Trade-Off Approach for GHASH Computation Based on a Block-Merging Strategy

Christophe Negre

Trade-Off Approach for GHASH Computation Based on a Block-Merging Strategy

Christophe Negre[1,2]

[1]Team DALI, Université de Perpignan, France

[2]LIRMM, Université de Montpellier and CNRS, Montpellier, France

[1,2]christophe.negre@univ-perp.fr

Abstract: In the Galois counter mode (GCM) of encryption an authentication tag is computed with a sequence of multiplications and additions in $\mathbb{F}_{2^m}$. In this paper we focus on multiply-and-add architecture with a suquadratic space complexity multiplier in $\mathbb{F}_{2^m}$. We propose a recombination of the architecture of P. Patel [16] which is based on a subquadratic space complexity Toeplitz matrix vector product. We merge some blocks of the recombined architecture in order to reduce the critical path delay. We obtain an architecture with a subquadratic space complexity of $O(\log_2(m)m^{\log_2(m)})$ and a reduced delay of $(1.59\log_2(m) + \log_2(\delta))D_X + D_A$ where δ is a small constant. To the best of our knowledge, this is the first multiply-and-add architecture with subquadratic space complexity and delay smaller than $2\log_2(m)D_X$.

Keywords: GHASH, Galois counter mode, multiply-and-add architecture, binary field multiplier, subquadratic space complexity

1. Introduction

One extensively used cryptographic operation is the encryption with a secret key. The Galois counter mode (GCM) [11] is a mode of encryption which encrypts a message of any length and

produces an authentication tag. It proceeds the encryption with the counter mode, which is a mode which can be parallelized. The authentication is done with the GHASH function which consists in a sequence of multiplications and additions in a field $\mathbb{F}_{2^m}$. This sequence of multiplications and additions can be parallelized [18]. This leads to a high-level of parallelization which renders GCM really attractive for high throughput encrypted communication. The encryption mode GCM used with the block-cipher AES [3, 13] was published as a standard by the NIST in 2007 [14].

Efficient implementation of GCM is based first on a parallelization and a pipelined implementation of the block-cipher [18, 9, 17, 1]. The multiplier in $\mathbb{F}_{2^m}$ used in the computation of the authentication tag have to be efficiently implemented. The most efficient multipliers in $\mathbb{F}_{2^m}$ are the parallel multipliers which compute a product in one clock cycle and have a small critical path delay. Such parallel multipliers can either have a quadratic space complexity and a delay of $(\log_2(m) + O(1))D_X + D_A$ (cf. [10, 6, 2, 21]) or have a subquadratic space complexity multiplier and a delay of $(2\log_2(m) + O(1))D_X + D_A$ [15, 20, 12, 4, 5, 7] (here D_X is the delay of an XOR gate and D_A is the delay of an AND gate and m is a power of 2). Subquadratic space complexity multipliers can be either based on subquadratic complexity formula for polynomial multiplication [15, 20, 12] or on subquadratic complexity formula for Toeplitz-matrix vector product [4, 5, 7].

In this paper we instigate a new strategy for a subquadratic space complexity architecture which has an improved delay. Our approach consists of the recombination of the different blocks of the multiply-and-add architecture based on a subquadratic TMVP multiplier in $\mathbb{F}_{2^m}$. This recombination enables us to merge some blocks in order to reduce the delay. We obtain a multiply-and-add architecture which has a subquadratic space complexity of $(\log_2(m) + O(1))m^{\log_2(3)} + O(m)$ gates and a delay of $(\log_2(m) + O(1))D_X + D_A$. To the best of our knowledge it is the first time a suquadratic space complexity multiply-and-add architecture has a delay smaller than $2\log_2(m)D_X + D_A$.

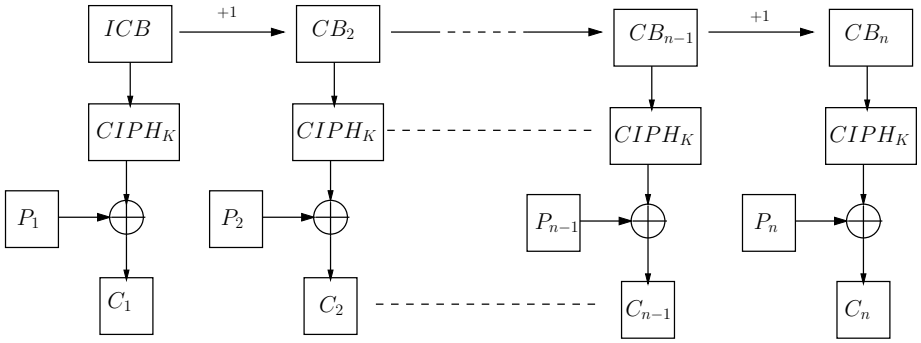
The remainder of this paper is organized as follows. In Section 2 we review GCM and GHASH algorithms. In Section 3 we review the suquadratic space complexity multiply-and-add architec-

ture of [16] which is based on a subquadratic space complexity TMVP multiplier. In Section 4 we present a recombination and some block-merging approaches in order to reduce the critical path-delay of the multiply-and-add architecture. In Section 5 we compare the complexity of the proposed architecture to the state of the art and we give some concluding remarks.

2. Galois Counter Mode and GHASH function

The GCM performs two main cryptographic operations: encryption/decryption and authentication. Encryption and decryption are done using a block cipher denoted CIPH of block size m which is operated in counter mode [11]. In the sequel we will assume that m is a power of 2. The counter that is used for encryption is initialized with an m-bit value called initial counter block (ICB). It is encrypted with the block-cipher $CIPH$ using a private key (K). The result of the encryption process from the counter block is XORed with the first m-bit block P_1 of the plaintext (P). This process is repeated with the counter incremented by one for $P_2, P_3, \ldots$, until the last block P_n of the message is encrypted. The counter mode of operation is depicted in Fig. 1.

Fig. 1: Block-cipher $CIPH$ in Counter Mode



The authentication tag is computed from the ciphertext C and the hash-key $H = CIPH_K(0^m)$ with the GHASH function. This GHASH function is defined as follows: let $C = C_1C_2 \cdots C_n$ be the decomposition of C into blocks of size m bits, then the hash value of the message C is

computed as follows

$$\text{GHASH}(C) = C_n H + C_{n-1} H + \cdots + C_1 H^n,$$

where C_i and H are considered as elements of $\mathbb{F}_{2^m}$. The computation of GHASH(C) can be computed as a sequence of multiplications by H followed by an addition of C_i in $\mathbb{F}_{2^m}$. This method is shown in Algorithm 1.

Algorithm 1 GHASH(C, H)

Require: $C = (C_1, \ldots, C_n)$ where $C_i \in \mathbb{F}_{2^m}$ and $H \in \mathbb{F}_{2^m}$
Ensure: $V =$ GHASH$](C, H)$
  $V \leftarrow C_1$
  for $i = 2$ **to** n **do**
    $V \leftarrow H \times V + C_i$
  end for
  $V \leftarrow (H \times V)$
  return V

Algorithm 1 computes $C_1 H^n + C_2 H^{n-1} + \cdots + C_1 H^n$ with n multiplications and $n-1$ additions in $\mathbb{F}_{2^m}$. These operations can be performed in sequence with one multiplier and one adder in $\mathbb{F}_{2^m}$. Let S_M and D_M be the space and time complexities of the considered $\mathbb{F}_{2^m}$ multiplier and let D_X be the delay of an XOR gate. The space complexity ($\mathcal{S}$) of this multiply-and-add architecture is as follows

$$\mathcal{S} = S_M + m \text{ XORs}. \tag{1}$$

The overall delay for the computation of GHASH(C, H) is as follows

$$\mathcal{D} = n(D_M + D_X). \tag{2}$$

3. Field Multiplier based on TMVP

The main operation in GHASH computation is the multiplication in $\mathbb{F}_{2^m}$. It is thus important to have an efficient $\mathbb{F}_{2^m}$ multiplier considering both time and space complexities. Consequently, we

focus on subquadratic complexity multiplier, in particular we consider the approach of Patel [16] which uses the subquadratic space complexity multiplier based on a TMVP approach [4].

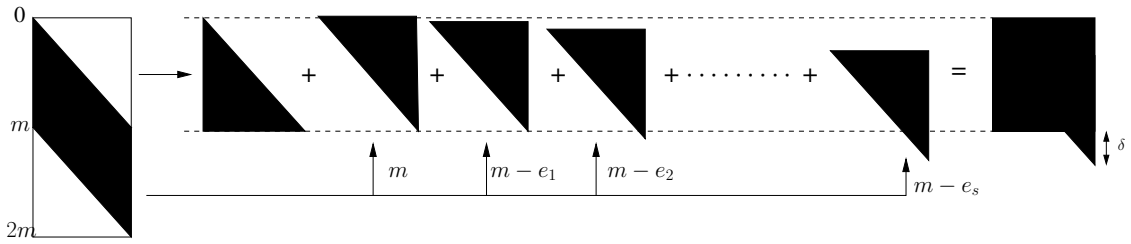
Let $\mathbb{F}_{2^m}$ be a binary field, it is defined as the set of binary polynomials modulo an irreducible polynomial $f(x) = x^m + e(x)$ of degree m. An element in $\mathbb{F}_{2^m} = \mathbb{F}_2[x]/(f(x))$ is then a binary polynomial of degree less than m.

We assume that $f(x) = x^m + e(x)$ where $e(x) = 1 + x^{e_1} + \ldots + x^{e_s}$ and $\delta = \deg e(x) = e_s$ is small compared to m. This is for example the case for the NIST standard for AES-GCM [14] where $f(x) = x^{128} + x^7 + x^2 + x + 1$. Following P. Patel in [16] we can derive a subquadratic multiplier modulo such irreducible polynomial with the following steps.

- **Step 1: matrix reduction.** We first follow the construction of Halbutogullary and Koç [6]. We start with the following $2m \times m$ matrix-vector formulation of the multiplication of two polynomials $U(x) = \sum_{i=0}^{m-1} u_i x^i$ and $V(x) = \sum_{i=0}^{m-1} v_i x^i$.

$$\begin{bmatrix} u_0 & 0 & 0 & \cdots & 0 \\ u_1 & u_0 & 0 & \cdots & 0 \\ u_2 & u_1 & u_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ u_{m-1} & u_{m-2} & u_{m-3} & \cdots & u_0 \\ 0 & u_{m-1} & u_{m-2} & \cdots & u_1 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & u_{m-1} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{m-1} \end{bmatrix} \tag{3}$$

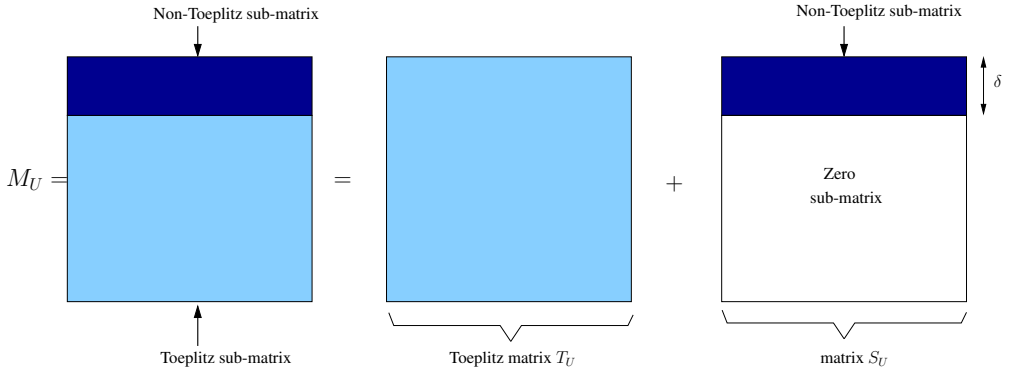
  Then we use that $x^m = e(x) \mod f(x)$ to reduce the rows of the matrix in (3) which correspond to the monomials x^i with $i \geq m$. This leads to the following scheme for the reduction.



This diagram shows that the matrix remains not reduced since there are δ non-zero rows corresponding to monomials x^m, x^{m+1}, ... ,$x^{m+\delta-1}$ after this first matrix reduction. We have to perform this matrix reduction a second time to have a fully reduced matrix.

- **Step 2 : matrix decomposition.** P. Patel noticed in [16] that after the second matrix reduction, the resulting $m \times m$ matrix, denoted M_U, has a submatrix formed by the rows $\delta, \delta+1, \ldots, m-1$ which is Toeplitz. One can then rewrite this matrix as the sum of an $m \times m$ Toeplitz matrix T_U and a matrix S_U which has non-zero entries only in the first δ rows as shown in Fig 2.

Fig. 2: Matrix decomposition



The matrix vector product $M_U \cdot V$ is then split into one Toeplitz matrix vector product $T_U \cdot V$ and one non-Toeplitz matrix vector product $S_U \cdot V$:

$$M_U \cdot V = T_U \cdot V + S_U \cdot V.$$

- **Step 3 : computation of $S_U \cdot V$.** The matrix vector product $S_U \cdot V$ is computed through δ independent circuits each performing a row-vector product and consisting of m parallel AND gates and a binary tree of $m-1$ XOR gates. This computation of the matrix product $S_U \cdot V$ has the following complexity

$$\begin{cases} \mathcal{S} &= \delta(m-1) \text{ XORs and } \delta m \text{ ANDs} \\ \mathcal{D} &= \log_2(m) D_X + D_A \end{cases} \tag{4}$$

- **Step 4 : computation of $T_U \cdot V$.** The Toeplitz-matrix-vector-product $T_U \cdot V$ can be computed with a divide and conquer approach leading to a subquadratic multiplier. Specifically, if m is even, Fan and Hasan proposed in [4] to use the *two-way split* formula of Winograd [22] shown in Table 1 to compute $T \cdot V$, where T is an $n \times n$ Toeplitz matrix and V is a vector of size m.

Table 1 TMVP two-way split formula

Splitting	Recursion	Reconstruction
$T = \begin{bmatrix} T_1 & T_0 \\ T_2 & T_1 \end{bmatrix}$ $V = \begin{bmatrix} V_0 \\ V_1 \end{bmatrix}$	$P_0 = (T_0 + T_1) \cdot V_1,$ $P_1 = T_1 \cdot (V_0 + V_1),$ $P_2 = (T_1 + T_2) \cdot V_0,$	$T \cdot V = \begin{bmatrix} P_0 + P_1 \\ P_2 + P_1 \end{bmatrix}$

When $m = 2^t$ Fan and Hasan obtained a TMVP multiplier with the following complexities when the formula in 1 is used recursively:

$$\begin{cases} \mathcal{S} &= \frac{11}{2} m^{\log_2(3)} - 6m + \frac{1}{2} \text{ XORs and } m^{\log_2(3)} \text{ ANDs} \\ \mathcal{D} &= 2\log_2(m) D_X + D_A \end{cases} \tag{5}$$

Hasan *et al.* in [8] noticed that the TMVP multiplier can be decomposed into four independent computations: Component Matrix Formation (CMF), Component Vector Formation (CVF), Component Multiplication (CM) and Reconstruction (R). The recursive formulas for these four independent computations are shown in Table 2 along with their time and space

complexities.

Table 2 Space and time complexities of different sub-computations in the Fan-Hasan multiplier

Computation	Split	Recursion	Complexity
CMF	$T = \begin{bmatrix} T_1 & T_0 \\ T_2 & T_1 \end{bmatrix}$	$CMF(T) = (CMF(T_0 + T_1), CMF(T_1), CMF(T_1 + T_2))$	$\mathcal{S}_{CMF} = \frac{5}{2}m^{\log_2(3)} - 3m + \frac{1}{2}$XORs $\mathcal{D}_{CMF} = \log_2(m)D_X$
CVF	$V = \begin{bmatrix} V_0 \\ V_1 \end{bmatrix}$	$CVF(V) = (CFV(V_1), CVF(V_0 + V_1), CVF(V_0))$	$\mathcal{S}_{CVF} = m^{\log_2(3)} - m$XORs $\mathcal{D}_{CVF} = \log_2(m)D_X$
CM	–	$\hat{W} = CTF(T) \otimes CVF(V)$	$\mathcal{S}_{CM} = m^{\log_2(3)}$ANDs $\mathcal{D}_{CM} = D_A$
R	$\hat{W} = [\hat{W}_0, \hat{W}_1, \hat{W}_2]$	$W = R(\hat{W}) = (R(\hat{W}_0) + R(\hat{W}_1), R(\hat{W}_1) + R(\hat{W}_2))$	$\mathcal{S}_R(n) = 2m^{\log_2(3)} - 2m$XORs $\mathcal{D}_R(n) = \log_2(m)D_X$

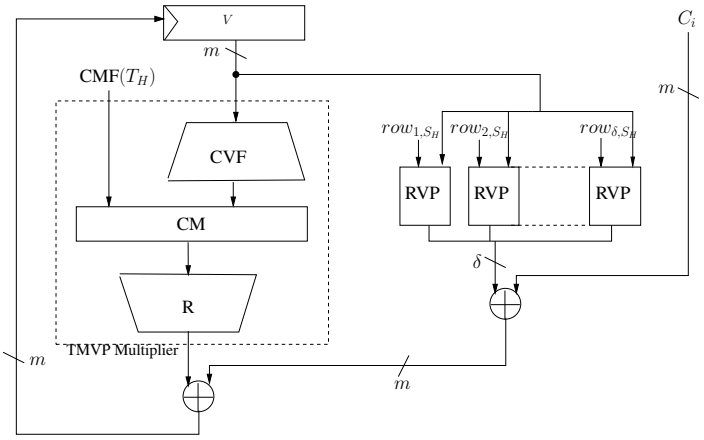
- **Final step: multiply-and-add architecture.** In Fig. 3 we provide the multiply-and-add architecture for the GHASH computation of [16] which is based on the decomposition

$$M_H \cdot V = T_H \cdot V + S_H \cdot V$$

for a multiplication by H. The matrix vector product $T_H \cdot V$ is computed with a suquadratic TMVP multiplier which is decomposed into the different computations (CMF,CVF, CM and R). We assume that the entries of T_H, $CMF(T_H)$ and S_H are precomputed. The multiplications $T_H \cdot V$ and $S_H \cdot V$ are computed in parallel and their results are added to get $M_H \cdot V$. In Fig. 3, RVP stands for row-vector-product and row_{i,S_H} is the i-th row of S_H. The space complexity of this multiply-and-add architecture is obtained by adding the complexity of $S_H \cdot V$ plus the complexity of $T_H \cdot V$ and an adder. The delay of this architecture is equal to the delay of a TMVP multiplier plus D_X.

$$\begin{cases} \mathcal{S}(m) &= \frac{11}{2}m^{\log_2(3)} + (\delta - 6)m + \frac{1}{2} - \delta \text{ XORs and } m^{\log_2(3)} + \delta m \text{ ANDs} \\ \mathcal{D}(m) &= (2\log_2(m) + 1)D_X + D_A \end{cases}$$

Fig. 3: Multiply-and-add architecture with TMVP multiplier [16]



4. Recombination of the single-multiply-and-add architecture based on Fan-Hasan multiplier

We present in this section our contribution for the multiply-and-add architecture. The current multiply-and-add architecture in Fig. 3 has a critical path delay of $(2\log_2(m)+1)D_X + D_A$. This delay is for the most part due to the blocks CVF and R, each contributing by $\log_2(m)D_X$ to the critical path delay. Our strategy to reduce the critical path delay is to recombine the blocks of the architecture in order to have a block $CVF \circ R$ which can computed with a delay of $\log_2(m)D_X$. So we first explain the recombination of the block of the architecture in the next subsection. The remainder of the paper will be dedicated to explained how the new blocks can be implemented.

4.1. Recombination of the multiply-and-add architecture

We recombine the blocks of the multiplier in order to have the output of the reconstruction block (R) directly input to the block CVF of the product $T_H \cdot V$ and $S_H \cdot V$. The blocks R and CVF are separated a bit-wise XOR. We use the following lemma to reverse the order of the block R and the addition.

Lemma 1. *We consider the term $T_U \cdot V + S_U \cdot V + C_i$ involved in Fig. 3 where T_U is a $m \times m$*

Toeplitz matrix. Let $\hat{W} = CMF(T_U) \otimes CVF(V)$ *then the following equation holds :*

$$\underbrace{R(\hat{W}) \oplus S_U \cdot V \oplus C_i}_{(*)} = R\left(\hat{W} \oplus (CMF(Id) \otimes CVF(S_U \cdot V)) \oplus CVF(C_i)\right). \tag{6}$$

Proof. Let us first denote $Y = S_U \cdot V + C_i$. The multiplication of Y by the identity matrix Id can be done with a TMVP multiplier as follows:

$$Y = R(CMF(Id) \otimes CVF(Y)). \tag{7}$$

Now we add Y and $R(\hat{W})$ to get $(*)$ in (6) and we use the linearity of R to obtain the following

$$\begin{aligned} R(\hat{W}) \oplus S_U \cdot V \oplus C_i &= R(\hat{W}) \oplus Y \\ &= R(\hat{W}) \oplus R(CMF(Id) \otimes CVF(Y)) \quad \text{(using (7))} \\ &= R\left(\hat{W} \oplus (CMF(Id) \otimes CVF(Y))\right) \quad \text{(by linearity of } R\text{)}. \end{aligned}$$

Finally, we obtain the required expression (6) by using the linearity of CVF which gives $CVF(Y) = CVF(S_U \cdot V) \oplus CVF(C_i))$.

□

Lemma 1 provides a way to transform the addition in

$$R(CMF(T_H) \otimes CVF(V)) \oplus S_U \cdot V \otimes C_i$$

to an addition in component formation representation. We use this property to recombine the multiply-and-add architecture.

Main recombination of the multiply-and-add architecture. We use Lemma 1 to recombine the multiply-and-add architecture as follows:

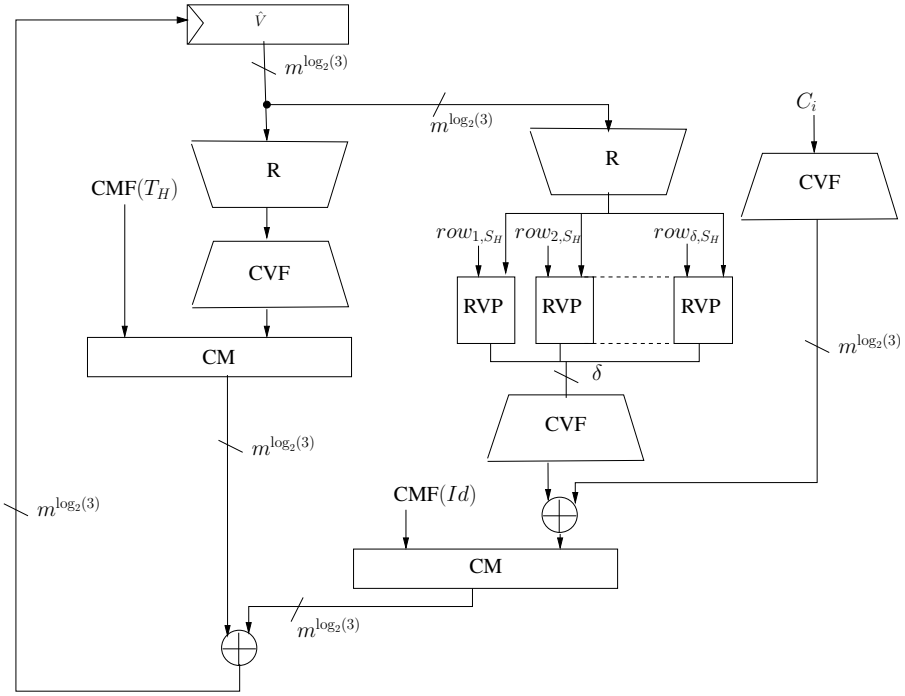
- We keep $\hat{V}$ in component formation in register of size $m^{\log_2(3)}$.

- At each iteration we compute $\hat{W} = CVF(R(\hat{V}))$ and then the new value of $\hat{V}$ is

$$\hat{V} = \hat{W} \oplus (CMF(Id) \otimes (CVF(S_U \cdot V) \oplus CVF(C_i)$$

This recombination leads to the architecture in Fig. 4. In this architecture we have what we sought: a block R which is directly connected to the block CVF. But we have also that the R is connected to the δ blocks RVP. We have some new blocks in the recombined architecture: two blocks CVF, one applied to $S_U \cdot V$ and one to C_i, and a block CM for the multiplication by $CMF(Id)$.

Fig. 4: Recombination of the mult-and-add architecture



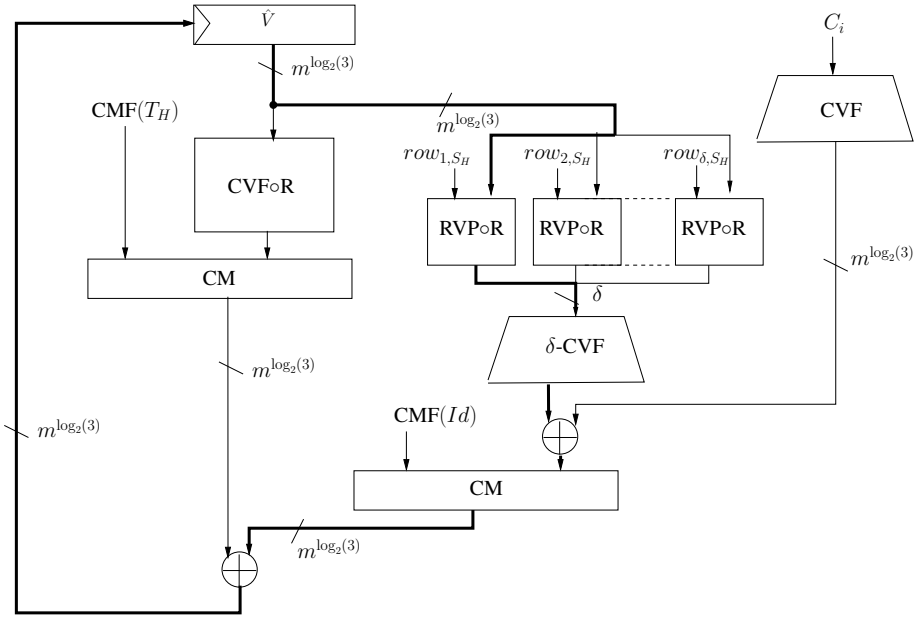
Final recombination by block merging. In order to get the final version of the proposed architecture, we modify the architecture in 4 by merging the following blocks

- Merging the block R and the block CVF into a $CVF \circ R$ block.

- Merging the block R with each block RVP into $RVP \circ R$ block.

The multiply-and-add architecture with such merged blocks is shown in Fig. 4. In this architecture the block δ-CVF is a regular CVF computation but which has an input vector with only δ non-zero coefficients. In the next three subsections we provide explicit methods for the computation of $CVF \circ R$, $RVP \circ R$ and also for δ-CVF.

Fig. 5: Recombination of the mult-and-add architecture



4.2. Merging the block Reconstruction and Component Vector Formation

We show here that the reconstruction block R and the block CVF in Fig.4 can be merged in a CVF∘R block in order to divide by two the delay of the sequence of operation *Reconstruction followed by a CVF*. We denote CVF∘R the function $\{0,1\}^{m^{\log_2(3)}} \rightarrow \{0,1\}^{m^{\log_2(3)}}$ which consists to reconstruct an array of size $m^{\log_2(3)}$ and then apply the component vector formation. The following proposition establishes a recursive formula for this merged CVF∘R.

Proposition 1 (Recursive formula for CVF∘R)**.** *Let $\hat{W}$ be a $m^{\log_2(3)}$ bit array where $n = 2^s$. The CVF∘R function can be computed recursively as follows:*

i) If $|\hat{W}| = 1$ *we have* $CVF \circ R(\hat{W}) = \hat{W}$.

ii) If $|\hat{W}| > 1$ *we split* $\hat{W} = [\hat{W}_0, \hat{W}_1, \hat{W}_2]$ *and we recursively compute:*

$$CVF \circ R(\hat{W}) = [CVF \circ R(\hat{W}_0 \oplus \hat{W}_2), CVF \circ R(\hat{W}_1 \oplus \hat{W}_2), CVF \circ R(\hat{W}_0 \oplus \hat{W}_1)]. \quad (8)$$

Proof. We prove it by induction.

- *Proof of i).* For $|\hat{W}| = 1$, using the definition of R we have $R(\hat{W}) = \hat{W}$ and then

  $$CVF \circ R(\hat{W}) = CVF(R(\hat{W})) = CVF(\hat{W}) = \hat{W},$$

  using the definition of CVF.

- *Proof of ii).* For $|\hat{W}| > 1$. The recursive formula for R in Table 1 provides

  $$R(\hat{W}) = [R(\hat{W}_0) \oplus R(\hat{W}_1), R(\hat{W}_1) \oplus R(\hat{W}_2)].$$

  We apply the recursive definition of CVF (cf. Table 1) to this vector and we obtain

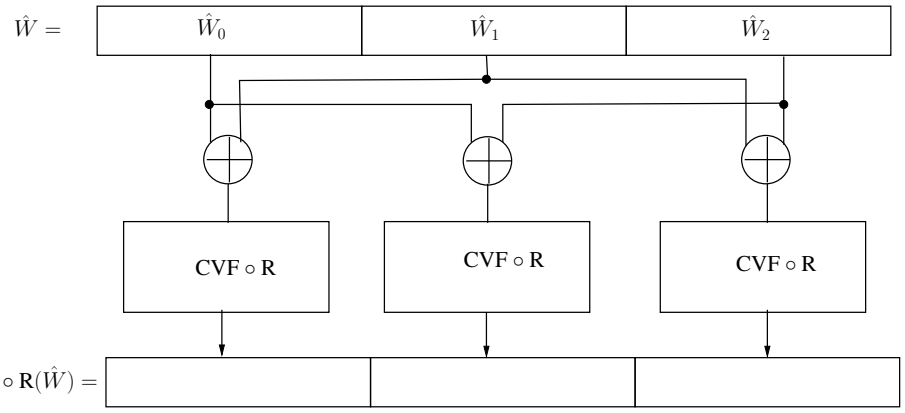
  $$\begin{aligned}
  &CVF \circ R(\hat{W})) = CVF([R(\hat{W}_0) \oplus R(\hat{W}_1), R(\hat{W}_1) \oplus R(\hat{W}_2)]) \\
  &= [CVF(R(\hat{W}_0) \oplus R(\hat{W}_1)), CVF((R(\hat{W}_0) \oplus R(\hat{W}_1)) \oplus (R(\hat{W}_1) \oplus R(\hat{W}_2))), CVF(R(\hat{W}_1) \oplus R(\hat{W}_2))] \\
  &= [CVF(R(\hat{W}_0) \oplus R(\hat{W}_1)), CVF((R(\hat{W}_0) \oplus R(\hat{W}_2))), CVF(R(\hat{W}_1) \oplus R(\hat{W}_2))] \quad \text{(cancellation)} \\
  &= [CVF(R(\hat{W}_0 \oplus \hat{W}_1)), CVF(R(\hat{W}_0 \oplus \hat{W}_2)), CVF(R(\hat{W}_1 \oplus \hat{W}_2))] \quad \text{(by linearity of } R\text{)} \\
  &= [CVF \circ R(\hat{W}_0 \oplus \hat{W}_1), CVF \circ R(\hat{W}_0 \oplus \hat{W}_2), CVF \circ R(\hat{W}_1 \oplus \hat{W}_2)],
  \end{aligned}$$

  which concludes the proof of Proposition 1.

□

We derive a circuit from the formula of Proposition 1 which is shown in Fig. 6. The following lemma establishes the complexity of this circuit.

Fig. 6: Circuit for CVF∘R operation



Lemma 2. *We consider the circuit shown in Fig. 6 for CVF∘R obtained by applying recursively* (8)*. This circuit has the following complexity*

$$\begin{cases} \mathcal{S} &= m^{\log_2(3)} \log_2(m) \textit{ XORs}, \\ \mathcal{D} &= \log_2(m) D_X. \end{cases} \tag{9}$$

Proof. We prove Lemma 2 by induction on t in $m = 2^t$. For $m = 1 = 2^0$ the complexity in (9) is in this case direct from i) in Proposition 1.

Now, we assume that the corresponding arithmetic circuit provided by (8) is true for $\hat{W}$ of size $\ell = m^{\log_2 3}$ with $m = 2^t$ and we show it for $m' = 2^{t+1}$, i.e., for $\hat{W}$ of size

$$\ell' = {m'}^{\log_2(3))} = (2m)^{\log_2(3)} = 3(m^{\log_2(3)}).$$

From Fig. 6, we can notice that $\mathcal{S}(\ell') = 3\mathcal{S}(\ell) + 3m^{\log_2(3)}$. The induction hypothesis provides us $\mathcal{S}(\ell) = \log_2(m) m^{\log_2(m)}$ which gives the required expression:

$$\begin{aligned} \mathcal{S}(\ell') &= 3\log_2(m) m^{\log_2(m)} + 3m^{\log_2(3)} \\ &= (\log_2(m) + 1) 3m^{\log_2(3)} \\ &= \log_2(m')(m')^{\log_2(3)}. \end{aligned}$$

For the delay we have from Fig. 6 that $\mathcal{D}(\ell') = \mathcal{D}(\ell) + D_X$. By induction hypothesis we have $\mathcal{D}(\ell) = \log_2(m)$, and then we obtain the required expression $\mathcal{D}(\ell') = \log_2(m)+1 = \log_2(m')$. $\square$

4.3. Merging the Reconstruction and Row Vector Product

In this subsection we provide a method to merge the reconstruction block and the RVP block in order to reduce the critical path delay of RVP $\circ$ R.

Lemma 3. *Let a row matrix **row** and let $\hat{W}$ be the component vector formation of a vector W. We can compute $RVP(\textbf{row}, R(\hat{W})) = \textbf{row} \cdot R(\hat{W})$ with the following formula*

$$RVP(\textbf{row}, R(\hat{W})) = Sum(CVF(\textbf{row}) \otimes \hat{W})$$

where Sum is the bit-wise XOR of all the coefficients of an array.

Proof. We prove the lemma by induction on the size of W. We split $\hat{W}$ in three part $\hat{W} = [\hat{W}_0, \hat{W}_1, \hat{W}_2]$. By definition of R in Table 2 we have $R(\hat{W}) = [R(\hat{W}_0) \oplus R(\hat{W}_1), R(\hat{W}_1) \oplus R(\hat{W}_2)]$. We then split **row** in two parts $\textbf{row} = [\textbf{row}_0, \textbf{row}_1]$ and we re-express the RVP $\textbf{row} \cdot R(\hat{W})$ as follows

$$\begin{aligned}
\textbf{row} \cdot R(\hat{W}) &= \textbf{row} \cdot [R(\hat{W}_0) \oplus R(\hat{W}_1), R(\hat{W}_1) \oplus R(\hat{W}_2)] \\
&= \textbf{row}_0 \cdot (R(\hat{W}_0) \oplus R(\hat{W}_1)) \oplus \textbf{row}_1 \cdot (R(\hat{W}_1) \oplus R(\hat{W}_2)) \\
&= \underbrace{\textbf{row}_0 \cdot R(\hat{W}_0)}_{(*)} \oplus \underbrace{(\textbf{row}_0 \oplus \textbf{row}_1) \cdot R(\hat{W}_1))}_{(**)} \oplus \underbrace{\textbf{row}_1 \cdot R(\hat{W}_2))}_{(***)}
\end{aligned}$$

We can apply the induction hypothesis to the three RVP $(*), (**)$ and $(***)$, this leads to the

following:

$$\begin{aligned}
\mathbf{row} \cdot R(\hat{W}) &= Sum(CVF(\mathbf{row}_0) \otimes \hat{W}_0) \oplus Sum(CVF(\mathbf{row}_0 \oplus \mathbf{row}_1) \otimes \hat{W}_1) \\
&\qquad \oplus Sum(CVF(\mathbf{row}_1) \otimes \hat{W}_2) \\
&= Sum([CVF(\mathbf{row}_0), CVF(\mathbf{row}_0 \oplus \mathbf{row}_1), CVF(\mathbf{row}_1)] \otimes [\hat{W}_0, \hat{W}_1, \hat{W}_2]) \\
&= Sum(CVF(\mathbf{row}) \otimes \hat{W}), \quad \text{(by definition of CVF in Table 2)}
\end{aligned}$$

and this ends the proof. □

Complexity of $RVP \circ R$. The circuit performing $RVP \circ R$ with input **row** and $\hat{V}$ consists in $m^{\log_2(3)}$ AND gates and $(m^{\log_2(3)} - 1)$ XOR gates organized in a binary tree performing the Sum. The delay of this computation is thus equal to

$$D_A + \lceil \log_2(m^{\log_2(3)}) \rceil D_X = D_A + \lceil \log_2(3) \log_2(m) \rceil D_X. \tag{10}$$

4.4. Computing δ-CVF

This CVF take as input a vector V which has its last $m - \delta$ coefficients which are equal to 0 assuming that δ is small compared to m. We will show here that the computation of CVF(V) is simplified in this case. Indeed, when we apply one recursion of CVF to a vector $V = [V_0, 0]$ we have

$$CVF(V) = [CVF(V_0), CVF(V_0 \oplus 0), CVF(0)] = [CVF(V_0), CVF(V_0), 0]. \tag{11}$$

So this means that the computation of $CVF(V)$ is reduced to the computation of $CVF(V_0)$. We can apply this property recursively. The following lemma establishes the expression of CVF for multiple recursions.

Lemma 4. *Let $V = [v_0, \ldots, v_{2^s-1}, 0, \ldots, 0]$ be a vector of size $m = 2^t$ with $t = s + u$ and let*

$\widetilde{V} = [v_0, \ldots, v_{2^s-1}]$. *Then we have* $CVF(V) = [\hat{V}_0, \hat{V}_1, \ldots, \hat{V}_{3^u-1}]$ *such that* $|\hat{V}_i| = 3^{t-u}$ *and*

$$\begin{aligned} \hat{V}_i &= CVF(\widetilde{V}) \text{ if } i = \textstyle\sum_{j=0}^{t-s} i_j 3^j \text{ with all } i_j \in \{0,1\}, \\ \hat{V}_i &= 0 \text{ if } i = \textstyle\sum_{j=0}^{t-s} i_j 3^j \text{ and there exists one } i_j = 2. \end{aligned} \tag{12}$$

Proof. We prove the lemma by induction on u. For $u = 1$ this is direct from (11). Then, we assume the lemma is true for u and we prove it for $u+1$. We consider a vector V of size 2^{s+u+1}. We split V as $V = [V_0, 0]$ which implies that V_0 is of size 2^{s+u} and we apply one recursion of CVF we get

$$CVF(V) = [CVF(V_0), CVF(V_0), 0].$$

We then use the induction hypothesis for $CVF(V_0)$ which yields:

$$CVF(V) = [\underbrace{\widehat{(V_0)}_0, \widehat{(V_0)}_1, \ldots, \widehat{(V_0)}_{3^u-1}}_{CVF(V_0)}, \underbrace{\widehat{(V_0)}_0, \widehat{(V_0)}_1, \ldots, \widehat{(V_0)}_{3^u-1}}_{CVF(V_0)}, 0, \ldots, 0]. \tag{13}$$

And by induction hypothesis $\widehat{(V_0)}_i = CVF(\widetilde{V})$ if $i = \sum_{j=0}^{u-1} i_j 3^j$ does not contain any digit equal to 2, otherwise it is 0. Let us relabel each term in (13) as in $CVF(V) = [\hat{V}_0, \hat{V}_1, \ldots, \hat{V}_{3^{u+1}-1}]$. In this case we have

$$\begin{aligned} \hat{V}_i &= \widehat{(V_0)}_i \text{ if } i = 0, \ldots, 3^u - 1, \\ \hat{V}_i &= \widehat{(V_0)}_i \text{ for } i = 3^u, \ldots, 2 \cdot 3^u - 1, \\ \hat{V}_i &= 0 \text{ for } i = 2 \cdot 3^u, \ldots, 3^{u+1} - 1. \end{aligned} \tag{14}$$

We can easily check that:

- For $i < 3^u$, $\hat{V}_i = CVF(\widetilde{V})$ only if $i = \sum_{j=0}^{u} i_j 3^j$ does not contain any digit equal to 2, since it is already the case for $\widehat{(V_0)}_i$.
- For $i = 3^u, \ldots, 2 \cdot 3^u - 1$, we can write $V_i = (\widehat{V_0})_{i'}$ with $i' = i - 3^u$. This implies that V_i is equal to $CVF(\widetilde{V})$ when $i' = \sum_{j=0}^{u-1} i'_j 3^j$ does not contain a 2, and consequently when $i = \sum_{j=0}^{u} i'_j 3^j + 3^u$ does not contain a 2.

- For $i > 2 \cdot 3^u$, we have $\hat{V}_i = 0$ and since $i = \sum_{j=0}^{u-1} i_j 3^j + 2 \cdot 3^u$ it contains a 2, as required.

This ends the proof. □

Lemma 4 teaches us that the computation of $\delta - \text{CVF}(V)$ is reduced to the computation of

$$\widetilde{V} = \text{CVF}([v_0, \ldots, v_{\delta-1}, \underbrace{0, \ldots, 0}_{2^{\lceil \log_2(\delta) \rceil} - \delta \ zeros}]).$$

The other coefficients of $\text{CVF}(V)$ are either 0 or a copy of a coefficient of $\widetilde{V}$. This leads to the following complexity of the computation of δ-$\text{CVF}(V)$: this is the complexity of a CVF applied to vector of size $2^{\lceil \log_2(\delta) \rceil}$:

$$\begin{cases} \mathcal{S}_{\delta-\text{CVF}} &= 3^{\lceil \log_2(\delta) \rceil} - 2^{\lceil \log_2(\delta) \rceil} \text{XORs}, \\ \mathcal{D}_{\delta-\text{CVF}} &= \lceil \log_2(\delta) \rceil D_X. \end{cases}$$

4.5. Overall complexity

We evaluate the complexity of the proposed architecture (Fig. 5), with the designs of the blocks $CVF \circ R$ and $RVP \circ R$ and $\delta-$CVF presented in Subsection 4.2, 4.3 and 4.4, respectively. We obtain the overall space complexity of the proposed multiply-and-add architecture by adding the space complexity of each block which appears in Fig. 5. We obtain the following

$$\begin{array}{rcl} 2 \times \mathcal{S}_{\text{CVF}} &=& 2m^{\log_2(3)} - 2m \text{ XORs} \\ 1 \times \mathcal{S}_{\delta-\text{CVF}} &=& 3^{\lceil \log_2(\delta) \rceil} - 2^{\lceil \log_2(\delta) \rceil} \text{XORs}, \\ 2 \times \mathcal{S}_{\text{CM}} &=& 2m^{\log_2(3)} \text{ ANDs} \\ 2 \times \mathcal{S}_{\text{CA}} &=& 2m^{\log_2(3)} \text{ XORs} \\ 1 \times \mathcal{S}_{\text{CVF}\circ\text{R}} &=& m^{\log_2(m)} \log_2(m) \text{ XORs} \\ \delta \times \mathcal{S}_{\text{RVP}\circ\text{R}} &=& \delta m^{\log_2(3)} - \delta \text{ XORs and } \delta m^{\log_2(3)} \text{ ANDs} \\ \hline \text{Total} &=& (\log_2(m) + \delta + 4) m^{\log_2(3)} - \delta - 2m + 3^{\lceil \log_2(\delta) \rceil} - 2^{\lceil \log_2(\delta) \rceil} \text{XORs} \\ & & \text{and } (2 + \delta) m^{\log_2(3)} \text{ ANDs} \end{array}$$

Now, we evaluate the time complexity of the architecture in Fig. 5. There are two main paths:

- The critical path is the thick line starting and finishing at the register containing $\hat{V}$ in Fig. 5. On this path we have the following blocks: RVP $\circ$ R, $\delta -$ CVF, CA, CM and CA. Then if we add the delay of each block we obtain an overall delay of this path:

$$(\log_2(3)\log_2(n) + \lceil \log_2(\delta) \rceil + 2)D_X + 2D_A.$$

- The path which is going through CVF $\circ$ R, CM and CA has a smaller delay which is equal to $(\log_2(n) + 1)D_X + D_A$.

We can conclude that the delay of the proposed multiply-and-add architecture is as follows

$$\mathcal{D} = (\log_2(3)\log_2(n) + \lceil \log_2(\delta) \rceil + 2)D_X + 2D_A.$$

5. Comparison and conclusion

We considered in this paper the implementation of the GHASH function used in the Galois counter mode for the generation of the authentication tag. We presented a recombination of the multiply-and-add architecture of P. Patel [16] which is based on a subquadratic space complexity TMVP multiplier. This recombination was meant to reduce the overall critical path delay of the architecture. This delay reduction was obtained by merging some blocks of the recombined architecture. In Table 3 we provide the complexity of the proposed approach and also the complexities of the usual multiply-and-add architecture using on the best approaches for the design of the multiplier.

The proposed approach is a subquadratic space complexity architecture which have delay which is smaller than $2\log_2(m)D_X + D_A$ when δ is sufficiently small. This improvement was obtained at a cost of an increase of the space complexity. We notice that it is the first approach with subquadratic space complexity which has a delay smaller than $(2\log_2(m) + O(1))D_X + D_A$. Indeed, Table 3 shows that the only approach of the literature which has delay smaller than $2\log_2(m)D_X + D_1$

Table 3 Complexities of multiply-and-add architectures

Approach	#XOR	# AND	Delay
Mastrovito [6]	$m^2 + O(m)$	$m^2 + O(m)$	$(\log_2(m) + 1)D_X + D_A$
Karatsuba [19]	$6m^{\log_2(3)} + O(m)$	$m^{\log_2(3)}$	$(2\log_2(m) + O(1))D_X + D_A$
Karatsuba [12]	$5.25m^{\log_2(3)} + O(m)$	$m^{\log_2(3)}$	$(2\log_2(m) + O(1))D_X + D_A$
TMVP [16]	$5.5m^{\log_2(3)} + O(m)$	$(2\log_2(m) + 1)D_X + D_A$	
TMVP [7]	$5.5m^{\log_2(3)} + O(m)$	$m^{\log_2(3)}$	$(2\log_2(m) + O(1))D_X + D_A$
Proposed*	$(\log_2(m) + \delta + 4)m^{\log_2(3)} - \delta$ $+3^{\lceil \log_2(\delta) \rceil} - 2^{\lceil \log_2(\delta) \rceil} - 2m$	$(2 + \delta)m^{\log_2(3)}$	$(1.59\log_2(m) + \lceil \log_2(\delta) \rceil + 2)D_X$ $+2D_A$

$*$ For multiplication modulo $x^m + e(x)$ and $\delta = \deg e(x) + 1$

is the one with a quadratic space complexity. This proposal remains a bit theoretical, but this is a first step towards obtaining a subquadratic space complexity multiply-and-add architecture or a multiplier in $\mathbb{F}_{2^m}$ with an *optimal* delay of $\log_2(m) + O(1)D_X + O(1)D_A$.

6. References

[1] D. Canright. A very compact s-box for AES. In *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005.

[2] Ç. K. Koç and B. Sunar. Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields. *IEEE Trans. on Comput.*, 47:353–356, March 1998.

[3] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[4] H. Fan and M. A. Hasan. A New Approach to Sub-quadratic Space Complexity Parallel Multipliers for Extended Binary Fields. *IEEE Trans. Computers*, 56(2):224–233, 2007.

[5] H. Fan and M. A. Hasan. Subquadratic Computational Complexity Schemes for Extended Binary Field Multiplication Using Optimal Normal Bases. *IEEE Trans. Computers*, 56(10):1435–1437, 2007.

[6] A. Halbutogullari and Ç.K. Koç. Mastrovito multiplier for general irreducible polynomials. *IEEE Trans. on Comp.*, 49(5):503 –518, May 2000.

[7] J. Han and H. Fan. GF(2^n) Shifted Polynomial Basis Multipliers Based on Subquadratic Toeplitz Matrix-Vector Product Approach for All Irreducible Pentanomials. *IEEE Trans. Computers*, 64(3):862–867, 2015.

[8] M. A. Hasan, N. Meloni, A. H. Namin, and C. Negre. Block Recombination Approach for Subquadratic Space Complexity Binary Field Multiplication Based on Toeplitz Matrix-Vector Product. *IEEE Trans. Computers*, 61(2):151–163, 2012.

[9] M. M. Kermani and A. Reyhani-Masoleh. Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM. *IEEE Trans. Computers*, 61(8):1165–1178, 2012.

[10] E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Dept. of Electrical Eng., Linköping Univ., Sweden, 1991.

[11] D. A. McGrew and J. Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT*, volume 3348 of *LNCS*, pages 343–355, 2004.

[12] C. Negre. Efficient binary polynomial multiplication based on optimized Karatsuba reconstruction. *J. Cryptographic Engineering*, 4(2):91–106, 2014.

[13] NIST. *Advanced Encryption Standard (AES)*, November 2001.

[14] NIST. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, November 2007.

[15] C. Paar. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields. *IEEE Trans. Comput.*, 45(7):856–861, 1996.

[16] P. Patel. Parallel multiplier designs for the Galois/counter mode of operation. Master's thesis, Electrical and Computer Engineering, University of Waterloo, 2008.

[17] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 239–254. Springer, 2001.

[18] A. Satoh, T. Sugawara, and T. Aoki. High-performance hardware architectures for Galois Counter Mode. *IEEE Transactions on Computers*, 58(7):917–930, 2009.

[19] J. Sun, M. Gu, K.-Y. Lam, and H. Fan. Overlap-free Karatsuba-Ofman Polynomial Multiplication Algorithm. *IET Information Security*, 4:8–14, March 2010.

[20] B. Sunar. A Generalized Method for Constructing Subquadratic Complexity GF(2^k) Multipliers. *IEEE Trans. on Comp.*, 53:1097–1105, 2004.

[21] B. Sunar and Ç. K. Koç. An Efficient Optimal Normal Basis Type II Multiplier. *IEEE Trans. on Comp.*, 50(1):83–87, 2001.

[22] S. Winograd. *Arithmetic Complexity of Computations*. Society For Industrial & Applied Mathematics, U.S., 1980.