Automatic Generation of Formally-Proven Tamper-Resistant Galois-Field Multipliers Based on Generalized Masking Scheme

Rei Ueno*, Naofumi Homma*, Sumio Morioka†, and Takafumi Aoki*
* Tohoku University, Aramaki Aza Aoba 6-6-05, Sendai-shi, 980-8579, Japan
Email: ueno@aoki.ecei.tohoku.ac.jp
† Interstellar Technologies Inc., 690-4 Memu, Taiki, Hiroo-gun, Hokkaido, 089-2113, Japan

***Abstract*—In this study, we propose a formal design system for tamper-resistant cryptographic hardwares based on Generalized Masking Scheme (GMS). The masking scheme, which is a state-of-the-art masking-based countermeasure against higher-order differential power analyses (DPAs), can securely construct any kind of Galois-field (GF) arithmetic circuits at the register transfer level (RTL) description, while most other ones require specific physical design. In this study, we first present a formal design methodology of GMS-based GF arithmetic circuits based on a hierarchical dataflow graph, called GF arithmetic circuit graph (GF-ACG), and present a formal verification method for both functionality and security property based on Gröbner basis. In addition, we propose an automatic generation system for GMS-based GF multipliers, which can synthesize a fifth-order 256-bit multiplier (whose input bit-length is 256×77) within 15 min.**

***Keywords*—formal verification, arithmetic circuits, Galois-field, threshold implementation, side-channel attack**

I. Introduction

Cryptography has been widely utilized in many systems for secure communications, authentication, and digital signatures. Owing to the increasing number of Internet of Things (IoT) applications, many cryptographic algorithms are required to be implemented in the hardware for resource-constrained devices. Meanwhile, various physical attacks on cryptographic hardware are attracting considerable attention because of the increasingly wider diffusion of cryptographic devices. Physical attacks pose a significant threat to many devices including those currently in circulation in the market, such as smart cards. There is definitely a high demand for designing tamper-resistant cryptographic devices in the context of IoT.

Side-channel attack, which is a typical physical attack, retrieves the secret key from cryptographic hardware using side-channel information such as power consumption [1]. Typical side-channel attacks on secret key cryptography, such as Differential Power Analysis (DPA), estimate the secret key by examining the correlation between power consumption and the intermediate value of cryptographic operations. Given that modern block ciphers employ Galois-field (GF) arithmetic to construct nonlinear functions, we should consider the utilization of tamper-resistant GF arithmetic circuits when designing cryptographic hardware. Recently, in addition to conventional DPAs on S-boxes in cryptographic algorithms, side-channel attacks on GF multiplications have been reported in [2]. GF multipliers are major primitives used for constructing authenticated encryption such as AES-GCM.

Masking is a typical countermeasure against DPAs, which eliminates the correlation assumed in the attacks using randomness. However, the conventional masking countermeasures can be vulnerable by advanced attacks using power consumption caused by gate switching and glitch if they are not implemented with specific tools or libraries, which, for example, make a symmetric physical layout possible. To address the issue, threshold implementation (TI) was proposed to guarantee resistance even against such attacks in register transfer level (RTL) description [3]. In [4], Generalized Masking Scheme (GMS) was presented to consolidate the existing masking schemes including TI. Consequently, GMS can be extended to higher-order resistance, which is to say that, the dth-order GMS-based circuits are resistant to the dth-order DPAs.

Currently, it is difficult to design GMS-based GF arithmetic circuits that meet the design criteria (i.e., performance and GMS order). The existing EDA tools do not support high-level description or automatic synthesis of GF arithmetic circuits. The deficiency of high-level design methodology forces designers to describe GF arithmetic circuits using low-level expressions by hand. Such expressions make the verification of the circuits difficult. GF arithmetic circuits for security modules are required to be completely verified because bugs in the modules would be linked to critical vulnerability [5]. However, a complete verification using the conventional logic simulation is usually impossible given that most cryptographic functions are performed with 64-bit or greater operands. Nonetheless, there are no formal methods available in literature for verifying the satiation of the side-channel security property in the design.

To address the above problems, we propose a formal design and verification method for GMS-based GF arithmetic circuits. The proposed method is based on a formal approach to describing and verifying GF arithmetic circuits in [6], [7]. The basic ideas revolve around the description of GF arithmetic circuits using a high-level mathematical graph called Galois-field arithmetic circuit graph (GF-ACG) and its verification using an algebraic procedure based on a Gröbner basis (GB) and a polynomial reduction technique. Moreover, in this study, we extend the above method for verifying the tamper-resistance properties (i.e., GMS properties) of circuits given in GF-ACG. We also demonstrate the applicability of a system that automatically generates RTL codes for GF multipliers, whose functional and security properties are completely verified from a design specification viewpoint (i.e., irreducible polynomial, GF representation, arithmetic algorithm, and GMS order). We show that our system successfully generates a variety of GMS-based GF multipliers that can be resistant up to 5th-order DPAs at RTL which is thought to be sufficient in practice because many literature for side-channel leakage evaluation do not discuss DPAs greater than 5th-order.

II. Generalized masking scheme

GMS is a state-of-the-art masking-based countermeasure against higher-order DPAs [4]. A dth-order GMS can describe any kind of arithmetic circuits over $GF(2^m)$ resistant to dth-order DPAs including advanced ones that utilize glitch induced in power consumption. In this section, we first describe three security properties of GMS. Then, we show a construction of GMS-based circuits using GF parallel multipliers as examples.

A. GMS properties

The working principle of masking schemes including GMS is to represent a secret value $a \in GF(2^m)$ by $a_0 + a_1 + \cdots + a_i + \cdots + a_{s-1}$, where $a_i \in GF(2^m)$ $(0 \le i \le s-1)$ is initially given by a random mask. Each element a_i is called a share. Let a be the input and $a_0, a_1, \ldots, a_i, \ldots, a_{s-1}$ be the share of a, where s is the number of input share. Let $r \in GF(2^m)$ be the output of combinational circuit and $r_0, r_1, \ldots, r_k, \ldots, r_{s'-1}$ be the share of r, where $r_k \in GF(2^m)$ $(0 \le k \le s'-1)$ denotes the kth share, and s' is the number of output shares.

(i) *Correctness*: the first property implies that the sum of shares is equal to the secret value at the input and output of the circuit, namely, $a = a_0 + a_1 + \cdots + a_{s-1}$ and $r = r_0 + r_1 + \cdots + r_{s'-1}$. This property indicates that the shared circuit correctly performs the original (i.e., nonshared) function.

(ii) *dth-order noncompleteness*: the second property implies that any d output shares of are independent of at least one input share. The number of input shares (i.e., s) required to meet the dth-order noncompleteness is dependent on d and the degree of the circuit function. Typically, s and s' are respectively given by $dt + 1$ and $\binom{s}{t}$, where t is the degree.

(iii) *Uniformity*: the third property indicates that the input and output values are uniformly distributed.

Correctness and dth-order noncompleteness can be realized for any GF function, while some functions cannot satisfy uniformity under the constraints of two above properties. However, the uniformity criterion can be satisfied by the addition of fresh mask(s) to the non-uniform outputs [3].

B. Construction of GMS-based circuit

As an example, we describe the construction of GMS-based $GF(2^m)$ multipliers. The number of input shares s is given by $2d + 1$ because the degree of a multiplication is two (the GMS-based two-operand multiplier has $2s$ inputs in total). The number of output shares is given by $s' = \binom{s}{2}$ $(= d(2d + 1))$. Figure 1 shows the (a) first- and (b) second-order GMS-based multipliers over $GF(2^m)$, where a_i and b_j $(0 \le j \le s-1)$ are input shares and c_i is output share of the multiplier. When $m \ge 2$, AND and XOR gates in Fig. 1 denote multiplier and adder over $GF(2^m)$, respectively. Hence, each GMS-based multiplier in Fig. 1 performs $c = a \times b$ where $a = \sum_i a_i$, $b = \sum_j b_j$, and $c = \sum_i c_i$. The GMS-based multipliers are composed of nonlinear, linear, refreshing, and compression

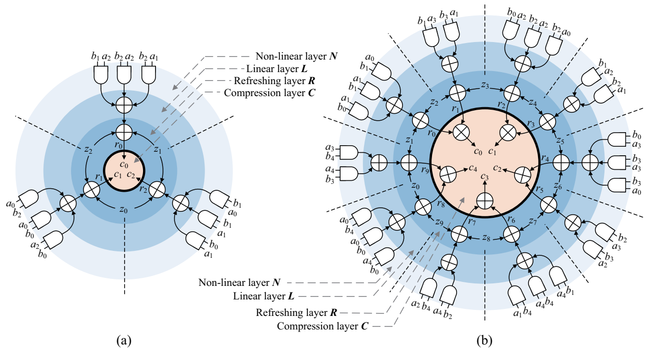

Fig. 1. GMS-based $GF(2^m)$ multipliers in [4]: (a) first- and (b) second-order.

layers and are denoted by $\boldsymbol{N}$, $\boldsymbol{L}$, $\boldsymbol{R}$, and $\boldsymbol{C}$, respectively. Note that the bold line separating $\boldsymbol{R}$ and $\boldsymbol{C}$ denotes registers and s' indicates the number of output shares for $\boldsymbol{R}$ (i.e., r_k).

The nonlinear layer $\boldsymbol{N}$ computes $a_i b_j$ $(0 \le i \le s-1$ and $0 \le j \le s-1)$ using multipliers over $GF(2^m)$. The number of inputs and outputs for $\boldsymbol{N}$ is given by $2s$ and s^2, respectively.

The linear layer $\boldsymbol{L}$ consists of adders to reduce the number of elements from s^2 to s'. The adders should be constructed to satisfy dth-order noncompleteness. In other words, any d outputs of $\boldsymbol{L}$ are independent of at least one input of $\boldsymbol{N}$.

The refreshing layer $\boldsymbol{R}$ adds fresh masks to the outputs of $\boldsymbol{L}$ to meet uniformity[1]. The ring-shaped addition shown in Fig. 1 makes it possible to add fresh masks while maintaining correctness and noncompleteness.

The compression layer $\boldsymbol{C}$ consists of adders to reduce the number of shares from s' to s. Note that, if $s' = s$, the $\boldsymbol{C}$ layer is composed of only wiring as Fig. 1(a). To satisfy dth-order noncompleteness, the registers are located at the boundary between $\boldsymbol{R}$ and $\boldsymbol{C}$.

III. Proposed method and system

In this section, we first present the design and synthesis of GMS-based multipliers based on GF-ACG [6]. Then, we present a formal method for verifying its functionality and security properties. Lastly, we present the overview of our automatic generation system for GMS-based multipliers and demonstrate the performance of our system through experimental generation of GMS-based GF multipliers for different extension degrees and GMS orders.

A. Synthesis of GMS-based multipliers

This section describes the design of GMS-based multipliers through the use of GF-ACG. See [7] for further details.

A GF-ACG G is defined as $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is a set of nodes, and $\boldsymbol{E}$ is a set of directed edges. The node represents an arithmetic circuit (i.e., a module) owing to its functional

[1]In the first-order GMS, $\boldsymbol{R}$ can be omitted if the outputs of $\boldsymbol{L}$ meet the required uniformity criterion. However, since it is known that two-input multiplication has no construction to satisfy the uniformity [3], $\boldsymbol{R}$ is always required for first-order GMS multipliers, as shown in Fig. 1(a). In other words, the uniformity is satisfied by $\boldsymbol{R}$ in the first-order GMS.

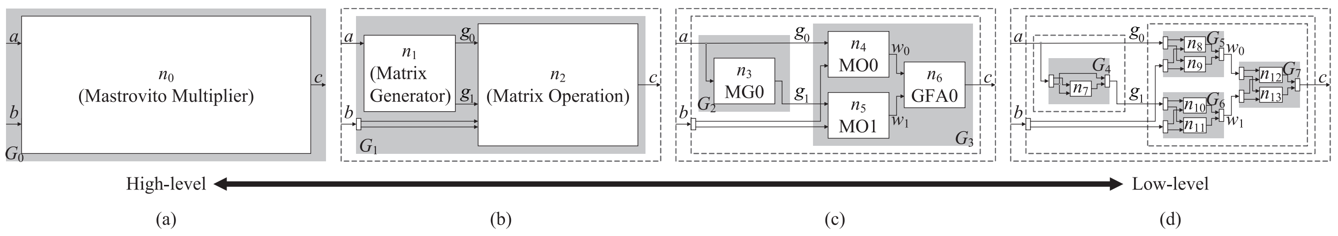

Fig. 2. GF-ACG for $GF(2^2)$ multiplier with four levels of abstraction: (a) top- to (d) lowest-level description.

TABLE I. NODES, GFS, AND VARIABLES IN FIG. 2

Nodes	
[Multiplier] $n_0 = (\{c = ab\}, G_1)$	
[Matrix Generator] $n_1 = (\{g_0 = a\beta^0, g_1 = a\beta^1\}, G_2)$	
[MG0] $n_3 = (\{g_1 = g_0\beta\}, G_4)$	
$n_7 = (\{g_{1,1} = g_{0,0} + g_{0,1}\}, nil)$	
[Matrix Operation] $n_2 = (\{c = \sum_{i=0}^{1} g_i \times (b_i \cdot \beta^{-i}), G_3)$	
[MOi] $n_{i+4} = (\{w_i = g_i \times (b_i \cdot \beta^{-i})\}, G_{i+5}), (0 \leq i \leq 1)$	
$n_{2i+j+8} = (\{w_{i,j} = g_{i,j} b_{i,i}\}, nil), (0 \leq i \leq 1,\ 0 \leq j \leq 1)$	
[GFA0] $n_6 = (\{c = w_0 + w_1\}, G_7)$	
$n_{i+12} = (\{c_i = w_{0,i} + w_{1,i}\}, nil)$	
Galois field	
$GF(2^2) = ((\beta^1, \beta^0), (\{0,1\}, \{0,1\}), \beta^2 + \beta^1 + \beta^0)$	
$GF(2) = ((\beta^0), (\{0,1\}), nil)$	
Galois field variables	
$a, b, c = (GF(2^2), (1,0))$	$b^{(i)} = (GF(2^2), (i,i)\ (0 \leq i \leq 1)$
$b^{(i,i)}, c^{(i)} = (GF(2), (0,0)),\ (0 \leq i \leq 1)$	
$Z_i, g_i, w_i = (GF(2^2), (1,0)),\ (0 \leq i \leq 1)$	
$Z_{i,j}, g_{i,j}, w_{i,j} = (GF(2), (0,0)),\ (0 \leq i \leq 1,\ 0 \leq j \leq 1)$	

assertion and internal structure. The directed edge represents the flow of data between nodes (i.e., a wire), associated with a GF variable. Since GF variable can be represented in multi-levels of abstraction (e.g., bit- and word-level), GF-ACG represents a GF arithmetic circuit in a hierarchical manner using functional assertion and internal structures.

For example, Fig. 2 shows the GF-ACGs for the Mastrovito multiplier over $GF(2^2)$, where GF-ACGs are represented using four levels of abstraction. The nodes in Figs. 2(a), (b), and (c) correspond to the shaded parts in Figs. 2(b), (c), and (d), respectively. Table I shows the nodes, GFs, and variables. Note that the multiplier structure of a larger extension degree is simply extended in accordance with the value of m.

We then design a DPA-resistant multiplier based on the dth-order GMS using GF-ACG. Figure 3 shows a GF-ACG for the GMS-based $GF(2^m)$ multiplier with $d = 1$ and an irreducible polynomial IP, which corresponds to the multiplier of Fig. 1(a). Table II shows nodes, GF, and variables in Fig. 3. Note here that the subscripts of p and z are members of $\mathbb{Z}/s\mathbb{Z}$ and $\mathbb{Z}/s'\mathbb{Z}$, respectively. The variables a_i, b_j, and c_i denote the shares of input and output a, b, and c, respectively (i.e., $a = \sum_i a_i, b = \sum_j b_j$, and $c = \sum_i c_i$ and the multiplier performs $c = a \times b$). Nodes $n_1, n_2, \ldots$, and n_9 denote the multipliers in the $\boldsymbol{N}$. (Note that we can use any type of multipliers for them [6]–[9] in addition to Mastrovito multiplier as shown in Fig. 2.) Nodes n_{10}, n_{11}, and n_{12} denote the adders in the $\boldsymbol{L}$, which are designed to satisfy noncompleteness. Nodes n_{13}, n_{14}, and n_{15} denote the adders in the $\boldsymbol{R}$, where fresh masks z_0, z_1, and z_2 are added to l_0, l_1, and l_2. After the $\boldsymbol{R}$, output shares r_k for the $\boldsymbol{R}$ are stored in registers n_{16}, n_{17}, and n_{18}. Finally, the register outputs are added together to reduce the number of

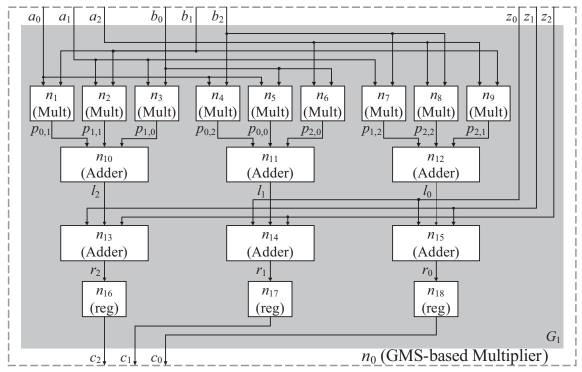

Fig. 3. GF-ACG for GMS-based GF parallel multiplier: top- and second-levels of abstractions.

TABLE II. NODES, GFS, AND VARIABLES IN FIG. 3

Nodes	
[GMS-based Multiplier]	
$n_0 = (\{c_0 = a_1b_2 + a_2b_2 + a_2b_1 + z_1 + z_2,$	
$c_1 = a_0b_2 + a_0b_0 + a_2b_0 + z_0 + z_2,$	
$c_2 = a_0b_1 + a_1b_1 + a_1b_0 + z_0 + z_1\}, G_1)$	
[Mult] $n_1 = (\{p_{0,1} = a_0b_1\}, G_2)$	$n_2 = (\{p_{1,1} = a_1b_1\}, G_3)$
$n_3 = (\{p_{1,0} = a_1b_0\}, G_4)$	$n_4 = (\{p_{0,2} = a_0b_2\}, G_5)$
$n_5 = (\{p_{0,0} = a_0b_0\}, G_6)$	$n_6 = (\{p_{2,0} = a_2b_0\}, G_7)$
$n_7 = (\{p_{1,2} = a_1b_2\}, G_8)$	$n_8 = (\{p_{2,2} = a_1b_1\}, G_9)$
$n_9 = (\{p_{2,1} = a_2b_1\}, G_{10})$	
[Adder] $n_{10+k} = (\{l_k = p_{k+1,k+2} + p_{k+2,k+2} + p_{k+2,k+1}\}, G_{11+k})$	
$n_{13+k} = (\{r_k = l_k + z_{k+1} + z_{k+2}\}, G_{14} + k)$	
[reg] $n_{16+k} = (\{c_k = r_k\}, G_{17+k})$	
Galois field	
$GF(2^m) = ((\beta^{m-1}, \beta^{m-2}, \ldots, \beta^0), (\{0,1\}, \{0,1\}, \ldots, \{0,1\}), IP)$	
$GF(2) = ((\beta^0), (\{0,1\}), nil)$	
Galois field variables	
$a_i, b_j, p_{i,j}, c_k, l_k, r_k, z_k = (GF(2^m), (m-1, 0))$	

shares from s' to s. In Fig. 3, we do not perform addition in the $\boldsymbol{C}$ because $s = s' = 3$. Note that higher-order GMS-based multipliers can be also described in the same manner.

Algorithm 1 synthesizes GF-ACG to represent a GMS-based multiplier from an irreducible polynomial IP and a GMS-order d. Function "$DegreeOf$" in Line 2 obtain the extension degree m from IP. In Line 3, we calculate the numbers of input and output shares (i.e., s and s', respectively). In Lines 4–8, we generate GF-ACGs for submultipliers in the $\boldsymbol{N}$. Function "$GenSubMult$" obtains a GF-ACG $G^{(N)}_{si+j}$ for a submultiplier to calculate a partial product a_ib_j. This function can be implemented using an algorithm in [10]. In Lines 9–19, we generate adders in the $\boldsymbol{L}$. Functions "$GenAdderL0(m, x, y)$" and "$GenAdderL1(m, x, y)$" obtain GF-ACGs $G^{(L)}_k$ for adders that calculate $a_xb_y + a_xb_x + a_yb_x$

Algorithm 1 GF-ACG synthesis

```
Input: Irreducible polynomial $IP$, GMS-order $d$
Output: GF-ACG $G_0 = (\boldsymbol{N_0}, \boldsymbol{E_0})$
 1: Function GF-ACGSYNTHESIS($IP$, $d$)
 2:     int $m \leftarrow DegreeOf(IP)$;
 3:     int $s \leftarrow 2d+1$; int $s' \leftarrow d(2d+1)$;
 4:     for $i$ from 0 to $s-1$ do
 5:         for $j$ from 0 to $s-1$ do
 6:             $G^{(N)}_{si+j} \leftarrow GenSubMult(m, i, j, IP)$;
 7:         end for
 8:     end for
 9:     for $\hat{i}$ from 1 to $s-2$ do
10:         $G^{(L)}_{s\hat{i}} \leftarrow GenAdderL0(m, \hat{i}, 0)$;
11:         for $\hat{j}$ from 1 to $\hat{i}-1$ do
12:             $G^{(L)}_{s\hat{i}+\hat{j}} \leftarrow GenAdderL1(m, \hat{i}, \hat{j})$;
13:         end for
14:     end for
15:     $G^{(L)}_{s'-s} \leftarrow GenAdderL0(m, 0, s-1)$;
16:     $G^{(L)}_{s'-s+1} \leftarrow GenAdderL0(m, s-1, 1)$;
17:     for $\hat{j}$ from 2 to $s-2$ do
18:         $G^{(L)}_{s'-s+\hat{j}+1} \leftarrow GenAdderL1(m, s-1, \hat{j})$;
19:     end for
20:     for $k$ from 0 to $s'-1$ do
21:         $G^{(R)}_{k} \leftarrow GenAdderR(m, k)$;
22:     end for
23:     for $i$ from 0 to $s-1$ do
24:         $G^{(C)}_{i} \leftarrow GenAdderC(m, i)$;
25:     end for
26:     $G_0 \leftarrow GenGMSMult(G^{(\boldsymbol{N})}, G^{(\boldsymbol{L})}, G^{(\boldsymbol{R})}, G^{(\boldsymbol{C})})$;
27:     return $G_0$;
28: end Function
```

and $a_x b_y + a_y b_x$, respectively, where x and y are integers. An adder in the $\boldsymbol{L}$ has two input shares of a and b. Therefore, the d outputs are dependent on at most $2d$ input shares, which indicates that the $\boldsymbol{L}$ would be satisfied with the dth-order noncompleteness. The GF adders are easily synthesized because they consist of bit-parallel XORs. In Lines 20–22, we generate adders in the $\boldsymbol{R}$ and registers for the outputs of the $\boldsymbol{R}$. Function "$GenAdderR$" obtains a GF-ACG $G^{(R)}_k$ for adders to compute $r_k = l_k + z_{k+1} + z_{k+2}$, and a register to store r_k. Lines 23–25 generate adders in the $\boldsymbol{C}$. Function "$GenAdderC$" obtains a GF-ACG $G^{(C)}_i$ for adders to compute $c_i = \sum_{e=0}^{d-1} r_{id+e}$. Lastly, function "$GenGMSMult$" in Line 26 generates a GF-ACG G_0 for the GMS-based multiplier, where $G^{(\boldsymbol{N})}$, $G^{(\boldsymbol{L})}$, $G^{(\boldsymbol{R})}$, and $G^{(\boldsymbol{C})}$ denote the sets of all $G^{(N)}_{si+j}$, $G^{(L)}_k$, $G^{(R)}_k$, and $G^{(C)}_i$, respectively.

B. Functional verification and GMS property checking

The synthesized GF-ACG code can be verified by checking the equivalence of functional assertions and internal structure of each node. In [6], an algebraic method using GB and polynomial reduction technique was shown for this purpose. The algebraic method can perform the equivalence checking through a fully automatic and algebraic procedure [11]. See [6] for details and examples of the verification method.

In this paper, we present a novel formal method for checking whether the GF-ACG satisfies the GMS properties. Algorithm 2 checks the correctness property of GF-ACG for a GMS-based multiplier. The algorithm focuses on the relation between the secret variable a and its shares a_i. (Other variables b and c can also be handled similarly.) In other words, since the correctness of multiplication indicates that the secret variables a, b, and c meet $c = a \times b$, the algorithm verifies whether $c = a \times b$ is derived from the equation of the internal structure, $a = \sum_i a_i$, $b = \sum_j b_j$, and $c = \sum_i c_i$, Function "$MaskingRelation$" in Line 4 derives the above equation representing the relation between secret variable and its shares from the set of edges. Then, the equivalence checking can be performed by means of GB and polynomial reduction in Lines 6 and 7, respectively. Algorithm 2 is less time consuming than the functional verification of GF-ACG because the computation of GB, which is a time-consuming procedure, is required only once. In addition, the proposed method can handle a reduced number of variables for computing GB owing to a hierarchical GF-ACG description.

Algorithm 2 Correctness checking for multiplier

```
Input: A GF-ACG $G_0 = (\boldsymbol{N}_0, \boldsymbol{E}_0)$
Output: $result \in \{true, false\}$
1: Function CHECKCORRECTNESS($n_0 = (\boldsymbol{F}_0, G_1) \in \boldsymbol{N}$)
2:     Bool $result \leftarrow true$;
3:     set $\boldsymbol{M} \leftarrow \{c - ab\}$;
4:     set $\boldsymbol{T} \leftarrow MaskingRelation(\boldsymbol{E}_0)$;
5:     set $\boldsymbol{G} \leftarrow \boldsymbol{F}_0 \cup \boldsymbol{T}$;
6:     set $\boldsymbol{GB} \leftarrow GröbnerBasisOf(\boldsymbol{G})$;
7:     $result \leftarrow IsIdealMember(\boldsymbol{M}, \boldsymbol{GB})$;
8:     return $result$;
9: end Function
```

Algorithm 3 Noncompleteness checking

```
Input: A GF-ACG $G_1 = (\boldsymbol{N}_1, \boldsymbol{E}_1)$, Order $d$
Output: $result \in \{true, false\}$
 1: Function CHECKNONCOMPLETENESS($G_1$)
 2:     set $\boldsymbol{U} \leftarrow \emptyset$; set $\boldsymbol{F}_{all} \leftarrow \emptyset$;
 3:     int $s' \leftarrow d(2d+1)$; Bool $result \leftarrow true$;
 4:     for each $(\boldsymbol{F}_{in}, G_{in}) \in \boldsymbol{N}_1$ do
 5:         $\boldsymbol{F}_{all} \leftarrow \boldsymbol{F}_{all} \cup \boldsymbol{F}_{in}$;
 6:     end for
 7:     set $\boldsymbol{GB} \leftarrow GröbnerBasisOf(\boldsymbol{F}_{all})$;
 8:     for $k$ from 0 to $s'-1$ do
 9:         if $ExtractPolyWithHT(r_k, \boldsymbol{GB}) = nil$ then
10:             $result \leftarrow result \ \& \ false$;
11:         else
12:             $\boldsymbol{U} \leftarrow \boldsymbol{U} \cup \{ExtractPolyWithHT(r_k, \boldsymbol{GB})\}$;
13:         end if
14:     end for
15:     set $\boldsymbol{V} \leftarrow CombinationsOfPolyIn(\boldsymbol{U}, d)$;
16:     for each $\boldsymbol{v} \in \boldsymbol{V}$ do
17:         if $\boldsymbol{v}$ contains all input shares then
18:             $result \leftarrow result \ \& \ false$;
19:         end if
20:     end for
21:     return $result$;
22: end Function
```

Algorithm 3 verifies the dth-order noncompleteness by checking whether any d output shares from the $\boldsymbol{R}$ (i.e., $r_0, r_1, \ldots, r_{s'-1}$) are independent of at least one input share (i.e., a_i or b_j). Noncompleteness can be checked with a symbolic manipulation after a formula manipulation because a GF-ACG represents the relation of input, output, and intermediate variables by GF equations. More precisely, we first compute the relationship between r_k and input shares. In Lines 3–5, we generate a polynomial set $\boldsymbol{F}_{all}$ that contains all the functional assertions of internal nodes for the GMS-based multiplier. Set $\boldsymbol{F}_{all}$ represents its function with intermediate variables $p_{i,j}$, l_k, and r_k. In Line 6, we compute the Gröbner basis corresponding to $\boldsymbol{F}_{all}$ with a lexicographical order $\preceq$ induced by $a_i \preceq b_j \preceq z_k \preceq r_k \preceq p_{i,j} \preceq l_k \preceq c_i$. Thus, in Lines 8–14, we can obtain the relationship between r_k and input shares in accordance with the elimination theorem, where the function "$ExtractPolyWithHT(r_k, \boldsymbol{GB})$" extracts a polynomial whose head term is r_k in $\boldsymbol{GB}$. If we fail to extract such a polynomial the GF-ACG is not satisfied with the dth-

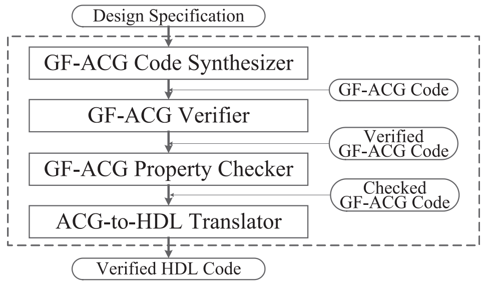

Fig. 4. Overview of proposed system.

order noncompleteness. In Line 15, we create a set $\boldsymbol{V}$ including all d-combinations of polynomials in $\boldsymbol{U}$, where each element is a set of d polynomials. Set $\boldsymbol{V}$ has $\binom{d(2d+1)}{d}$ sets because $\boldsymbol{V}$ consists of s' $(= d(2d+1))$ polynomials according to the number of outputs of the $\boldsymbol{R}$ layer. In Lines 16–20, we check if each element (i.e., set) $\boldsymbol{v}$ of $\boldsymbol{V}$ does not contain all input shares. The time or memory complexity of Alg. 3 increases at least proportionally to the factorial of d because $d(2d+1)$ elements have to be examined.

Finally, we can easily verify the uniformity by looking at the $\boldsymbol{R}$ because an $\boldsymbol{R}$ can guarantee uniformity.

C. Proposed generation system and experimental evaluation

Figure 4 illustrates the overview of our automatic generation system for GMS-based multipliers. When a design specification (i.e., an irreducible polynomial, a GF representation, an arithmetic algorithm, GMS order d) is input, the proposed system generates a GMS-based multiplier whose function and GMS-properties are formally verified. The system consists of (i) GF-ACG code synthesizer, (ii) GF-ACG verifier, (iii) GMS property checker, and (iv) ACG-to-HDL translator. First, the GF-ACG code synthesizer synthesizes the GF-ACG description from the design specification according to Algorithm 1. Then, the GF-ACG verifier formally verifies the synthesized GF-ACG using computer algebra techniques as presented in [6]. In addition, the GF-ACG property checker checks whether it satisfies GMS-properties by using Algorithms 2 and 3. Finally, the ACG-to-HDL translator generates the verified HDL code for the multiplier by translating GF-ACG to HDL code. We can obtain the HDL code corresponding to the GF-ACG by a simple one-to-one mapping since nodes and edges of GF-ACG correspond to modules and wires in HDL, respectively.

The performance of the proposed system is demonstrated through experimental generations of GMS-based multipliers. We measured the processing time of each step in Fig. 4 by generating GMS-based Mastrovito multipliers for different extension degrees m $(= 8, 16, 32, 64, 128, \text{and } 256)$ and GMS orders d $(1 \leq d \leq 5)$. The generation procedures was performed with a computer algebra software Risa/Asir on an Intel Xeon E5450 3.00 GHz processor and 32 GB RAM.

Table III shows the experimental result. Note that the conventional logic simulation cannot verify even the smallest multiplier (i.e., when $m = 8$ and $d = 1$) because its total input size is 72 bits (including fresh masks for the $\boldsymbol{R}$ layer). When $d \leq 4$, our system could generate all the multipliers including the 256-bit ones within 8 min. In the experiment, the verification time was given by $O(m^2)$ because the number of nodes in the GMS-based multiplier increased proportionally to m^2, and each node was verified in less than 1 s. Because the time for functional verification dominated the total generation time, we could significantly reduce it using GF-ACG. On the other hand, when $d \geq 5$, the GMS-property checking dominated the generation time. This is because the computation time of noncompleteness checking by Alg. 3 increases at least proportionally to the factorial of d. However, we confirmed that the proposed system successfully generated such a huge multiplier with $m = 256$ and $d = 5$ in only 15 min.

D. Discussion

The result shows that functional verification occupies a large portion of the generation time when $d \leq 4$, which indicates that the limitation of our system is determined by functional verification. Our system is based on GF-ACG, but other types of verification methods can be used. In earlier related works, functional verification methods were primarily based on decision diagrams (DDs) and binary moment diagrams (BMDs) [12], [13]. However, these methods are basically limited to not only integer arithmetic, but also rather small component circuits such as adders and multipliers. Although there are some DDs for GFs [14], [15], handling practical GFs such as $GF(2^{16})$ and $GF(2^{32})$ is difficult. According to [17], a BDD- (Binary DD-)based method required 1,899.69 s to verify a $GF(2^{16})$ multiplier, and could not verify a $GF(2^{22})$ multiplier in 10 h. In addition to DD-based methods, SAT- and SMT-solvers are also used for the functional verification of arithmetic circuits in some commercial EDA tools. Many SAT-solvers can efficiently solve satisfiability of CNF (Conjunctive Normal Form) formulae, but they have difficulty in verifying GF circuits, which are usually described in an AND-XOR expression. Although a SAT-solver called CryptoMiniSAT has been developed for cryptographic applications, it cannot verify a Mastrovito multiplier over $GF(2^{16})$ in 10 h [17]. Applying SMT-solvers to GF arithmetic is also difficult because SMT-solvers have also been developed basically for integer arithmetic. In addition, such methods require reference circuits (i.e., golden models), which are often not prepared in advance.

On the other hand, recently, algebraic methods based on a GB were also presented in [16]–[18], which verified up to 163-bit GF multipliers effectively without any reference model. However, these methods were only applied to multipliers based on polynomial basis (PB). It seems difficult to apply them to larger and practical circuits commonly used in cryptographic processors (e.g., multipliers over $GF(2^{233})$ and $GF(2^{283})$ in elliptic curve cryptography and circuits based on a normal basis (NB) and redundant representations for compact and efficient implementation). In contrast, the GF-ACG-based method supports various arithmetic algorithms (e.g., full-tree and Massey-Omura) and representations (e.g., composite field, NB, and PRR [6]–[9]) in addition to Mastrovito multipliers used for our experimental evaluation in this paper.

Another discussion point is the applicability of our method. For example, a $GF(2^{128})$ multiplier is used in AES-GCM. Because the multiplication can be targeted by a side-channel attacker [2], there is a high demand for designing tamper-resistant multipliers for AES-GCM, which can be efficiently generated by our system as shown in Table III. In addition,

TABLE III. GENERATION TIME OF GMS-BASED GF MULTIPLIERS (S)

Extension degree m	8					16					32				
GMS order d	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
GF-ACG synthesis	0.07	0.07	0.07	0.07	0.08	0.09	0.09	0.09	0.09	0.10	0.16	0.15	0.16	0.16	0.16
Functional verification	2.97	3.18	3.24	3.67	4.44	4.53	4.73	4.81	5.28	6.22	8.19	8.41	8.49	8.94	9.96
Property checking	0.18	0.18	0.23	4.17	331.96	0.18	0.18	0.24	3.89	330.02	0.18	0.19	0.24	4.11	330.61
GF-ACG to HDL	0.01	0.01	0.01	0.01	0.01	0.03	0.03	0.03	0.03	0.03	0.14	0.14	0.14	0.15	0.15
Total	3.23	3.43	3.56	7.92	336.50	4.82	5.03	5.17	9.29	336.36	8.67	8.90	9.03	13.36	340.88

Extension degree m	64					128					256				
GMS order d	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
GF-ACG synthesis	0.41	0.41	0.41	0.41	0.42	1.48	1.49	1.49	1.50	1.50	5.95	5.94	5.92	5.94	6.02
Functional verification	18.40	18.87	19.01	19.28	21.32	63.76	63.92	64.33	64.94	69.73	405.29	406.63	406.21	407.29	426.09
Property checking	0.19	0.20	0.26	4.03	336.20	0.24	0.24	0.30	3.93	364.15	0.41	0.42	0.48	5.70	406.74
GF-ACG to HDL	0.83	0.85	0.85	0.83	0.85	5.67	5.69	5.63	5.66	5.66	44.71	44.91	44.96	45.17	46.34
Total	19.83	20.33	20.53	24.56	358.78	71.16	71.34	71.76	76.03	441.04	456.35	457.91	457.57	464.10	885.19

we could generate larger multipliers such as $m = 233$ and $m = 283$ that can be used in elliptic curve cryptography, which indicates that our system is also useful for designing tamper-resistant elliptic curve cryptographic processors.

Our system supports GMS-based multipliers whose order is not greater than five. However, the DPA order discussed in many literatures (e.g., [19]) is at most five. As the DPA order increases, attackers find retrieving the key more difficult because of noise. Until now, DPA-leakages have been evaluated with at most fifth-order even in a state-of-the-art laboratory setting [19]. Owing to the abovementioned reasons, the proposed system would be considerably useful in many practical or commercial devices considering tamper-resistance.

Finally, it was shown that side-channel-resistant circuits based on dth-order GMS could be constructed with $d + 1$ shares at minimum, which would be useful for designing more compact circuits than conventional ones with $dt + 1$ shares [20]. Our system would be extended to multipliers with $d+1$ shares, while our system would easily synthesize and verify the multipliers in the manner similar to the above.

IV. CONCLUSION

This paper presented a formal design and verification system for GMS-based GF parallel multipliers. In this paper, we first presented a formal design methodology of GMS-based GF arithmetic circuits based on GF-ACG, and demonstrated an efficient verification method for both functionality and security property based on Gröbner basis. In addition, we proposed an automatic generation system for GMS-based GF multipliers, which can synthesize a fifth-order 256-bit multiplier (whose input bit-length is 256×40) within a practical time. The proposed system generates HDL codes whose function and GMS-property are completely verified through only design specification. An extension of our system to other functions (e.g., GF inversion) would remain in future work.

ACKNOWLEDGMENT

We are grateful to Mr. Yukihiro Sugawara for his corporation. This work has been supported by JSPS KAKENHI Grants No. 25240006 No. 15KK0001.

REFERENCES

[1] P. Kocher, "Differential power analysis," *CRYPTO*, 1999, pp. 388–397.

[2] S. Belaïd, J.-S. Coron, P.-A. Fouque, B. Gérard, J.-G. Kammerer, and E. Prouff, "Improved side-channel analysis of finite-field multiplication," *CHES*, 2015, pp. 395–415.

[3] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glithces," *J. Cryptology*, vol. 24, pp. 292–321, 2011.

[4] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating masking schemes," *CRYPTO*, 2015, pp. 764–783.

[5] E. Biham, Y. Carmeli, and A. Shamir, "Bug attacks," *CRYPTO*, 2008, pp. 221–240.

[6] N. Homma, K. Saito, and T. Aoki, "A formal approach to designing cryptographic processors based on $GF(2^m)$ arithmetic circuits," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 3–13, Feb. 2012.

[7] N. Homma, K. Saito, and T. Aoki, "Toward formal design of practical cryptographic hardware based on Galois field arithmetic," *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2604–2613, Jun. 2014.

[8] K. Okamoto, N. Homma, and T. Aoki, "A graph-based approach to designing parallel multipliers over Galois fields based on normal basis representations," *ISMVL*, 2013, pp. 54–59.

[9] R. Ueno, N. Homma, Y. Sugawara, and T. Aoki, "Formal design of Galois-field arithmetic circuits based on polynomial ring representation," *ISMVL*, 2015, pp. 48–53.

[10] Y. Sugawara, R. Ueno, N. Homma, and T. Aoki, "System for automatic generation of parallel multipliers over Galois field," *ISMVL*, 2015, pp. 54–59.

[11] D.A. Cox, J.B. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, 2nd ed. NY: Springer-Verlag, 1996, 536 pages.

[12] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[13] R.E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," *DAC*, 1995, pp. 535–541.

[14] R. Stankovic and R. Drechsler, "Circuit design from Kronecker Galois field decision diagrams for multiple-valued functions," *ISMVL*, 1997, pp. 275-280.

[15] A.M. Jabir and D.K. Pradhan, "A graph-based unified technique for computing and representing coefficients over finite fields," *IEEE Trans. Computers*, vol. 56, no. 8, pp. 1119–1132, Aug. 2007.

[16] J. Lv, P. Kalla, and F. Enescu, "Verification of composite Galois field multipliers over $GF((2^m)^n)$ using computer algebra techniques," *HLDVT*, 2011, pp. 136–143.

[17] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner basis reductions for formal verification of Galois field multipliers," *DATE*, 2012, pp. 899–904.

[18] J. Lv and P. Kalla, "Formal verification of Galois field multipliers using computer algebra techniques," *VLSID*, 2012, pp. 388–393.

[19] A. Moradi and W. Alexander, "Assessment of hiding the higher-order leakages in hardware—What are the achievements versus overheads?" *CHES*, 2015, pp. 453–474.

[20] T. D. Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen, "Masking AES with $d + 1$ shares in hardware," *CHES*, 2016, pp. 194–212.