High Performance GHASH Function for Long Messages

Nicolas Méloni[1], Christophe Négre[2] and M. Anwar Hasan[1]

[1] Department of Electrical and Computer Engineering
University of Waterloo, Canada
[2] Team DALI/ELIAUS
University of Perpignan, France

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 effectively computes $X_1H^n + X_2H^{n-1} + \dots X_nH$, where H is an element of the binary field $\mathbb{F}_{2^{128}}$. This operation is usually computed by using n successive multiply-add operations over $\mathbb{F}_{2^{128}}$. In this work, we propose a method to replace all but a fixed number of those multiplications by additions on the field. This is achieved by 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.

Keywords: Galois/Counter mode, GHASH function, characteristic polynomial.

1 Introduction

The Galois/Counter mode (GCM) is one of the modes of operation recommended by the National Institute of Standards and Technology (NIST) [10]. As an authenticated encryption mode, it generates both a cipher text and a authentication tag for each session. The cipher text is obtained by using the counter mode of encryption (CRT) with a 128 bit block cipher algorithm (usually AES). The authentication part is performed by using a universal hashing (GHASH) based on multiplication in the binary field $\mathbb{F}_{2^{128}}$ [9]. More precisely, if $X = X_1 \dots X_n$ is a bit stream divided into 128 bit blocks, the authentication process computes $X_1H^n + X_2H^{n-1} + \dots + X_nH$, 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, 16] and bit-parallel multiplier over $\mathbb{F}_{2^{128}}$ [1, 3, 8, 11]. One can refer to [7, 13–15] for various implementations of the AES-GCM mode of encryption.

In practice, the $\mathbb{F}_{2^{128}}$ multiplier operates faster than the block cipher. However, it is easy to speed up the block cipher computation by taking advantage of the natural parallelism of the CRT mode. In that case, the $\mathbb{F}_{2^{128}}$ multiplier's critical path of the GHASH function becomes the bottleneck. To overcome this problem, it is always possible to parallelize the GHASH computation process as suggested in [9]. Such solutions imply a proportional increase in the number of multipliers.

In this paper, we propose a new way to speed up the GHASH function for long messages, by moving the bottleneck from the multiplier to that of one XOR and one AND gate. This is achieved by first computing the characteristic polynomial of H and then performing the computation of the GHASH function modulo this polynomial. Effectively, we replace the finite field multiplication by a faster operation. In addition, our method can be parallelized to further reduce the GHASH computation time.

The remainder of this paper is organized as follow: Section 2 briefly describes the functioning of GHASH and the implementation of the binary field $\mathbb{F}_{2^{128}}$. In section 3, we describe our approach

to use characteristic polynomials to compute GHASH and a parallelization of this operation using multiple polynomial reduction units. In section 4, we present how to efficiently compute a characteristic polynomial over $\mathbb{F}_{2^{128}}$. Finally, a few concluding remarks are made in section 5.

2 GHASH authentication function

GCM uses two main operations: encryption and authentication. Encryption is performed using a block cipher encryption function whereas the authentication mainly requires the use of multiplication over $\mathbb{F}_{2^{128}}$. One can find a full description of the encryption process in [9]. Below we only describe the authentication part.

2.1 Authentication function

Let $X = X_1X_2 \dots X_n$ be a bit string divided into 128 bit blocks (X_n might be padded with 0's) and H the 128 bit hash sub-key. The GHASH function performs as follow:

Algorithm 1 GHASH function

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

One can verify that Algorithm 1 computes $X_1H^n + X_2H^{n-1} + \cdots + X_nH$. In practice, X is obtained as the concatenation of the cipher text, the authentication data and the length of those two data, and H is generated by applying the block cipher to the zero block. Note that Algorithm 1 requires n multiplications and n additions on the field $\mathbb{F}_{2^{128}}$. Assuming that we have only one multiplier and one adder, these operations would be performed in sequence and the total computation time for GHASH is approximately n times the combined delay of a multiplication and an addition over the field. On binary fields, the delay of an addition is exactly that of a bitwise XOR operation. In Table 1, we summarize various bit-parallel multiplication methods and their space and time complexities in terms of gate counts and gate delays.

Multiplier	#AND	#XOR	Gate delay (D_M)
CRT [1]	$O(m^{1.6})$	$O(m^{1.6})$	$O(m)D_X + 4m^{0.4}D_A$
Karatsuba [11]	$O(m^{1.58})$	$O(m^{1.58})$	$(3\log_2(m) + 1)D_X + D_A$
Fan-Hasan [3]	$O(m^{1.58})$	$O(m^{1.58})$	$(2\log_2(m) + 1)D_X + D_A$
Mastrovito [8]	$O(m^2)$	$O(m^2)$	$\log_2(m)D_X + D_A$

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.

2.2 Underlying computations

In GCM specifications, the binary field $\mathbb{F}_{2^{128}}$ is defined as $\mathbb{F}_2[x]/(x^{128}+x^7+x^2+x+1)$. That is to say, the set of residues of polynomials with coefficients in $\mathbb{F}_2$ modulo $x^{128}+x^7+x^2+x+1$. An element A of this field can be easily represented as a vector of length 128 with coefficient in $\mathbb{F}_2$.

In that case, addition is very simple as it just consists of a bitwise XOR between the two summands. Multiplication can be performed by a succession of shifts and additions. Depending on the platform on which GCM is implemented, it is possible to improve this operation, as shown in [9] or [3] for instance.

One common feature among several of the previously reported methods for improving GCM is that all efforts are made to optimize the field multiplication only. In this work, we propose a way to improve the efficiency of the overall computation of $\text{GHASH}_H(X) = X_1H^n + X_2H^{n-1} + \cdots + X_nH$.

3 Computing GHASH using a characteristic polynomial

In order to easily present our method, here we consider a bit string, divided into n blocks: $X_1X_2 \ldots X_n$ and an element H of $\mathbb{F}_{2^{128}}$. Our goal is to compute $\text{GHASH}_H(X) = X_1H^n + X_2H^{n-1} + \cdots + X_nH$. As $H \in \mathbb{F}_{2^{128}}$ there exists a polynomial χ_H of degree 128 (at most), with coefficients in $\mathbb{F}_2$, such that $\chi_H(H) = 0$, called the characteristic polynomial of H in $\mathbb{F}_2$. This section shows how to use such a polynomial to limit the number of field multiplications involved in the polynomial evaluation.

3.1 Using χ_H

Let $H \in \mathbb{F}_{2^m}$ and $\chi_H(T) = \Pi_{i=0}^{m-1}(T + H^{2^i})$ be a polynomial in T of degree m. Polynomial $\chi_H(T)$ is called the characteristic polynomial of H and satisfies $\chi_H(H) = 0$ and has its coefficients in $\mathbb{F}_2$.

Hence $P = X_1H^n + X_2H^{n-1} + \cdots + X_nH = X_1H^n + X_2H^{n-1} + \cdots + X_nH \mod \chi_H$ is considered as a polynomial in H with coefficients in $\mathbb{F}_{2^m}$. Let us now write χ_H as $\sum_{i=0}^{m} c_iT^i$. It can be shown that $c_m = c_0 = 1$. Using the fact that $H^m = \sum_{i=0}^{m-1} c_iT^i$, it is possible to reduce a polynomial in H of degree n to a polynomial of degree $m-1$. As an example, let $P = X_1H^{128} + X_2H^{127} + \cdots + X_{128}H$ with $X_i \in \mathbb{F}_{2^m}$, where $m = 128$. Then $P = (X_2 + c_{127}X_1)H^{127} + (X_3 + c_{126}X_1)H^{126} + \cdots + (X_{128} + c_1X_1)H + c_0X_1$. It is important to note that all the c_i's are in $\mathbb{F}_2$, which means that computing $(X_{m-i+1} + c_iX_i)$ is just an addition over $\mathbb{F}_{2^{128}}$ or a simple parameter assignment. Moreover, all these additions are independent and can all be performed in parallel. Figure 1 shows a diagram of a possible implementation of this operation in hardware. The output of the polynomial reduction unit (PRU) is a polynomial of degree $m-1$ in H. We can take the output of the PRU to an multiply-and-add unit to apply the Horner scheme and eventually obtain the desired GHASH value, which is an element of $\mathbb{F}_{2^{128}}$ (see Figure 2).

A generalized description of the operation of the structure of Figure 2 is given in Algorithm 2, where a polynomial P of degree $n \geq m$ can be computed using at most $m-1$ field multiplications, replacing each of the remaining $n-m$ multiplications by a maximum of m parallel field additions.

Algorithm 2 computes P in two steps:

- first it computes the remainder of P modulo χ_H using a shift-and-add algorithm,
- then it effectively calculates P using the Horner scheme.

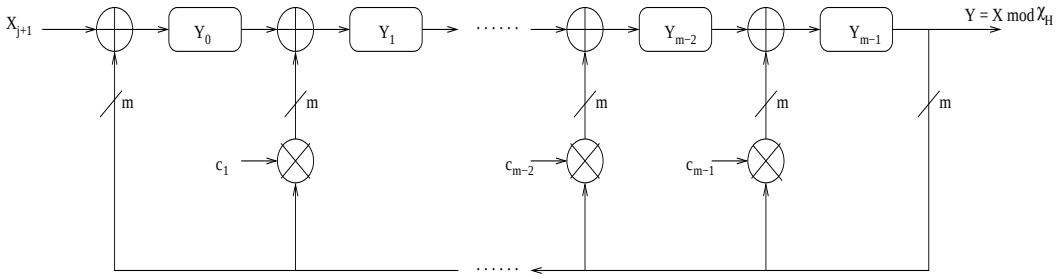

Fig. 1. Implementation of the Polynomial Reduction Unit (PRU).

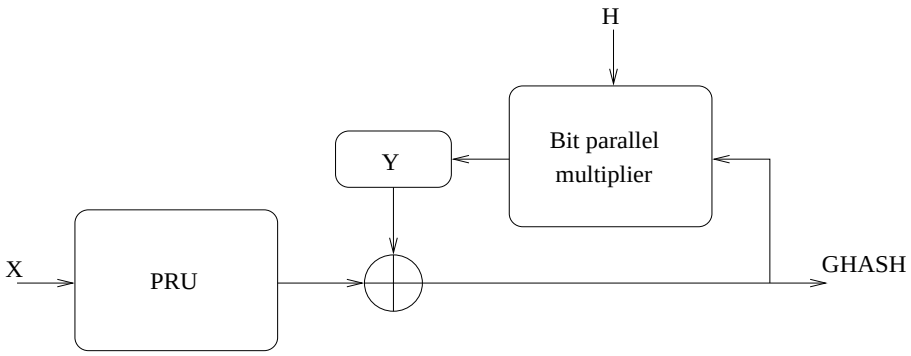

Fig. 2. Implementation of the GHASH function using a PRU.

Algorithm 2 $GHASH_H(X)$ using χ_H

Require: $X = X_1X_2 \dots X_n$ and $\chi_H(T) = \sum_{i=0}^{m} c_i T^i$
Ensure: $GHASH_H(X) = X_1H^n + X_2H^{n-1} + \cdots + X_nH$
  $Y_{m-1} \dots Y_0 \leftarrow X_1 \dots X_m$, $GHASH = 0$
  for $j = m$ to n **do**
    $C = Y_{m-1}$
    $Y_i \leftarrow Y_{i-1} + c_iC, 1 \le i \le m-1$
    $Y_0 = X_{j+1} + c_0C$
    } (in parallel)
  end for
  for $i = m-1$ down to 1 **do**
    $GHASH = (GHASH + Y_i) \times H$
  end for
  return $(GHASH + Y_0)$

Complexity: 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 1 MBytes, then $n = 2^{16}$, and in Algorithm 2, the upper *for* loop will dominate in terms of computational cost. As noted previously, one important feature of Algorithm 2 is that all the additions involved in the upper *for* loop are independent. Moreover, the c_i's are elements of $\mathbb{F}_2$, which means that the operation $Y_{i-1} + c_i C$ is just a bitwise XOR operation. In the end, each step of this loop requires m bitwise XOR operations. Thus, our method trades $n - m$ multiplications by H over $\mathbb{F}_{2^m}$ against $m(n-m)(W_H - 1)$ bit level XOR operations, where W_H is the number of non-zero coefficients of χ_H.

In the case of a hardware implementation, on a parallel architecture, a step of the loop would require m^2 XOR gates, all used in parallel, and the gate delay is $D_X + D_A$ (delay of one XOR and one AND gates). In the end, the implementation of the GHASH function as shown in Figure 2 has a space complexity of $O(m^2)$ and the total time delay for the upper loop is $(n-m)(D_X + D_A)$. The final $m - 1$ field multiplications can be performed using any of the already available methods as listed in Table 1.

Comparison: Let n be the length of the message to be treated (considered as a sequence of m bit blocks). Implementations based on Algorithm 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, the total delay is:

$$n(D_M + D_X).$$

On the other hand, Algorithm 2 first requires the computation of the characteristic polynomial. We denote the cost of this computation as D_χ. It is important to remark that this cost is constant, i.e. independent of n. Then, Algorithm 2 performs m^2 parallel bitwise XOR operations in each pass of the upper loop. Finally, the algorithm ends by computing $(m-1)$ field multiplications and m bitwise XOR operations on m bit blocks. Assuming that the m Y-registers of the PRU can be initialized in parallel, the total delay can then be approximated as:

$$D_\chi + (n-m)(D_X + D_A) + (m-1)(D_M + D_X).$$

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.

3.2 Delay reduction with multiple PRUs

Let $n = 2n'$ be an even integer (in order to simplify notations). Then one can write

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

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 (1), 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 zeros to X if needed). Then we can decompose X into r different sets $P_1, P_2, \ldots, P_r$ such that

$$P = P_1 H^r + P_2 H^{r-1} + \cdots + P_r H. \tag{2}$$

where for $1 \leq i \leq r$ 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 $P'_i = P_i \mod \chi_{H^r}$. Using r PRUs operating concurrently, we can reduce $P_i \mod \chi_{H^r}$ to obtain the corresponding P'_i, $\forall i$, in $\frac{n}{r}$ iterations as shown in the left part of Fig. 3. (The number of iterations reduces to $\frac{n}{r} - m$ if each PRU can be initialized with m coefficients in parallel at the beginning of its operation.) We now write (2) in terms of P'_i as follows:

$$P \equiv P'_1 H^r + P'_2 H^{r-1} + \cdots + P'_r H. \tag{3}$$

Since each P'_i is a degree $m-1$ polynomial in H^r, one can verify that the sum of the products in (3) 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 reduce the polynomial in H from degree $r(m-1)$ to degree $m-1$. 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 and hence the latter does not need to be computed a second time. 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 (see Fig. 3), the GHASH computation of X has the following time delay

$$\left(\frac{n}{r} + rm + 1\right)(D_X + D_A) + (m-1)(D_M + D_X). \tag{4}$$

Comparison: As noted in [9], the conventional method of computing GHASH can also be parallelized. Assume that we have r field multipliers. Then all P_i, for $1 \leq i \leq r$, can be computed in $\frac{n}{r} - 1$ iterations, assuming that we have $H, H^2, \ldots, H^r$, and the time delay of each iteration is $D_M + D_X$. With one additional delay of D_M, one can compute $P_i H^{r-i+1}$, $1 \leq i \leq r$. Finally P is obtained as $\sum_{i=1}^{r} P_i H^{r-i+1}$, which incurs a delay of $(\log_2 r) D_X$, assuming additions are done in parallel in a binary tree fashion. Thus, given $H^i, 1 \leq i \leq r$, using r field multipliers and $r-1$ adders, one can compute GHASH based on Algorithm 1 with the following time delay:

$$\left(\frac{n}{r} - 1\right)(D_M + D_X) + D_M + (\log_2 r) D_X \tag{5}$$

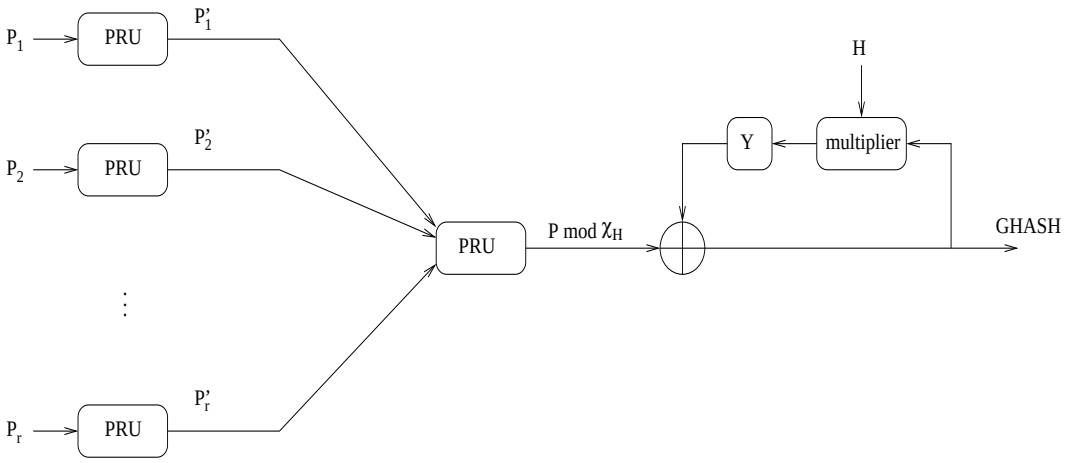

Fig. 3. Implementation of the GHASH function using multiple PRUs.

In hardware implementation, D_X is normally larger than D_A, and more importantly, for bit parallel implementation we have $D_M \geq (\log_2 m)D_X$. For example, as mentioned in Table 1, the Karatsuba algorithm based bit-parallel multiplier over $\mathbb{F}_{2^{128}}$ has $D_M > 7D_X$. Thus for a large n (i.e., long messages), it is clear that (4) is considerably smaller than (5).

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

In this section, 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, 127 multiplications and 127 squarings. We then propose a new method taking advantage of the tower structure of $\mathbb{F}_{2^{128}}$ that can be faster than the first one depending on the representation of the finite field.

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

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

We want to find the polynomial in the form $\sum_{i=0}^{m} B_i T^i$, with $B_i \in \mathbb{F}_2$. Gordon's method is based on the observation that $\chi_A(x) = \prod_{i=0}^{m-1} (x + A^{2^i}) = \sum_{i=0}^{m} B_i x^i$. In other words, the coefficients of the representation of field element $\chi_A(x)$ correspond to those of the polynomial $\chi_A(T) \mod \chi_x(T)$. As $\chi_A(T)$ has degree 128, we have $\chi_A(T) = (\chi_A(T) \mod \chi_x(T)) + \chi_x(T)$. This can be easily evaluated by adding $\chi_A(x)$ to $\chi_x(x)$ modulo 2. Evaluating $\chi_A(x)$ can be done using 127 field multiplications and 127 fields squaring.

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 $P = \sum_{i=0}^{d} p_i T^i$ be a polynomial in $\mathbb{F}_{2^n}[T]$ and k an

integer. We denote

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

Lemma 1. *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 $P, Q \in \mathbb{F}_{2^m}[T]$ and k an integer then*

$$\sigma_k(PQ) = \sigma_k(P)\sigma_k(Q).$$

2. *If $P \in \mathbb{F}_{2^{2m}}[T]$ satisfies $P(A) = 0$, then the polynomial $Q = P\sigma_m(P)$ satisfies*

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

Proof. 1. Let us write $P = \sum_{i=0}^{d} p_i T^i$ and $Q = \sum_{i=0}^{d'} q_i T^i$ then

$$PQ = \sum_{i=0}^{d} \sum_{j=0}^{d'} p_i q_i T^{i+j}.$$

Then, we apply σ_k to PQ

$$\begin{aligned}
\sigma_k(PQ) &= \sigma_k\left(\textstyle\sum_{i=0}^{d} \sum_{j=0}^{d'} p_i q_i T^{i+j}\right) \\
&= \textstyle\sum_{i=0}^{d} \sum_{j=0}^{d'} (p_i q_i)^{2^k} T^{i+j} \\
&= \textstyle\sum_{i=0}^{d} \sum_{j=0}^{d'} p_i^{2^k} q_i^{2^k} T^{i+j} \\
&= \sigma_k(P)\sigma_k(Q)
\end{aligned}$$

which proves the assertion.

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

$$Q(A) = P(A)\sigma_q(P)(A) = 0 \times \sigma_q(P)(A) = 0$$

In order to show that $Q \in \mathbb{F}_{2^m}[T]$ we have to prove that $\sigma_m(Q) = Q$, since this would mean that each coefficient of Q is in $\mathbb{F}_{2^m}$. We compute

$$\sigma_m(Q) = \sigma_m(P)\sigma_m(\sigma_m(P))$$

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

$$\sigma_m(Q) = P\sigma_m(P) = Q$$

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 $P_0 = T + A$ which satisfies $P(A) = 0$ and $P \in \mathbb{F}_{2^8}[T]$.

– Now we apply the lemma for the field extension $\mathbb{F}_{2^8}/\mathbb{F}_{2^4}$ to obtain a polynomial $P_1 = P_0\sigma_4(P_0)$ which satisfies

$$P_1(A) = 0 \text{ and } P_1 \in \mathbb{F}_{2^4}[T]$$

Moreover we have $P_1 = (T + A)(T + A^{2^4})$.

– We apply again the lemma for the field extension $\mathbb{F}_{2^4}/\mathbb{F}_{2^2}$ to obtain a polynomial $P_2 = P_1\sigma_2(P_1)$ which satisfies

$$P_1(A) = 0 \text{ and } P_1 \in \mathbb{F}_{2^2}[T].$$

Moreover we also have $P_2 = (T + A)(T + A^{2^4})(T + A^{2^2})(T + A^{2^6})$.

– Finally we apply the lemma a third time for the field extension $\mathbb{F}_{2^2}/\mathbb{F}_2$ to the polynomial P_2. We get

$$\begin{aligned} P_3 &= P_2\sigma_1(P_2) \\ &= \underbrace{(T + A)(T + A^{2^4})(T + A^{2^2})(T + A^{2^6})}_{P_2} \\ &\quad \times \underbrace{(T + A^2)(T + A^{2^5})(T + A^{2^3})(T + A^{2^7})}_{\sigma_2(P_2)}, \end{aligned}$$

and $P_3 \in \mathbb{F}_2[T]$ and satisfies $P_3(A) = 0$. If we look at the expression of P_3 we can see that it is equal to the characteristic polynomial of A.

This method is generalized for binary field $\mathbb{F}_{2^m}$ where $m = 2^k$ is a power of 2 in Algorithm 3.

Algorithm 3 Computing the characteristic polynomial of A

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

Proposition 1. *Algorithm 3 returns the characteristic polynomial of A over $\mathbb{F}_2$.*

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

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

Specifically, we will show by induction on the index of the loop that the computed P in the jth loop is the characteristic polynomial of A over the field $\mathbb{F}_{2^{k-j}}$.

– For $j = 0$ this is clear that $P = T + A$ is the characteristic polynomial of A over $\mathbb{F}_{2^k}$.
– For $j = 1$ we have $P = (T + A)(T + A^{2^{k-1}})$, thus the assertion is true.

– Suppose that $P = \prod_{i=0}^{j-1}(T + A^{d^i})$ where $d = 2^{k-j}$, then

$$P\sigma_{j+1}(P) = \left(\prod_{i=0}^{j-1}(T + A^{d^{2i}})\right)\left(\prod_{i=0}^{j-1}(T + A^{d^{2i+1}})\right)$$

where $d = 2^{k-(j+1)}$. This completes the proof.

Complexity.

Let us now evaluate the complexity of Algorithm 3. We suppose that each polynomial multiplication is done using the Karatsuba method, and each exponentiation to 2^k is done by computing k successive squares.

First, we make two observations:

– Let us first check that the degree of P after j iterations of the *for* loop has degree 2^j. This is true when $j = 0$ since P 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 j-th iteration of the *for* loop, the polynomial P belongs to $\mathbb{F}_{2^{2^{k-j}}}$. Indeed, for $j = 1$, $P = (T+A)\sigma_{2^{2^{k-1}}}(T+A)$ with $A \in \mathbb{F}_{2^{2^k}}$, so $P \in \mathbb{F}_{2^{2^{k-1}}}[T]$ from Lemma 1. If we suppose that after j iterations $P \in \mathbb{F}_{2^{2^{k-j}}}[T]$, then, at step $j+1$ we compute $P = P \times \sigma_{2^{2^{k-(j+1)}}}(P)$, which belongs to $P \in \mathbb{F}_{2^{2^{k-(j+1)}}}[T]$.

From the above two remarks, we can now evaluate the complexity of Algorithm 3.

– At the j-th iteration of the *for* loop, P 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 $\sigma_{2^{k-j}}(P)$. 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 $\sigma_{2^{k-j}}(P)$ 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 j-th iteration, after computing $\sigma_{2^{k-j}}(P)$, the algorithm computes $P\sigma_{2^{k-j}}(P)$. 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

$$\begin{aligned}\sum_{j=1}^{k} 3^j M_{2^{k-j+1}} &= \sum_{j=1}^{k} 3^j \frac{M_{2^k}}{3^{j-1}} \\ &= \sum_{j=1}^{k} 3M_{2^k} \\ &= 3kM_{2^k}\end{aligned}$$

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 $\mathbb{F}_{2^{2^k}}$ is $3kM_{2^k} + 2^{k+1}S_{2^k}$.

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 1. It is not always suitable to decompose the field $\mathbb{F}_{2^{2^k}}$ into k extensions of degree 2. As a 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:

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

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^{31} 128 bit blocks), this additional cost can be considered as negligible for long sessions.

Remark 2. 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 [12], 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.

5 Conclusions

In this paper we have proposed a new way to improve the performance of the GHASH function of GCM. Our method is based on the use of the characteristic polynomial of the authentication data. It has allowed use to trade most of the field multiplications involved during the authentication tag computation by a series of 128 independent fields additions. This is very attractive for high

performance implementations where all such additions can be performed in parallel. This allows us to reduce the delay of each of the first $n - 128$ multiplications over $\mathbb{F}_{2^{128}}$ to that of one XOR and one AND operation. To illustrate the effectiveness of the proposed method, we have considered $n = 10,000$ and the Karatsuba algorithm based bit parallel multiplier on FPGA from [12], 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.

In this paper, we have also shown the flexibility of our method in terms of parallelization. Using multiple polynomial reduction units 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.

References

1. J.-C. Bajard, L. Imbert, and G. A. Jullien. Parallel Montgomery multiplication in GF(2^k) using trinomial residue arithmetic. In *Proc. 17th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 164–171, 2005.
2. P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy. Implementation of the AES-128 on virtex-5 FPGAs. In *Progress in Cryptology - AFRICACRYPT*, volume 5023 of *LNCS*, pages 16–26. Springer, 2008.
3. H. Fan and M.A. Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Transactions on Computers*, 56(2):224–233, 2007.
4. T. Good and M. Benaissa. AES on FPGA from the fastest to the smallest. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 3659 of *LNCS*, pages 427–440. Springer, 2005.
5. J. A. Gordon. Very simple method to find the minimum polynomial of an arbitrary nonzero element of a finite field. *Electronics Letters*, 12(25):663–664, December 1976.
6. K. U. Jarvinen, M. T. Tommiska, and J. O. Skyttae. A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In *International symposium on Field programmable gate arrays - FPGA*, pages 207–215. ACM, 2003.
7. S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli. Multi-gigabit GCM-AES architecture optimized for FPGAs. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 4727, pages 227–238. Springer, 2007.
8. E.D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Dept. of Electrical Eng., Link ping Univ., Sweden, 1991.
9. D. A. McGrew and J. Viega. *The Galois/Counter Mode of Operation (GCM)*, 2005.
10. NIST. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, November 2007.
11. C. Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–861, July 1996.
12. P. Patel. Parallel multiplier designs for the Galois/counter mode of operation. Master's thesis, Electrical and Computer Engineering, University of Waterloo, 2008.
13. A. Satoh. High-speed hardware architectures for authenticated encryption mode gcm. In *IEEE International Symposium on Circuits and Systems - ISCAS*, pages 4831–4834, 2006.
14. A. Satoh. High-speed parallel hardware architecture for Galois counter mode. In *IEEE International Symposium on Circuits and Systems - ISCAS*, pages 1863–1866, 2007.
15. A. Satoh, T. Sugawara, and T. Aoki. High-speed pipelined hardware architecture for Galois counter mode. In *10th International Conference - ISC*, volume 4779 of *LNCS*, pages 1863–1866. Springer, 2007.
16. F. X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D Legat. Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 2779 of *LNCS*, pages 334–350. Springer, 2003.