simply multiply and add according to the equality $P \equiv P_1 H^2 + P_2 H$.

More generally, let us assume that we have r PRUs whose feedback connections are defined by the characteristic polynomial of H^r over $\mathbb{F}_2$. For simplicity, we also assume that n is a multiple of r (we pad zero blocks to X if needed). Then we can decompose X into r different sets $P_1, P_2, \dots, P_r$ such that

$$P = P_1 H^r + P_2 H^{r-1} + \cdots + P_r H. \qquad (4)$$

where for $1 \le i \le r$ we have used $P_i = X_i (H^r)^{\frac{n}{r}-1} + X_{i+r}(H^r)^{\frac{n}{r}-2} + \cdots + X_{n-r+i}(H^r)^0$.

Let $P_i' = P_i \mod \chi_{H^r}$. With r PRUs operating concurrently, we can reduce $P_i \mod \chi_{H^r}$ to obtain the corresponding P_i' for $i = 1, \dots, n$, in $\frac{n}{r} - m$ iterations as shown in the left part of Fig. 5. We now write (4) in terms of P_i' as follows:

$$P \equiv P_1' H^r + P_2' H^{r-1} + \cdots + P_r' H. \qquad (5)$$

Since each P_i' is a degree $m-1$ polynomial in H^r, one can verify that the sum of the products in (5) is nothing but a degree rm polynomial in H (with the constant term being zero). We can then use one more PRU whose feedback connection is defined by the characteristic polynomial of H and

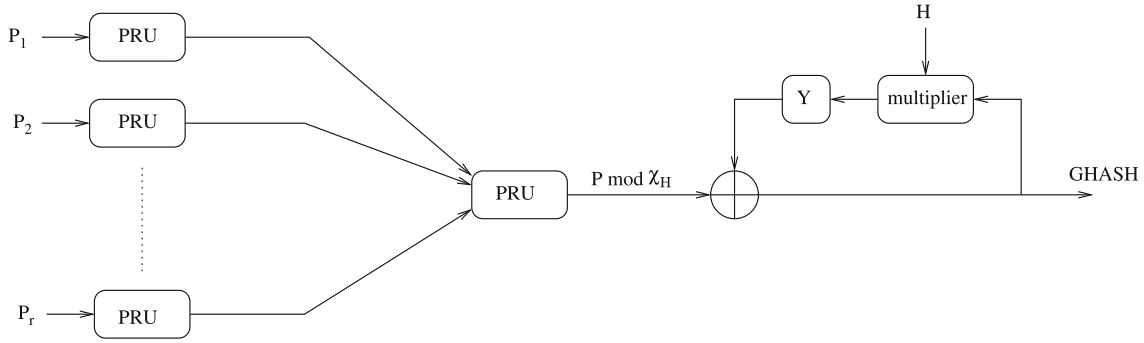

Fig. 5 Implementation of the GHASH function using multiple PRUs

reduce the polynomial in H from degree rm to degree m. Note that computing this characteristic polynomial, unlike that of H^r, can be done in parallel with obtaining P'_i. Moreover, if r is a power of two, then H^r and H share the same characteristic polynomial. We obtain the final result by applying the Horner scheme to the output of the final PRU and this requires $m-1$ multiply-and-add operations.

Hence, given χ_{H^r} and using $r+1$ PRUs and one multiplier-and-add (see Fig. 5), the GHASH computation of X has the following time delay

$$\left(\frac{n}{r}+rm-2m\right)(D_{\mathrm{X}}+D_{\mathrm{A}})+(m-1)(D_{\mathrm{M}}+D_{\mathrm{X}}). \quad (6)$$

The space complexity is equal to $((r+1)m^2+m)S_{\mathrm{X}}$ and $(r+1)m^2 S_{\mathrm{A}}$ and one S_{M}. Note that in order to form the polynomial of (5), an $r:1$ multiplexer can be used. The multiplexer will need to be placed between the array of r first-stage PRUs and the single second-stage PRU of Fig. 5 For the sake of simplicity, the multiplexer is omitted from the diagram in Fig. 5.

4 Towards multiplication-free GHASH

The architectures presented in the previous section can significantly reduce the delay of the GHASH computation. For example, the use of one PRU reduces the number of field multiplications from n to $m-1$, assuming $n \geq m$. For long messages requiring GCM, this is a quite significant improvement. However, for those remaining $m-1$ multiplications, there is still a need for a field multiplier unit, even though the latter can be of sequential type and hence of low cost. To this end, below we attempt to eliminate the need for a field multiplier completely.

In the GCM specification [12], field multiplication has been described using the polynomial basis representation of field elements. The security proof of GCM as given in [10] does not depend on the choice of bases, and conceptually any other suitable bases can be used. Indeed, in order to reduce the cost of field multiplication, other bases, such as *normal*, *dual* and *shifted-polynomial*, have been suggested in the literature, although not specifically for GHASH applications. Below, we attempt to use two bases.

4.1 Using basis generated by H

We consider a message $X = X_1 X_2 \ldots X_n$ decomposed in blocks $X_i = [x_{i,m-1}, \ldots, x_{i,0}]$ of m bits long. In the GCM specification [10,12], the field $\mathbb{F}_{2^m}$ is defined by $\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(f(t))$ for a fixed irreducible polynomial $f(t)$ of degree $m=128$. In particular, [10] and [12] use $f(t) = t^{128}+t^7+t^2+t+1$. The implicit basis used to represent field elements is $\mathcal{B}_t = \{1, t, t^2, \ldots, t^{m-1}\}$. Thus, blocks X_i is a bit string that corresponds to an element in $\mathbb{F}_{2^m}$ whose representation in $\mathcal{B}_t$ is $\sum_{j=0}^{m-1} x_{i,j}t^j$ where $x_{i,j}$ is the jth bit of block X_i. Similarly H is an m-bit block and also seen as a random element expressed in the basis $\mathcal{B}_t$ of $\mathbb{F}_{2^m}$. If we use the notation $X_i^{(\mathcal{B}_t)}$ to indicate that X_i is given in $\mathcal{B}_t$, the expression of GHASH is

$$\mathrm{GHASH}_H(X) = \sum_{i=1}^{n} X_{n-i+1}^{(\mathcal{B}_t)} H^i. \quad (7)$$

Here, we propose to relax the choice of the basis involved in (7). Since H is the "fixed" input to the n multiplications used in (7), it is an obvious first choice, especially in terms of efficient implementation, to perform the multiplication using the following basis

$$\mathcal{B}_H = \{1, H, H^2, \ldots, H^{m-1}\}.$$

Remark 2 In the case of $m=128$, the set $\mathcal{B}_H$ is a basis only when $H \in \mathbb{F}_{2^{128}} \backslash \mathbb{F}_{2^{64}}$ since in this case its minimal polynomial is equal to its characteristic polynomial and is of degree 128. This happens with a high probability of $1-2^{-64}$.

Remark 3 While the polynomial or canonical form of $\mathcal{B}_H$ is public, the exact bit string of H is secret. In other words, the minimal polynomial of H is secret.

The use of basis $\mathcal{B}_H$ makes the multiplication by H involved in Algorithm 1 really efficient. Let $Y^{(\mathcal{B}_H)} = \sum_{i=0}^{m-1} y_i H^i$ be the intermediate value of Algorithm 1 expressed in the basis $\mathcal{B}_H$. We also assume that the characteristic polynomial $\chi_H(T) = T^m + \sum_{i=0}^{m-1} c_i T^i \in \mathbb{F}_2[T]$ of H is known, i.e., the polynomial is computed from H within the GCM unit.. The previous expression of the characteristic polynomial of H induces $H^m = \sum_{i=0}^{m-1} c_i H^i$. We can then perform the multiplication of $Y^{(\mathcal{B}_H)}$ by H during the GHASH computation (Algorithm 1) as follows:

$$\begin{aligned} Y^{(\mathcal{B}_H)} \times H &= y_0 H + y_1 H^2 + \cdots + y_{m-2} H^{m-1} + y_{m-1} H^m \\ &= y_{m-1}c_0 + (y_0 + y_{m-1}c_1)H + (y_1 + y_{m-1}c_2)H^2 \\ &\quad + \cdots + (y_{m-2} + y_{m-1}c_{m-1})H^{m-1}. \end{aligned} \quad (8)$$

The above expression of $Y^{(\mathcal{B}_H)} \times H$ is obtained after the replacement of H^m by $\sum_{i=0}^{m-1} c_i H^i$. This multiplication can thus be performed simply using a linear feedback shift register (LFSR) with feedback polynomial being χ_H as shown in Fig. 6. The vertical inputs, in Fig. 6, labelled as $x_{i,j}$, $0 \leq j \leq m-1$ make the circuit to perform the multiplication by H followed by addition of $X_i^{(\mathcal{B}_H)}$, i.e., $Y^{(\mathcal{B}_H)} \times H + X_i^{(\mathcal{B}_H)}$, where $X_i^{(\mathcal{B}_H)}$ means that X_i is assumed to be expressed in $\mathcal{B}_H$.

If all parties using GCM/GHASH agree beforehand on the use of the $\mathcal{B}_H$ for representing $\mathrm{GHASH}_H(X)$, then the output of the LFSR does not need to be transferred to the $\mathcal{B}_t$ representation. We remark that to obtain the same GHASH bit string as that in Algorithm 1 (or Fig. 1), this transformation needs

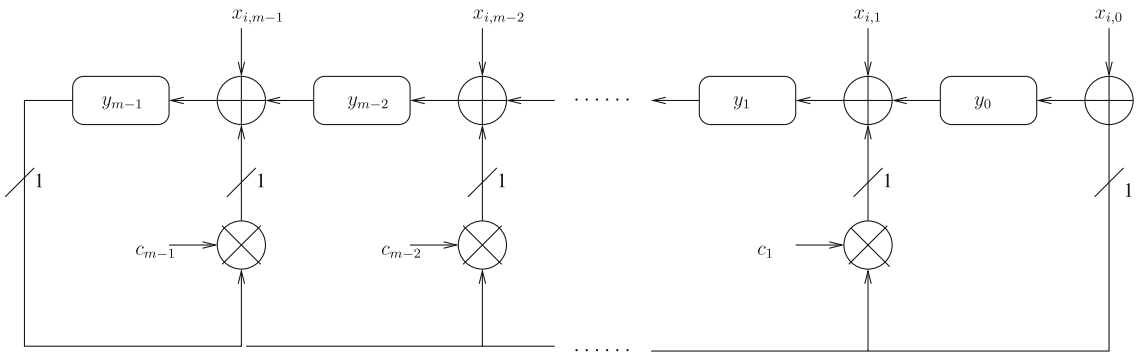

Fig. 6 MGHASH architecture with basis generated by H

to be performed. In addition, both inputs $H = \sum_{j=0}^{m-1} h_j t^j$ and $X_i = \sum_{j=0}^{m-1} x_{i,j} t^j$, $i = 1, 2 \ldots, n$ need to be converted to the $\mathcal{B}_H$ basis. It is worth mentioning that if Algorithm 2 is used, then instead of converting all n X_i blocks, only the m Y_j, $j = m-1, m-2, \ldots, 0$, which are the result of the reduction phase (i.e., the first loop), need to be converted.

The transformation of the hash key H from $\mathcal{B}_t$ to $\mathcal{B}_H$ representation is trivial: $H = [h_{m-1}, \ldots, h_1, h_0]_{\mathcal{B}_t} = [0, 0, \ldots, 0, 1, 0]_{\mathcal{B}_H}$. On the other hand, the transfer of the representation of X_i from $\mathcal{B}_t$ to $\mathcal{B}_H$ is not so simple and its cost could be similar to that of a matrix-vector multiplication.

In order to avoid the costly transformation of the representation of X_i, we follow a simple approach. We first note that block $X_i = [x_{i,m-1}, \ldots, x_{i,0}]$ is nothing but a bit string of length m and is part of the data that needs to be authenticated. For the GHASH computation that we have considered so far and as illustrated in [12], these m bits are considered to be the $\mathcal{B}_t$ representation of an element which is $\sum_{j=0}^{m-1} x_{i,j} t^j$. Below, we will treat the m bits of X_i to be the $\mathcal{B}_H$ representation of an element which is $\sum_{j=0}^{m-1} x_{i,j} H^j$. It is important to note that the latter is not the $\mathcal{B}_H$ representation of $\sum_{j=0}^{m-1} x_{i,j} t^j$, unless $H = t$. As a result, if $\sum_{j=0}^{m-1} x_{i,j} H^j$ is used as X_i in Algorithm 1, then there is no need to change the representation of X_i and the field multiplier can be replaced with an LFSR mentioned earlier. However, the algorithm will produce an output that will have a different bit string than the one of (7). Thus, this approach requires that both the sender and the receiver agree beforehand on the use of the $\mathcal{B}_H$ basis. Since the bit string in the new hash output is different than that from (7), we will refer to this scheme as *modified* GHASH (MGHASH).

We can clearly see in Fig. 6 that the MGHASH architecture requires m AND gates, $2m-1$ XOR gates and m flip-flops. We also see that the critical path delay of the architecture of Fig. 6 is equal to $2D_X$. The full computation of $\mathrm{MGHASH}_H(X)$ requires n iterations of the feedback register of Fig. 6 and it thus has a delay of $2nD_X$. Consequently, the use of basis $\mathcal{B}_H$ for field representation provides a very area efficient architecture. However, below we argue that a security flaw prevents the use of such an approach in a straight-forward way.

Security flaw Let us now explain how the use of the basis $\mathcal{B}_H$ to represent the field elements induces some weakness in MGCM, the modified GCM which uses MGHASH. The combined authentication and encryption algorithm of MGCM, takes as inputs an authentication data A and a plaintext P. It then outputs (IV, A, C, τ) where IV is a random initial value, C the ciphertext and τ the authentication tag, which are computed as follows

$$
\begin{aligned}
H &= E(K, 0^m) \\
J_0 &= \begin{cases} IV||0^{31}1 & \text{if } \mathrm{len}(IV) = m - 32 \\ MGHASH_H(IV) & \text{otherwise} \end{cases} \\
J_i &= \mathrm{incr}(J_{i-1}) \text{ for } i = 1, \ldots, n \\
C_i &= P_i + E(K, J_i) \\
C_n^* &= P_n^* + MSB_u(E(K, J_n)) \\
&\quad \text{where } u = (m - \mathrm{len}(P)) \mod m \\
X &= (A||0^v)||(C||0^u)||(\mathrm{len}(A)||\mathrm{len}(C)) \\
&\quad \text{where } v = (m - \mathrm{len}(A)) \mod m \\
\tau &= MSB_s(\mathrm{MGHASH}_H(X) + E(K, J_0))
\end{aligned}
$$

Note that the MSB_s function extract the s most significant bit of a block, the incr function adds one to the 32 least significant bit considered as an integer modulo 2^{32}.

Context of the forgery attack We consider the following scenario: An adversary has access to an oracle that generates ciphertexts with MGCM. The adversary does not necessary know the corresponding plaintext. The goal of the adversary is to generate a MGCM tuple, different to the ones produced by the oracle, which has a correct authentication code. The adversary is authorized to generate an MGCM tuple with the same IV as the one generated by the oracle.

Forgery attack on MGCM We assume that the oracle has output an MGCM tuple (IV, A, C, τ) with $\mathrm{len}(C) > 2m$. We will denote in the sequel n_A and n_C as the number of m-bit blocks of A and C respectively. We want to forge a second MGCM tuple with a valid authentication tag. For this purpose, we construct a second message $C' \neq C$ such that $\mathrm{MGHASH}_H(X') = \mathrm{MGHASH}_H(X)$. This message is given

below:

$$\begin{aligned} A'_i &= A_i \quad \text{for } i = 1, \ldots, n_A, \\ C'_i &= C'_i \quad \text{for } i = 1, \ldots, n_C - 3, \\ C'_{n_C-2} &= C'_{n_C-2} + [0^{m-2}01], \\ C'_{n_C-1} &= C_{n_C-1} + [0^{m-2}10], \\ C'_{n_C} &= C_{n_C}. \end{aligned}$$

The new message has the same length as (A, C). From the definition of (A', C') we can deduce the expression of the m-bit blocks of X', which is the input bit array to MGHASH

$$\begin{aligned} X'_i &= A'_i \quad \text{for } i = 1, \ldots, n_A - 1, \\ X'_{n_A} &= A'_{n_A} || 0^v, \\ X'_i &= C'_i \quad \text{for } i = n_A + 1, \ldots, n_A + n_C - 3, \\ X'_{n_A+n_C} &= C'_{n_A} || 0^u, \\ X'_{n_A+n_C+1} &= \text{len}(A') || \text{len}(C'). \end{aligned}$$

Let $n = n_A + n_C + 1$, we remark that $X'_i = X_i$ unless $i = n - 2$ and $i = n - 3$ where we have

$$X'_{n-3} = X_{n-3} + [0^{m-2}01] \quad \text{and} \quad X'_{n-2} = X_{n-1} + [0^{m-2}10]$$

We use this facts to express MGHASH(X') in terms of MGHASH(X) and we have

$$\begin{aligned} \text{MGHASH}_H(X') &= \underbrace{(X'_1 H^n + \cdots + X'_{n-4} H^5)}_{g'_2} \\ &\quad + \underbrace{(X'_{n-3} H^4 + X'_{n-2} H^3)}_{g'_1} \\ &\quad + \underbrace{(X'_{n-1} H^2 + X'_n H)}_{g'_0} \\ &= \underbrace{(X_1 H^n + \cdots + X_{n-4} H^5)}_{g_2} \\ &\quad + \underbrace{(X'_{n-3} H^4 + X'_{n-2} H^3)}_{g'_1} \\ &\quad + \underbrace{(X_{n-1} H^2 + X_n H)}_{g_0} \end{aligned} \tag{9}$$

We now consider g'_1 of MGHASH$_H(X')$. When we replace the expression of X'_{n-2} and X'_{n-3} in terms of X_{n-2} and X_{n-3}, we obtain, in basis $\mathcal{B}_H$, the following

$$\begin{aligned} H^4 X'^{\mathcal{B}_H}_{n-3} &= x_{n-3,m-1} H^{m+3} + \cdots \\ &\quad + x_{n-3,1} H^5 + (x_{n-3,0} + 1) H^4 \\ + H^3 X'^{\mathcal{B}_H}_{n-2} &= x_{n-2,m-1} H^{m+2} + x_{n-2,m-2} H^{m+1} + \cdots \\ &\quad + (x_{n-2,1} + 1) H^4 + x_{n-2,0} H^3 \\ \hline g'_1 &= x_{n-3,m-1} H^{m+3} + (x_{n-3,m-2} + x_{n-3,m-1}) H^{m+2} \\ &\quad + \cdots + (x_{n-3,0} + x_{n-2,1}) H^4 + x_{n-2,0} H^3 \end{aligned}$$

This last expression is thus equal to $g_1 = H^4 X_{n-3} + H^3 X_{n-2}$. Replacing g'_1 by g_1 in (9) implies

$$\text{MGHASH}_H(X') = \text{MGHASH}_H(X).$$

Finally, the adversary outputs the MGCM tuple (IV, A', C', τ) where τ is the authentication tag of (IV, A, C). The tag is still valid for the new MGCM tuple. This is because in the authentication of (IV, A', C', τ), we have $J'_0 = J_0$ since the IV does not change, and

$$\begin{aligned} \text{MGHASH}_H(X') + E(K, J'_0) &= \text{MGHASH}_H(X) \\ &\quad + E(K, J_0) = \tau \end{aligned}$$

which is correct.

Remark 4 The attack presented here does not apply to the original GCM. Indeed, in the original GCM, if we want to apply the attack, we would need to perform a change of representation from $\mathcal{B}_t$ to $\mathcal{B}_H$ at least for C_{n_C-2} and C_{n_C-1} in order to get the bits $c_{n_C-2,0}$ and $c_{n_C-1,1}$ and flip them. But such a change of representation requires the knowledge of H which is however secret. Nevertheless the attack shows that the basis or a set of bases may need to be explicitly identified for GHASH and similar other polynomial evaluation based hash functions, e.g., [16].

4.2 MGHASH with basis generated by $\sqrt[m]{H}$

The main reason behind the apparent success of the above attack is that when basis $\mathcal{B}_H$ is used to represent X_i's, the product terms like $X_1 H$ and $X_2 H^2$ in MGHASH introduces some fixed overlaps in their addition (e.g. in $X_1 H + X_2 H^2$) using $\mathcal{B}_H$ basis representations. In fact, other bases can also lead to similar overlaps. For example, a canonical basis generated by H^2 will result in overlaps (including the one between bit $x_{1,1}$ of X_1 and bit $x_{3,0}$ of X_2). As a result, the basis can be chosen such that the bits of X_i do not overlap in a fixed way in the multiply-and-add operation. Below, we consider such a basis that provides efficient implementation of MGHASH, yet it does not induce the above-mentioned weakness in MGCM. Specifically, we consider the following basis of $\mathbb{F}_{2^m}$

$$\mathcal{B}_{H'} = \{1, H', H'^2, \ldots, H'^{m-1}\},$$

where $H' = \sqrt[m]{H}$. In the sequel, we will still denote MGHASH for the modified GHASH based on $\mathcal{B}_{H'}$ as MGASH and the corresponding GCM by MGCM. Without loss of generality, we assume that m is a power of 2 (this is the case when the underlying used block cipher is AES). The following lemma will help us to design an efficient architecture for the MGHASH function.

Lemma 1 *Let $X = X_1 \ldots X_n$ be a bit string split in blocks of m bits each. We assume that blocks X_i, for $i = 1, 2, \ldots, n$,*

are expressed in $\mathcal{B}_{H'}$, i.e.,

$$X_i = x_{i,0} + x_{i,1}H' + x_{i,2}H'^2 + \cdots + x_{i,m-1}H'^{m-1}.$$

Let $P_X = \sum_{k=m}^{(n+1)m-1} p_k T^k$ be the binary polynomial with coefficients p_k defined by $p_k = x_{i,j}$ where $k = (n-i+1)m + j$. Then the MGHASH computation of X consists of the evaluation of P_X in H'

$$MGHASH_H(X) = P_X(H').$$

Proof By definition we have

$$\text{MGHASH}_H(X) = \sum_{i=1}^{n} X_i H^{n-i+1}.$$

In this expression of $\text{MGHASH}_H(X)$ we replace each X_i by its expression in $\mathcal{B}_{H'}$ and we obtain

$$\begin{aligned}\text{MGHASH}_H(X) &= \sum_{i=1}^{n} X_i H^{n-i+1} \\ &= \sum_{i=1}^{n} \left(\sum_{j=0}^{m-1} x_{i,j} H'^j \right) H^{n-i+1} \qquad (10)\end{aligned}$$

Now, by definition of H', we have $H = H'^m$, this yields that $\text{MGHASH}_H(X) = \sum_{i=1}^{n} \sum_{j=0}^{m-1} x_{i,j} H'^{(n-i+1)m+j}$. Finally, since $x_{i,j} = p_k$ where $k = (n-i+1)m + j$, this latter expression of MGHASH_H of (10) becomes

$$\text{MGHASH}_H(X) = \sum_{k=m}^{(n+1)m-1} p_k H'^k = P_X(H').$$

□

Now we can use this result to implement the MGHASH function. Since m is a power of 2, the characteristic polynomial of H and $H' = \sqrt[m]{H}$ are the same, i.e., we have $\chi_{H'} = \chi_H = (\sum_{i=0}^{m-1} c_i T^i) + T^m$. Then, from Lemma 1 we know that the computation $\text{MGHASH}_H(X)$ is given by $P_X(H')$ mod $\chi_{H'}$ (we can reduce modulo $\chi_{H'}$ since $\chi_{H'}(H') = 0$). We compute $P_X(H')$ mod $\chi_{H'}$ through a sequence of multiplications by H' modulo $\chi_{H'}$ and additions of p_k. The multiplication by H' modulo $\chi_{H'}$ is the same as the method as described in (8). The resulting MGHASH computation is given in Algorithm 3.

Algorithm 3 MGHASH computation using $\mathcal{B}_{H'}$

Require: P_X, H' and $\chi_{H'}$
Ensure: $Y = \text{MGHASH}_H(X)$
  for $j = 0$ **to** $m-1$ **do**
    $y_j \leftarrow p_{nm+j}$
  end for
  for $k = nm-1$ **to** 0 **do**
    for $j = 1$ **to** $m-1$ **do**
      $y_j \leftarrow y_{j-1} + c_j y_{m-1}$
    end for
    $y_0 \leftarrow c_0 y_{m-1} + p_k$
  end for
  return Y

We now design an architecture corresponding to Algorithm 3. We can use an LFSR which shifts the coefficients of Y to the left and then adds $c_i y_{m-1}$ to y_{i-1} for $i = 1, \ldots, m-2$. The coefficient p_k is added to $c_0 y_{m-1}$ to obtain the new value of y_0. The resulting hardware architecture, called binary polynomial reduction unit (BPRU), since the reduced polynomial is a binary polynomial, is depicted in Fig. 7.

Complexity The circuit shown in Fig. 7 requires m flip-flops and m XOR gates and m AND gates. The critical path delay is equal to $D_A + D_X$: it corresponds to the path used by y_{m-1} which goes through an AND gate and then an XOR gate. The final result $\text{MGHASH}_H(X) = P_X(H')$ mod $\chi_{H'}$ is computed after nm clock cycles. The overall delay of $\text{MGHASH}_H(X)$ computation is equal to $nm(D_A + D_X)$.

Security of MGHASH with $\mathcal{B}_{H'}$ We now provide some security proof of the $\mathcal{B}_{H'}$ based MGCM. Our proof follows the same steps as the one proposed by McGrew and Viega [9]. In the Appendix, we state several lemmas which replace the Lemmas of [9] in our context, as well as the resulting the-

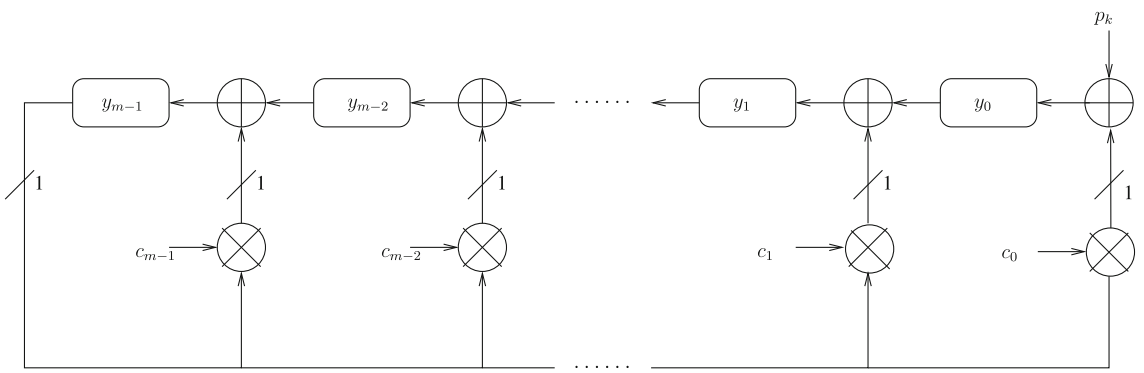

Fig. 7 Binary polynomial reduction unit (BPRU mod $\chi_{H'}$)

orems which can be proven the same way as in [9], but using the new lemmas presented in the Appendix of this paper.

The lemmas state the basic property used in the security proof. In particular, the main idea behing the proof consists of transfering the problem of forging a correct GHASH (or MGHASH) tag to finding a polynomial which has H as root (for more details we refer to [9] or the Appendix). Specifically, we have to following situations:

- *GHASH with* $\mathcal{B}_t$. The data $X = X_1 \dots X_n$ is transformed to a polynomial $P_X = \sum_{i=1}^{n} X_{n-i+1}^{(\mathcal{B}_t)} T^i \in \mathbb{F}_{2^m}$ and then evaluated at $H \in \mathbb{F}_{2^m}$.
- *MGHASH with* $\mathcal{B}_H$. The data $X = X_1 \dots X_n$ processed by MGHASH is transformed to a polynomial $P_X = \sum_{i=1}^{n} \sum_{j=0}^{m-1} x_{n-i+1,j} T^{i+j}$ in $\mathbb{F}_2[T]$ and then evaluated at $H \in \mathbb{F}_2[t]/(\chi_H(t))$.
- *MGHASH with* $\mathcal{B}_{H'}$. As stated in Lemma 1, the data $X = X_1 \dots X_n$ is transformed to the polynomial $P_X = \sum_{i=1}^{n} \sum_{j=0}^{m} \sum_{k=m}^{(n+1)m-1} x_{i,j} T^{(n-i+1)m+j}$ in $\mathbb{F}_2[T]$ and then evaluated in $H' \in \mathbb{F}_{2^m}[t]/(\chi_{H'}(t))$ (or equivalently a reduction modulo $\chi_{H'}(T)$).

We then remark GCM and MGCM with $\mathcal{B}_{H'}$ have a security which relies on a similar underlying problem. In the case of $\mathcal{B}_t$ and $\mathcal{B}_{H'}$ the transformation $X \mapsto P_X(T)$ is injective and the problem to find X such that H is a root of P_X has similar difficulties. On the other hand in the case of $\mathcal{B}_H$ the transformation $X \mapsto P_X(T)$ is not injective. The problem to find X such that $P_X(H) = 0$ can be solved as follows: we first find X such that $P_X(T) = 0$ this is possible since $X \to P_X(T)$ is not injective and then we have $P_X(H) = 0$.

4.3 Parallel architecture for MGHASH computation using $\mathcal{B}_{H'}$

Now we present a parallelized version of the previous MGHASH architecture. In order to parallelize the computation of MGHASH using $\mathcal{B}_{H'}$ we decompose the polynomial P_X (defined in Lemma 1) in r polynomials $P_1, \dots, P_r$ as follows

$$P_X(T) = T P_1(T^r) + \cdots + T^{r-1} P_{r-1}(T^r) + T^r P_r(T^r).$$

Let $d = (n+1)m - 1$ be the degree of P_X. For the sake of simplicity, we assume that d is a multiple of r. This means that $P_i = \sum_{\ell=0}^{d/r-1} p_{\ell r+i} T^{\ell}$ for $i = 1, \dots, r$. We define $H'' = H'^r$ and we assume that the characteristic polynomial $\chi_{H''}$ of H'' is known (if r is a power of 2 then $\chi_{H''} = \chi_{H'}$). The computation of $P_X(H')$ is then done in two steps:

- we compute $G_i = P_i(H'') \mod \chi_{H''}$ in parallel using a BPRU with $\chi_{H''}$ the characteristic polynomial of H'';
- we now remark that $\mathrm{MGHASH}_H(X) = H'G_1 + \cdots + H'^{r-1} G_{r-1} + H'^r G_r$. The reduced expressions $G_i, i = 1, \dots, r$ are given in $\mathcal{B}_{H'}$; consequently, $G_i = \sum_{j=0}^{m-1} g_{i,j} H''^j = \sum_{j=0}^{m-1} g_{i,j} H'^{rj}$ and this implies that

$$\mathrm{MGHASH}_H(X) = \sum_{j=0}^{m-1} \sum_{i=1}^{r} g_{i,j} H'^{rj+i} \tag{11}$$

which is a polynomial in H' of degree $rm-1$. We can apply a binary reduction process modulo $\chi_{H'}$ through a BPRU architecture to obtain the reduced result of $\mathrm{MGHASH}_H(X)$ modulo $\chi_{H'}$ Fig. 8.

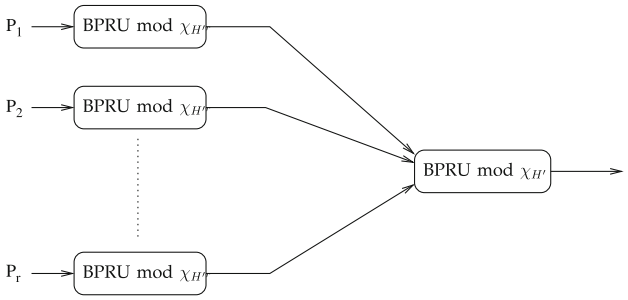

Fig. 8 Parallel MGHASH architecture using $\mathcal{B}_{H'}$

The resulting r-parallel MGHASH architecture using $\mathcal{B}_{H'}$ is shown in Fig. 7.

Complexity We first evaluate the space complexity: it is equal to $r+1$ times the space complexity of a single BPRU. The complexity of a single BPRU has been evaluated in the previous subsection. Consequently, the architecture shown in Fig. 7 requires $(r+1)m$ XOR gates and $(r+1)m$ AND gates and $(r+1)m$ flip-flops. The critical path delay of one BPRU is equal to $D_{\mathrm{A}} + D_{\mathrm{X}}$. Then, since each P_i has degree d/r, the delay for the computation of $P'_1, \dots, P'_r$ is equal to $(d/r - m)(D_A + D_X) = (\frac{(n+1)m-1}{r} - m)(D_A + D_X)$. Similarly, since the degree of the polynomial in H' in (11) is equal to rm, the number of iterations in the BPRU is equal to $rm - m$. Thus, the delay of the second step is equal to $(r-1)m(D_{\mathrm{A}} + D_{\mathrm{X}})$. Finally we obtain the following expression for the overall delay for the computation of $\mathrm{MGHASH}_H(X)$

$$\mathcal{D} = \left(\frac{(n+1)m - 1}{r} - m + (r-1)m \right) (D_{\mathrm{A}} + D_{\mathrm{X}}).$$

5 Complexity comparison

In this section, we compare the different approaches presented in Sects. 3 and 4. Their corresponding complexities are given in Table 2. For comparison purposes we give the complexities of the two approaches of [10] reviewed in Sect. 2. In

Table 2 Complexities of GHASH and MGHASH architectures

Method	Space complexity				Time complexity
	# AND	# XOR	# FM	# flip-flops	
Multiply-and-Add [10]	0	m	1	0	$n(D_{\mathrm{M}}+D_{\mathrm{X}})$
CharPol With $\mathcal{B}_t$	m^2	m^2+m	1	m^2	$(n-m)(D_{\mathrm{X}}+D_{\mathrm{A}})+(m-1)(D_{\mathrm{M}}+D_{\mathrm{X}})$
CharPol With $\mathcal{B}_{H'}$	m	m	0	m	$nm(D_{\mathrm{A}}+D_{\mathrm{X}})$
r-Parallel Multiply-and-Add [10]	0	$(2r-1)m$	r	0	$\frac{n}{r}D_{\mathrm{M}}+(\frac{n}{r}-1+\lceil\log_2 r\rceil)D_{\mathrm{X}}$
r-Parallel CharPol With $\mathcal{B}_t$	$(r+1)m^2$	$(r+1)m^2+m$	1	$(r+1)m^2$	$(\frac{n}{r}+(r-2)m)(D_{\mathrm{X}}+D_{\mathrm{A}})+(m-1)D_{\mathrm{M}}$
r-Parallel CharPol With $\mathcal{B}_{H'}$	$(r+1)m$	$(r+1)m$	0	$(r+1)m$	$(\frac{(n+1)m-1}{r}+(r-2)m)(D_{\mathrm{A}}+D_{\mathrm{X}})$

Table 2, #FM represents the number of field multipliers and #FF the number of flip-flops required in the architectures.

Comparison of CharPol with $\mathcal{B}_t$ with multiply-and-add Asymptotically (i.e. for large n), the new method reduces the critical path delay of the GHASH function to that of one XOR plus one AND operation (i.e. $O(1)$), whereas that of the traditional method is due to the multiplier, i.e. $O(\log m)$ for bit-parallel implementations. This reduction in the critical path is obtained at the cost of a PRU. However, the number of multiplications in the final Horner scheme being constant, using a smaller but slower multiplier can greatly reduce the additional cost, in terms of space, without changing the overall asymptotic complexity.

In hardware implementation, D_{X} is normally larger than D_{A}, and more importantly, for bit parallel implementation we have $D_{\mathrm{M}} \geq (\log_2 m)D_{\mathrm{X}}$. For example, as mentioned in Table 1, the Karatsuba algorithm based bit-parallel multiplier over $\mathbb{F}_{2^{128}}$ has $D_{\mathrm{M}} > 7D_{\mathrm{X}}$. Thus for a large n (i.e., long messages), it is clear that the delay of architecture based on CharPol with $\mathcal{B}_t$ is considerably smaller than the delay of multiply-and-add architectures.

Comparison of CharPol with $\mathcal{B}_t$ versus CharPol with $\mathcal{B}_{H'}$ A direct comparison of complexity values for non-parallelized approaches gives that the CharPol method with $\mathcal{B}_t$ is much faster: it is m times faster than Charpol using $\mathcal{B}_{H'}$. But, on the other hand it requires a circuit which is m times larger.

The main difference between the two approaches is that CharPol with $\mathcal{B}_t$ processes blocks with m bits long while CharPol with $\mathcal{B}_{H'}$ processes only one bit at a time. It is thus more appropriate to compare CharPol with a $\mathcal{B}_t$ with m-parallel CharPol with $\mathcal{B}_{H'}$. In this case both approaches have similar delay when n is large

- CharPol with $\mathcal{B}_t$ has a delay of $(n-m)(D_{\mathrm{A}}+D_{\mathrm{X}})+(m-1)(D_{\mathrm{M}}+D_{\mathrm{X}})$,
- m-parallel CharPol with $\mathcal{B}_{H'}$ has a delay of $\leq (n+1)+(m-2)m(D_{\mathrm{A}}+D_{\mathrm{X}})$.

Considering the space complexity, m-parallel CharPol with $\mathcal{B}_{H'}$ requires m^2+m of XORs, ANDs and flip-flops while CharPol with $\mathcal{B}_t$ requires $\cong m^2$ of XORs, ANDs and flip-flops plus one field multiplier. Consequently, CharPol with $\mathcal{B}_{H'}$ approach saves the cost of one multiplier.

6 Computation of the characteristic polynomial over $\mathbb{F}_{2^{128}}$

The GHASH computation methods presented in Sects. 3 and 4 require us to compute of characteristic polynomials of a certain element H. In this section we discuss this issue.

We first recall a method from Gordon [5] that determines the characteristic polynomial of an element of a finite field and requires, in our context, $\frac{127\times 128}{2}$ multiplications in $\mathbb{F}_{2^{128}}$ and 127 squarings. We then propose a new method taking advantage of the tower structure of $\mathbb{F}_{2^{128}}$ that is faster than the method of Gordon when we use a specific representation of the finite field.

Gordon's method Let $A \in \mathbb{F}_{2^m}$. Then the characteristic polynomial of A is given by

$$\chi_A(T) = \prod_{i=0}^{m-1} \left(T + A^{2^i}\right).$$

The method of Gordon computes sequentially for $j = 0, \ldots, m-1$ the polynomials $\chi_{A,j}(T) = \prod_{i=0}^{j}(T + A^{2^i})$. Specifically we start from $\chi_{A,0} = T + A$ and we apply the following formula $\chi_{A,j} = \chi_{A,j-1} \times (T + A^{2^j})$ for $j = 1, \ldots, m-1$. We then obtain Algorithm 4.

Gordon's method is quite general and can be applied to any binary field. We now propose a new method that takes into account that $\mathbb{F}_{2^{128}}$ has a very special structure. We will use the following lemma to compute the polynomial χ_A. Let $\chi(T) = \sum_{i=0}^{d} c_i T^i$ be a polynomial in $\mathbb{F}_{2^m}[T]$ and k an integer. We denote

Algorithm 4 Gordon's method for characteristic polynomial calculation

Require: $A \in \mathbb{F}_{2^m}$
Ensure: χ_A (the minimal polynomial of A)
  $\chi \leftarrow T + A$
  $Z \leftarrow A$
  for $i = 1$ to $m - 1$ **do**
    $Z \leftarrow Z^2$
    $\chi_A \leftarrow \chi_A \times T \ + \chi_A \times Z$
  end for
  return (χ_A)

$$\sigma_k(\chi) = \sum_{i=0}^{d} c_i^{2^k} T^i.$$

Lemma 2 *Let $\mathbb{F}_{2^m}$ be a binary field, and let $\mathbb{F}_{2^{2m}}$ a degree 2 field extension of $\mathbb{F}_{2^m}$. The following assertions hold*

1. *If $\chi, \chi' \in \mathbb{F}_{2^m}[T]$ and k is an integer then*

   $$\sigma_k(\chi\chi') = \sigma_k(\chi)\sigma_k(\chi').$$

2. *If $\chi \in \mathbb{F}_{2^{2m}}[T]$ satisfies $\chi(A) = 0$, then the polynomial $\chi' = \chi \times \sigma_m(\chi)$ satisfies*

   $$\chi'(A) = 0 \text{ and } \chi' \in \mathbb{F}_{2^m}[T].$$

Proof 1. Let us write $\chi = \sum_{i=0}^{d} c_i T^i$ and $\chi' = \sum_{i=0}^{d'} c'_i T^i$ then

$$\chi\chi' = \sum_{i=0}^{d}\sum_{j=0}^{d'} c_i c'_j T^{i+j}.$$

Then, we apply σ_k to $\chi\chi'$

$$\begin{aligned}\sigma_k(\chi\chi') &= \sigma_k\left(\sum_{i=0}^{d}\sum_{j=0}^{d'} c_i c'_j T^{i+j}\right)\\ &= \sum_{i=0}^{d}\sum_{j=0}^{d'} (c_i c'_j)^{2^k} T^{i+j}\\ &= \sum_{i=0}^{d}\sum_{j=0}^{d'} c_i^{2^k} c_j'^{2^k} T^{i+j}\\ &= \sigma_k(\chi)\sigma_k(\chi')\end{aligned}$$

which proves the assertion.

2. We now prove the second assertion of the lemma. Let $\chi' = \chi \times \sigma_{2^m}(\chi)$. We first check that $\chi'(A) = 0$, we have

$$\chi'(A) = \chi(A)\sigma_m(\chi)(A) = 0 \times \sigma_m(\chi)(A) = 0$$

In order to show that $\chi' \in \mathbb{F}_{2^m}[T]$ we have to prove that $\sigma_m(\chi') = \chi'$, since, in this case, each coefficient of χ is in $\mathbb{F}_{2^m}$. We compute

$$\sigma_m(\chi') = \sigma_m(\chi)\sigma_m(\sigma_m(\chi))$$

But $\sigma_m((\sigma_m(\chi)) = \sigma_{2m}(\chi) = \chi$ since $\chi \in \mathbb{F}_{2^{2m}}[T]$. Finally

$$\sigma_m(\chi') = \chi\sigma_m(\chi) = \chi'$$

This completes the proof.

Let us now see, how to use the above result to compute the characteristic polynomial of an element $A \in \mathbb{F}_{2^8}$ over $\mathbb{F}_2$. We begin with the polynomial $\chi_0 = T + A$ which satisfies $\chi(A) = 0$ and $\chi \in \mathbb{F}_{2^8}[T]$.

- We apply Lemma 2 for the field extension $\mathbb{F}_{2^8}/\mathbb{F}_{2^4}$ to obtain a polynomial $\chi_1 = \chi_0 \times \sigma_4(\chi_0)$ which satisfies

  $$\chi_1(A) = 0 \text{ and } \chi_1 \in \mathbb{F}_{2^4}[T]$$

  Moreover we have $\chi_1 = (T + A)(T + A^{2^4})$.
- We apply again Lemma 2 for the field extension $\mathbb{F}_{2^4}/\mathbb{F}_{2^2}$ to obtain a polynomial $\chi_2 = \chi_1 \times \sigma_2(\chi_1)$ which satisfies

  $$\chi_2(A) = 0 \quad \text{and} \quad \chi_2 \in \mathbb{F}_{2^2}[T].$$

  Moreover we also have $\chi_2 = (T + A)(T + A^{2^4})(T + A^{2^2})(T + A^{2^6})$.
- Finally we apply the lemma a third time for the field extension $\mathbb{F}_{2^2}/\mathbb{F}_2$ to the polynomial χ_2. We get

$$\begin{aligned}\chi_3 &= \chi_2 \times \sigma_1(\chi_2)\\ &= \underbrace{(T + A)(T + A^{2^4})(T + A^{2^2})(T + A^{2^6})}_{\chi_2}\\ &\quad \times \underbrace{(T + A^2)(T + A^{2^5})(T + A^{2^3})(T + A^{2^7})}_{\sigma_2(\chi_2)},\end{aligned}$$

and $\chi_3 \in \mathbb{F}_2[T]$ and satisfies $\chi_3(A) = 0$. If we look at the expression of χ_3 we can see that it is equal to the characteristic polynomial of A.

This method is generalized for the binary field $\mathbb{F}_{2^m}$ where $m = 2^k$ is a power of 2 in Algorithm 5.

Proposition 1 *Algorithm 5 returns the characteristic polynomial of A over $\mathbb{F}_2$.*

Algorithm 5 Computing the characteristic polynomial of A

Require: $A \in \mathbb{F}_{2^{2^k}}$
Ensure: χ_A (the characteristic polynomial of A over $\mathbb{F}_2$)
  $\chi \leftarrow T + A$
  for $i = k - 1$ to 0 **do**
    $\chi \leftarrow \chi \times \sigma_{2^i}(\chi)$
  end for
  return (χ)

Proof Applying Lemma 1, for successive extension field of degree 2 shows that the returned polynomial χ satisfies $\chi(A) = 0$ and $\chi \in \mathbb{F}_{2^m}$. Let us now prove that the polynomial χ is the characteristic polynomial of χ, i.e., we prove that the returned χ satisfies

$$\chi(T) = \prod_{i=0}^{m-1} (T + A^{2^i}).$$

We prove by induction that the computed χ after the jth loop is the characteristic polynomial of A over the field $\mathbb{F}_{2^{2^{k-j}}}$

$$\chi = \prod_{i=0}^{2^j-1} (T + A^{i2^{k-j}}).$$

- For $j = 0$ this is clear that $\chi = T + A$ is the characteristic polynomial of A over $\mathbb{F}_{2^{2^k}}$.
- For $j = 1$ we have $P = (T + A)(T + A^{2^{2^{k-1}}})$, thus the assertion is true.
- Suppose that $\chi = \prod_{i=0}^{2^j-1}(T + A^{2^{id}})$ after the jth loop where $d = 2^{k-j}$, then

$$\begin{aligned}
\chi\sigma_{2^{k-j-1}}(\chi) &= \left(\prod_{i=0}^{2^j-1}\left(T + A^{2^{id}}\right)\right) \\
&\quad \times \left(\prod_{i=0}^{2^j-1}\left(T + A^{2^{id}\times 2^{2^{k-j-1}}}\right)\right) \\
&= \left(\prod_{i=0}^{2^j-1}\left(T + A^{2^{2id'}}\right)\right) \\
&\quad \times \left(\prod_{i=0}^{2^j-1}\left(T + A^{2^{2id'}\times 2^{d'}}\right)\right) \\
&= \prod_{i=0}^{2^{j+1}-1}\left(T + A^{2^{id'}}\right)
\end{aligned}$$

where $d' = 2^{k-(j+1)}$. Consequently, χ has the required expression after the $j + 1$th loop and this completes the proof. □

Complexity Let us now evaluate the complexity of Algorithm 5. We suppose that each polynomial multiplication is performed using the Karatsuba method, and each exponentiation to 2^k is done by computing k successive squares.

We first make two observations:

- Let us first check that the degree of χ after j iterations of the **for** loop has degree 2^j. This is true when $j = 0$ since χ is initialized by $T + A$. In each loop the degree of P is multiplied by 2, thus after j loop, the degree of P must be equal to 1×2^j.
- At the end of the jth iteration of the **for** loop, the polynomial χ belongs to $\mathbb{F}_{2^{k-j}}$. Indeed, for $j = 1$, $\chi = (T + A)\sigma_{2^{2^{k-1}}}(T + A)$ with $A \in \mathbb{F}_{2^{2^k}}$, so $\chi \in \mathbb{F}_{2^{2^{k-1}}}[T]$ from Lemma 1. If we suppose that after j iterations $\chi \in \mathbb{F}_{2^{2^{k-j}}}[T]$, then, at step $j + 1$ we compute $\chi = \chi \times \sigma_{2^{k-(j+1)}}(\chi)$, which belongs to $\mathbb{F}_{2^{2^{k-(j+1)}}}[T]$.

From the above two remarks, we can now evaluate the complexity of Algorithm 5.

- At the jth iteration of the **for** loop, χ is a monic polynomial of degree 2^j (and thus has 2^j coefficients). As its coefficients are in $\mathbb{F}_{2^{2^{k-j+1}}}$ and that we need to compute the 2^{k-j}th power of each of them, we have to perform $2^j \times 2^{k-j}$ squarings over $\mathbb{F}_{2^{2^{k-j+1}}}$ to compute $\sigma_{2^{k-j}}(\chi)$. Let us denote S_{2^k} as the cost of a squaring over $\mathbb{F}_{2^{2^k}}$, we assume that $S_{2^k} = 2S_{2^{k-1}}$ (squaring is linear over binary fields). Then, the total complexity of computing $\sigma_{2^{k-j}}(\chi)$ is

$$\begin{aligned}
\sum_{j=1}^{k} 2^j 2^{k-j} S_{2^{k-j+1}} &= \sum_{j=1}^{k} 2^k \frac{S_{2^k}}{2^{j-1}} \\
&= 2^k S_{2^k} \sum_{j=1}^{k} \frac{1}{2^{j-1}} \\
&= 2^k S_{2^k}(2 - 2^{-(k-1)}) \\
&= 2^{k+1} S_{2^k}(1 - 2^{-k}) \\
&\sim 2^{k+1} S_{2^k}
\end{aligned}$$

- In the same manner, at the jth iteration, after computing $\sigma_{2^{k-j}}(\chi)$, the algorithm computes $\chi\sigma_{2^{k-j}}(\chi)$. Thus, we multiply two degree 2^j polynomials. Performing this multiplication using the Karatsuba algorithm requires 3^j multiplications of fields elements. Let us denote M_{2^k} as the cost of a multiplication over $\mathbb{F}_{2^{2^k}}$, we assume that $M_{2^k} = 3M_{2^{k-1}}$. Then, the computational cost is equal to $\sum_{j=1}^{k} 3^j M_{2^{k-j+1}}$, so we obtain

$$\sum_{j=1}^{k} 3^j M_{2^{k-j+1}} = \sum_{j=1}^{k} 3^j \frac{M_{2^k}}{3^{j-1}}$$

$$= \sum_{j=1}^{k} 3M_{2^k}$$
$$= 3kM_{2^k}$$

In the end, the overall cost of computing the characteristic polynomial of an element of $\mathbb{F}_{2^{2^k}}$ over $\mathbb{F}_2$ is, in terms of number of operations over $\mathbb{F}_{2^{2^k}}$, equal to $3kM_{2^k} + 2^{k+1}S_{2^k}$.

In the case of GCM, $k = 7$ and thus the number of field operations is 21 multiplications and 256 squarings over $\mathbb{F}_{2^{128}}$.

Remark 5 It is not always suitable to decompose the field $\mathbb{F}_{2^{2^k}}$ into k extensions of degree 2. As an example, $\mathbb{F}_{2^{128}}$ can be seen as a degree 4 extension of $\mathbb{F}_{2^{32}}$ and $\mathbb{F}_{2^{32}}$ a degree 32 extension of $\mathbb{F}_2$. In that case, multiplying two elements of $\mathbb{F}_{2^{64}}$ is performed on the extension field $\mathbb{F}_{2^{128}}$, which means that $M_{2^7} = M_{2^6}$. In the end, the total complexity would be:

$$\sum_{j=1}^{2} 3^j M_{2^7} + \sum_{j=3}^{6} 3^j M_{2^5} + 3^7 M_2$$
$$= 12M_{2^7} + 1080M_{2^5} + 2187M_2 \quad = 135M_{2^7}$$

This shows that depending on the representation, the computational cost might vary. However, this computation is done once and for all at the beginning of a GCM session. As such a session can involve thousands of field multiplications (the plain text can have up to 2^{39} bits, i.e. 2^{32} 128-bit blocks), this additional cost can be considered as negligible for long sessions.

Remark 6 For large values of n, the computation of GHASH using the characteristic polynomial is expected to be several times faster than traditional methods that use n field multiplications. For example, consider $n = 10,000$ and the FPGA based bit parallel multiplier from [14], which has a delay of $D_M = 6.637$ ns. This multiplier is based on the Karatsuba algorithm and hence $D_M = 3(\log_2(2^7))D_X + D_A$, i.e., ignoring the delay due to the single level of AND gates, we have $D_M/D_X \approx 21$. Thus, the traditional method for computing GHASH will require $10,000 \times (D_X + D_M) \approx 69.6$ μs. On the other hand, the characteristic polynomial based GHASH using the same multiplier will require approximately $10,000 \times (D_X + D_A) + 128 \times (D_M + D_X) + 135 \times D_M \approx 8.1$ μs, resulting in more than 8 fold reduction in the computation time.

7 Conclusions

In this paper, we have proposed a new way to improve the performance of the GHASH function of GCM. Our first method is based on the use of the degree m characteristic polynomial of hash key. We could thus eliminate $n - m$ field multiplications involved during the authentication tag computation. Each of these $n-m$ multiplications is replaced by m independent field additions. This is very attractive for high performance implementations where m additions can be performed in parallel. This allows us to reduce the delay of each of the first $n - m$ multiplications over $\mathbb{F}_{2^m}$ to that of one XOR and one AND operation. To illustrate the effectiveness of the proposed method, we have considered $n = 10,000$, $m = 128$ and the Karatsuba algorithm based bit parallel multiplier on FPGA from [14], which has a delay of $D_M = 6.637$ ns and $D_M/D_X \approx 21$, and we have estimated that compared to the traditional method, the new method can significantly reduce the GHASH computation time, that it to say, eight times.

We have also presented a different approach which uses a specific representation of the field $\mathbb{F}_{2^m}$. This variant has the same advantage of the former one regarding the delay of the GHASH computation, but it is cheaper in space since it does not require any field multiplier. We also argue that special care needs to be taken while choosing an arbitrary basis of representation for polynomial evaluation based universal hash functions.

In this paper, we have also shown the flexibility of our methods in terms of parallelization. Using multiple PRUs allows us to efficiently parallelize the computations and improve the performance of GHASH even further.

Finally, we have also proposed a method, specific to $\mathbb{F}_{2^{128}}$, to compute the initial characteristic polynomial efficiently.

Acknowledgments The authors are grateful to the reviewers for their constructive feedback on the paper. A preliminary version of this work was published in the proceedings of ACNS 2010 [11]. That preliminary work concerned the characteristic polynomial approach given in Sect. 3 and also the method to compute the characteristic polynomial of an element in $\mathbb{F}_{2^{2^s}}$ presented in Sect. 6. This work was supported in part by an NSERC grant awarded to Dr. Hasan.

Appendix A. Definition of modified GHASH (MGHASH) and modified GCM (MGCM)

We assume that we have a block cipher $E(K, P)$ which encrypts a plaintext P of bit length m with a key K. Here we first explicitly define MGHASH and consider that H is the hash key, i.e., H is the m-bit block $E(K, 0^m)$. We assume that H is an element of $\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(f(t))$ for a fixed $f(t)$ of degree m. We denote $\chi_H(T) \in \mathbb{F}_2[T]$ as the characteristic polynomial of H. The characteristic polynomial χ_H is irreducible if H is not an element of any subfield of $\mathbb{F}_{2^m}$. Here we will use the following definition of MGHASH.

Definition 1 Let A be some public authentication data and C a ciphertext. We consider the m bit block $L = (\text{len}(A)||\text{len}(C))$ and define $X = A||C||L$ of bit length $\ell = \text{len}(X)$, i.e., $X = x_1 \ldots x_\ell$. Then we define

$$\text{MGHASH}(X) = \left(\sum_{i=0}^{\ell-1} x_{\ell-i} T^{i+m}\right) \mod \chi_H(T). \quad (12)$$

The definition of MGHASH uses a polynomial expression in indeterminate T. It is an expression equivalent to the one given in Subsect. 4.2. Following Lemma 1, the computation of $\text{MGHASH}_H(X)$ can be computed as a polynomial reduction $P_X(H') \mod \chi_{H'}$. But this latter computation consists only of the formal computation of binary polynomial in H' and furthermore, since m is assumed to be a power of 2, we have $\chi_{H'} = \chi_H$. Consequently, this corresponds to the expression of (12).

We can use the definition of MGHASH to design a modified GCM (MGCM). We denote n_P the number of m-bit block containing the plaintext P, u the number of bits of P_{n_P} and A the authentication data consisting of n_A blocks and $\text{len}(A_{n_A}) = v$. Then $\text{MGCM}(K, A, P) = (IV, A, C, \tau)$ where ciphertext C and tag τ are defined below

$$\begin{cases} H &= E(K, 0^m), \\ J_0 &= \begin{cases} IV||0^{31}1 & \text{if len}(IV) = m - 32 \\ \text{MGHASH}(H, \{\}, IV) & \text{otherwise} \end{cases} \\ J_i &= \text{incr}(J_{i-1}) \text{ for } i = 1, \ldots, n_P, \\ C_i &= P_i \oplus E(K, J_i) \text{ for } i = 1, \ldots, n_P - 1, \\ C_n &= P_n^\star \oplus MSB_u(E(K, J_n)), \\ X &= (A||0^v)||(C||0^u)||(\text{len}(A)||\text{len}(C)), \\ \tau &= MSB_s(\text{MGHASH}(H, A, C) \oplus E(K, J_0)). \end{cases}$$

We further assume that the bit length $\text{len}(P)$, which is equal to $\text{len}(C)$, is between 0 and $(2^{32} - 2)m$ bits, $\text{len}(A)$ is between 0 and $2^{m/2}$ bits, $\text{len}(IV)$ is between 1 and $2^{m/2}$ bits and $s = \text{len}(\tau) \leq m$.

Appendix B. Lemmas

Here we state the corresponding lemmas for MGHASH in replacement of the GHASH lemmas in the security proofs of GCM [9]. We first state Lemma 3 which concerns the probability that $\text{MGHASH}_H(X) = 0$.

Lemma 3 *Let $\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(f(t))$ be a binary field with degree m and a fixed irreducible polynomial f. Let H be a randomly chosen element in $\mathbb{F}_{2^m}$ and $\chi_H(T) \in \mathbb{F}_2[T]$ its characteristic polynomial over $\mathbb{F}_2$. We consider an $(\ell+1)$-bit sequence $R = r_0, \ldots, r_\ell$ then*

$$\mathbf{P}\left[\sum_{i=0}^{\ell} r_i T^i = 0 \mod \chi_H(T) \mid H \xleftarrow{\$} \mathbb{F}_{2^m}\right] \leq \frac{\ell}{2^m}$$

Proof We assume that we have $R(T) = \sum_{i=0}^{\ell} r_i T^i = 0 \mod \chi_H(T)$. Let $M_H(T)$ be the minimal polynomial of H. Since, by definition, M_H divides χ_H we have $\sum_{i=0}^{\ell} r_i T^i = 0 \mod M_H$ and since $(T - H)$ divides M_H, we have $\sum_{i=0}^{\ell} r_i T^i = 0 \mod (T - H)$ and thus H is a root of $R(T)$. Since $R(T)$ has at most ℓ roots and since H is chosen randomly in $\mathbb{F}_{2^m}$, then the probability that H is equal to one of these ℓ roots is no more that $\ell/2^m$.

Now, we use this result to show that MGHASH is a ε-almost XOR universal function for some ε.

Definition 2 (ε-*almost XOR universal*) A function $g(K, M)$ is ε-almost XOR universal if

$$\mathbf{P}\left[g(K, X) \oplus g(K, X') = a \mid *^{m-s} \mid K \xleftarrow{\$} \{0, 1\}^{\ell_K}\right] \leq \varepsilon$$

for all $X \neq X'$ of length $\leq \ell$ and a of length s and where the expression $K \xleftarrow{\$} \{0, 1\}^{\ell_K}$ means that K is chosen randomly in the set $\{0, 1\}^{\ell_K}$.

Lemma 4 (MGHASH is almost XOR universal) *The function MGHASH is $(\ell + m)2^{-s}$-almost XOR universal when the sum of the lengths of its input are restricted to ℓ or fewer bits and its output is truncated to s bits.*

Proof We consider two distinct inputs (A, C) and (A', C') and X and X' their corresponding bit sequence used in MGHASH such that $\text{len}(X), \text{len}(X') \leq \ell$ and we consider a of length s bits. We then analyze the probability of the event

$$\text{MGHASH}_H(X) \oplus \text{MGHASH}_H(X') = a||z, \quad (13)$$

for some z of bit length $(m - s)$ and under the condition that H is randomly chosen. We rewrite (13) by replacing $\text{MGHASH}_H(X)$ and $\text{MGHASH}_H(X')$ by their respective polynomial expression in T

$$\left(\sum_{i=0}^{m-t-1} z_i T^i + \sum_{i=m-t}^{m-1} a_{i-t} T^i + \sum_{i=m}^{\ell+m-1} (x_{\ell-i+m} + x'_{\ell-i+m}) T^i\right) \mod \chi_H(T) = 0 \quad (14)$$

Since $X \neq X'$ this expression involves nonzero polynomial in T. Lemma 3 tells us that the probability that (14) is satisfied is less than $(\ell + m)/2^m$.

Now, since there are 2^{m-s} possible values for z, we obtain

$$\mathbf{P}\left[\text{MGHASH}_H(X) \oplus \text{MGHASH}_H(X') = a \,||\, *^{m-s} \mid H \xleftarrow{\$} \mathbb{F}_{2^m}\right] \leq (\ell + m)/2^s.$$

We consider the following function, which is used to generate J_0, the initial counter in MGCM.

$$\gamma(H, P) = \begin{cases} P||0^{31}||1 & \text{if len}(P) = m - 32 \\ \text{MGHASH}_H(P) & \text{otherwise.} \end{cases} \quad (15)$$

Lemma 5 (γ is unlikely to collide) *The inequality*

$$\mathbf{P}\left[\gamma(H, IV) = incr^j(\gamma(H, IV')) \text{ s.t. } H \xleftarrow{\$} \mathbb{F}_{2^m}\right] \leq \ell_{IV} 2^{-m}, \quad (16)$$

holds for the function γ defined in (15) for any value of j, IV and IV' where the inputs IV and IV' are distinct and have bit lengths $\leq \ell_{IV}$.

Proof We call $\mathcal{E}$ the event $\gamma(H, IV) = \text{incr}^j(\gamma(H, IV'))$ for some $j \geq 0$. We consider the following cases

1. $\text{len}(IV) = \text{len}(IV') = m - 32$,
2. $\text{len}(IV) = m - 32$ and $\text{len}(IV') \neq m - 32$,
3. $\text{len}(IV) \neq m - 32$, and $\text{len}(IV') = m - 32$,
4. $\text{len}(IV) \neq m - 32$, and $\text{len}(IV') \neq m - 32$.

For case 1, since IV and IV' are distinct, we can easily conclude that $\gamma(H, IV) = \gamma(H, IV')$ happens with probability 0 for any j in the incr^j function. For the case 2, when $\mathcal{E}$ occurs we have

$$\gamma(H, IV') = \text{incr}^{-j}(IV||0^{31}||1) = (IV||(1 - j \mod 2^{32}))$$

This condition can be re-expressed as $R(T) = 0 \mod \chi_H$, where the binary polynomial R has a degree $\leq \ell_{IV} + m$ and is defined by $R = \sum_{i=0}^{\ell+m} r_i T^i$ where the coefficient r_i is as follows

$$r_i = \begin{cases} (1 - j \mod 2^{32})_i & \text{for } i = 0, \ldots, 31 \\ (IV)_{i-32} & \text{for } i = 32, \ldots, m-1 \\ (IV')_{\text{len}(IV')+m-i} & \text{for } i = m, \ldots, \ell + m - 1 \end{cases}$$

Note that we have denoted $(M)_i$ as the ith bit of array M. There are at most $\ell_{IV} + m$ values of $H \in \mathbb{F}_{2^m}$ for which $R(H) = 0$ holds. And this is true for any j, since each value of j corresponds to a distinct polynomial. Consequently, the probability of $\mathcal{E}$, given that H is chosen at random in $\mathbb{F}_{2^m}$ is equal to $(\ell_{IV} + m)/2^m$ (cf. Lemma 4). Case 3 is identical to Case 2 after exchanging the role of IV and IV'.

In case 4, the probability that $\gamma(H, IV) = \gamma(H, IV')$ is equal to the probability that $\text{MGHASH}_H(IV) = \text{MGHASH}_H(IV')$, which is no greater that $\lceil(\ell_{IV} + m)/2^m\rceil$ as per Lemma 4. Thus, we have shown that the claimed result holds for all four cases.

Lemma 6 (γ is unlikely to return zero) *The function γ defined in (15) satisfies*

$$\mathbf{P}\left[\gamma(H, IV) = 0^m \mid H \xleftarrow{\$} \mathbb{F}_{2^m}\right] \leq (\ell_{IV} + m)/2^m$$

for all IV such that len$(IV) \leq \ell_{IV}$.

Proof When $\text{len}(IV) = m - 32$, the considered probability is equal to zero. Otherwise, the condition that $\gamma(H, IV) = 0^m$ can be re-expressed as $R(T) = 0 \mod \chi_H(T)$, where the polynomial $R(T) = \sum_{i=0}^{\ell_{IV}+m-1} r_i T^i$ has its coefficients in $\mathbb{F}_2$ which are as follows

$$r_i = \begin{cases} 0 & \text{for } i = 0, \ldots, m/2 - 1 \\ (\text{len}(IV))_i & \text{for } i = m/2, \ldots, m - 1 \\ (IV)_{\text{len}(IV)-i+m} & \text{for } i = m, \ldots, \text{len}(IV) + m - 1 \end{cases}$$

Note that the probability that $R(T) = 0 \mod \chi_H$ is smaller that the probability that $R(H) = 0$ in $\mathbb{F}_{2^m}$. Besides, there are no more than $\ell_{IV} + m$ values of H that satisfy $R(H) = 0$. Because H is chosen at random, the probability that $R(H) = 0$ is no more than $(\ell_{IV} + m)2^{-m}$.

Appendix C. Security theorems

In this section we present the security results for the cipher mode MGCM presented in Appendix A

C.1 Security model for the block cipher function E

We follow the same outline for the security proof of the original GCM involving GHASH (cf. [9]). The security stands on a single cryptographic conjecture: the block cipher E is assumed to be a secure pseudorandom permutation (PRP). In other words the advantage A_E, defined below, of a distinguisher for E compared to a PRP is assumed to be small.

Definition 3 (*Advantage A_E*) At the beginning of the attack, the oracle chose to implement either the function E (event B_E) or a pseudorandom permutation (event B_E^c), each with a probability 1/2. For the implementation of E the oracle chose randomly K and then applies $E(K, \cdot)$ at each query. For the pseudoramdom permutation, at each new query the oracle chose randomly an encrypted block which had not been already been an output.

During the attack, the adversary performs q queries and then outputs a bit D which states the function used by the oracle for the cipher. The advantage is the difference between the event when D correctly guesses that E is used $[D|B_E]$ and the event when D wrongly guesses that E is used $[D|B_E^c]$. The advantage of the adversary is the following

$$A_E = \mathbf{P}[D|B_E] - \mathbf{P}[D|B_E^c].$$

Definition 4 (*Advantage A_{PRF}*) At the beginning of the attack, the oracle chooses to implement either the function E (event B_E) or a pseudorandom function (event B_E^c), each with a probability 1/2. For the implementation of E the oracle chooses randomly K and then applies $E(K, \cdot)$ at each query. For the pseudoramdom function, at each new query it chooses randomly the encrypted block.

During the attack, the adversary then performs q queries and then outputs a bit D which states the function used by the oracle for the cipher. The advantage is the difference between

the probability of the event when D correctly guesses that E is used $[D|B_E]$ and the event when D wrongly guesses that E is used $[D|B_E^c]$. The advantage of the adversary is the following

$$A_{\text{PRF}} = \mathbf{P}[D|B_E] - \mathbf{P}[D|B_E^c].$$

In [15] we can find the following lemma which states that the advantages A_E and A_{PRF} differ in a quadratic term in q, the number of queries. Consequently, in general, the security proof is done in the PRF context and then are rationalized using Lemma 7 to obtain a PRP security proof.

Lemma 7 (A PRP can be a good PRF) *The advantage A_{PRF} of an adversary in distinguishing an m-bit PRP E from a random function is bounded by*

$$A_{\text{PRF}} \leq A_E + q(q-1)2^{-(m+1)},$$

where A_E is the adversary's advantage in distinguishing E from a random permutation and the value q is the number of queries to the function oracle.

C.2 Authentication security

We are now in a position to state some security results concerning the message authentication code of MGCM. Specifically, we would like to link the problem to forge a correct MGCM tag to the problem to distinguish E from a random permutation. We use the standard security model for this purpose, in the presence of a chosen message attack. The adversary is given access to a tag generation oracle and a message/tag verification oracle. The adversary can pass messages to the tag generation oracle and construct any message/tag pairs that he wants and sends these to the verification oracle. Queries to the oracles can be interleaved by the adversary if desired.

The forgery advantage F_{MGCM} is the probability that the adversary can get the verification oracle to accept a message/tag pair other that one generated by the tag generation oracle after making q queries to the oracle.

Theorem 1 (MGCM authentication is secure) *An adversary with forgery advantage F_{MGCM} against MGCM with q queries to oracles where packets are of length $\leq \ell$ and IV are of length $\leq \ell_{IV}$ and the full bit length of the queried data is no larger than ℓ_P has a distinguishing advantage A_E against E which satisfies*

$$A_E \geq F_{\text{MGCM}} - (\ell_P/m + 2q)^2 2^{-m-1} - q((\ell_P/m+2q) \times(\ell_{IV}+m)2^{1-m}(\ell_{IV}+m+1)2^{-m} + (\ell+m)2^{-s}).$$

Proof To prove this theorem we just have to follow the proof of Theorem 2 of [9] and then replace the different uses of Lemmas 2, 3 and 4 of [9], by the corresponding lemmas of Appendix B.

C.3 Encryption security

Here, we focus on the security of the encryption of MGCM. We will express the advantage of the distinguisher for MGCM compared to a PRP in terms of the advantage A_E of the block cipher E used in MGCM. We follow the same approach of McGrew and Viega [9]. In this model we use two oracles:

- The *authenticated encryption oracle* models the GCM authenticated encryption operation. It takes as input the bit strings IV, A and P and returns the bit strings C and τ (the ciphertext and the authentication tag).
- The *authenticated decryption oracle* accepts input of the type (IV, A, C, τ) and returns as either the special symbol **FAIL**, which suggest that the tag τ is not correct, or the plaintext P.

The adversary is free to choose the IV he wants under the condition it is nonce-respecting and will not submit the same IV to the same oracle multiple times (but he can submit the same to the two oracles). The adversary is free to perform the queries to the oracle in the order it wants and interleaves the queries.

The adversary makes different queries to the oracles during the attack. Specifically, the oracle randomly chooses (after flipping a coin at the beginning of the attack) to implement a correct MGCM fixing randomly the key K and using the block cipher E and MGHASH to cipher and decipher MGCM-packet. We will denote B_{MGCM} as the latter event. It also choses to implement MGCM through a PRF function which generates cipher C and tag τ randomly. We denote this event by B_{MGCM}^c.

At the end of the attack (after q queries), the adversary outputs D the bit which is equal to one if the adversary guesses that the oracle used MGCM to forge the MGCM-packets, and D is equal to 0 if the adversary assumes that the packets were generated randomly. The advantage A_{MGCM} of the adversary is the following

$$A_{\text{MGCM}} = \mathbf{P}[D|B_{\text{MGCM}}] - \mathbf{P}[D|B_{\text{MGCM}}^c].$$

Theorem 2 *If there is an adversary that can distinguish MGCM*
encryption from a random function with advantage A_{GCM} when the output of that function is limited to q queries to the authenticated encryption and decryption oracles, where $len(C) + len(A) \leq \ell$ and $len(IV) \leq \ell_{IV}$ for each query. Then the adversary can distinguish E from a random permutation with an advantage A_E where

$$A_E \geq A_{\text{MGCM}} - (\ell_P/m + 2q)^2 2^{-m-1} - (q\ell/m + 1)2^{-t} - q(\ell_P/m + 2q)(\ell_{IV} + m)2^{1-m}$$

Proof To prove this theorem we just have to follow the proof of Theorem 1 of [9] and then replace the different uses of

Lemmas 2, 3 and 4 of [9], by the corresponding lemmas of Appendix B. □

References

1. Bajard, J.-C., Imbert, L., Jullien, G.A.: Parallel Montgomery multiplication in GF(2^k) using trinomial residue arithmetic. In: Proceedings of 17th IEEE Symposium on Computer Arithmetic (ARITH), pp. 164–171 (2005)
2. Bulens, P., Standaert, F.-X., Quisquater, J.-J., Pellegrin, P., Rouvroy, G.: Implementation of the AES-128 on Virtex-5 FPGAs. In: Progress in Cryptology—AFRICACRYPT. LNCS, vol. 5023, pp. 16–26. Springer, Berlin (2008)
3. Fan, H., Hasan, M.A.: A new approach to subquadratic space complexity parallel multipliers for extended binary fields. IEEE Trans. Comput. **56**(2), 224–233 (2007)
4. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: Cryptographic Hardware and Embedded Systems—CHES. LNCS, vol. 3659, pp. 427–440. Springer, Berlin (2005)
5. Gordon, J.A.: Very simple method to find the minimum polynomial of an arbitrary nonzero element of a finite field. Electron. Lett. **12**(25), 663–664 (1976)
6. Jarvinen, K.U., Tommiska, M.T., Skyttae, J.O.: A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In: International symposium on Field programmable gate arrays—FPGA, pp. 207–215. ACM, New York (2003)
7. Lemsitzer, S., Wolkerstorfer, J., Felber, N., Braendli, M.: Multi-Gigabit GCM-AES Architecture Optimized for FPGAs. In: Cryptographic Hardware and Embedded Systems—CHES, vol. 4727, pp. 227–238. Springer, Berlin (2007)
8. Mastrovito, E.D.: VLSI Architectures for Computation in Galois Fields. PhD thesis, Dept. of Electrical Eng., Link ping Univ., Sweden (1991)
9. McGrew D.A., Viega J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: INDOCRYPT. LNCS, vol. 3348, pp. 343–355 (2004)
10. McGrew, D.A., Viega, J.: The Galois/Counter Mode of Operation (GCM) (2005)
11. Meloni, N., Negre, C., Hasan, M.A.: High performance GHASH function for Galois/Counter Mode. In: Applied Cryptography and Network Security (ACNS), Beijing, China. LNCS, vol. 6123 (2010)
12. NIST: Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC (2007)
13. Paar, C.: A new architecture for a parallel finite field multiplier with low complexity based on composite fields. IEEE Trans. Comput. **45**(7), 856–861 (1996)
14. Patel, P.: Parallel multiplier designs for the Galois/counter mode of operation. Master's thesis, Electrical and Computer Engineering, University of Waterloo (2008)
15. Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 98–107 (2002)
16. Sarkar, P.: Efficient tweakable enciphering schemes from (block-wise) universal hash functions. IEEE Trans. Inform. Theory **55**(10), 4749–4760 (2009)
17. Satoh, A.: High-speed hardware architectures for authenticated encryption mode GCM. In: IEEE International Symposium on Circuits and Systems—ISCAS, pp. 4831–4834 (2006)
18. Satoh, A.: High-Speed Parallel Hardware Architecture for Galois Counter Mode. In: IEEE International Symposium on Circuits and Systems—ISCAS, pp. 1863–1866 (2007)
19. Satoh, A., Sugawara, T., Aoki, T.: High-Speed Pipelined Hardware Architecture for Galois Counter Mode. In: 10th International Conference—ISC. LNCS, vol. 4779, pp. 1863–1866. Springer, Berlin (2007)
20. Standaert, F.X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D: Efficient implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs. In: Cryptographic Hardware and Embedded Systems—CHES. LNCS, vol. 2779, pp. 334–350. Springer, Berlin (2003)