Formal Approach for Verifying Galois Field Arithmetic Circuits of Higher Degrees

Rei Ueno, *Student Member, IEEE*, Naofumi Homma, *Member, IEEE*,
Yukihiro Sugawara, and Takafumi Aoki, *Member, IEEE*

Abstract—This paper presents an efficient approach to verifying higher-degree Galois-field (GF) arithmetic circuits. The proposed method describes GF arithmetic circuits using a mathematical graph-based representation and verifies them by a combination of algebraic transformations and a new verification method based on natural deduction for first-order predicate logic with equal sign. The natural deduction method can verify one type of higher-degree GF arithmetic circuit efficiently while the existing methods require an enormous amount of time, if they can verify them at all. In this paper, we first apply the proposed method to the design and verification of various Reed-Solomon (RS) code decoders. We confirm that the proposed method can verify RS decoders with higher-degree functions while the existing method needs a lot of time or fail. In particular, we show that the proposed method can be applied to practical decoders with 8-bit symbols, which are performed with up to 2,040-bit operands. We then demonstrate the design and verification of the Advanced Encryption Standard (AES) encryption and decryption processors. As a result, the proposed method successfully verifies the AES decryption datapath while an existing method fails.

Index Terms—Formal verification, design methodology for arithmetic circuits, Galois field, advanced encryption standard, computer algebra, arithmetic logic

1 Introduction

Applications of cryptography and Error Correction Codes (ECCs) based on arithmetic operations over Galois fields (GFs) have been rapidly increasing because of high demands for reliable and secure communications and transactions [1]. In the Internet of Things (IoT) era, hardware implementations for these operations are essential to embedded systems with limited resources such as smart cards. However, the existing design environments do not support the high-level synthesis of GF arithmetic circuits. This lack of high-level design methodology forces designers to describe the structural details of GF arithmetic circuits in massive AND-XOR expressions by hand. As a result, it is difficult to design, optimize, and debug practical GF arithmetic algorithms using conventional register transfer level (RTL) design. It is difficult to apply such a high level design methodology to integer ring arithmetic circuits for GF arithmetic circuits because GF arithmetic operations vary depending on the modular polynomial. In addition, the functional verification of such GF arithmetic circuits is even more critical. A conventional RTL simulation cannot cover all the inputs of practical cryptographic and ECC processors because they are usually performed with operands that are longer than 64 bits. The test pattern generation of GF arithmetic itself is also much more difficult than that of the familiar integer arithmetic.

In earlier related research, formal verification was primarily performed based on Decision Diagrams (DDs) and Binary Moment Diagrams (BMDs) [2], [3], [4]. However, these approaches are basically limited not only to binary arithmetic over the integer ring, but also to rather small component circuits such as adders and multipliers. Although there are some DDs for GFs based on the decomposition of multiple-valued functions [5], [6], it is difficult to handle practical GFs such as $GF(2^{16})$ and $GF(2^{32})$. This limitation leads to difficulty applying them to sophisticated functions. To make matters worse, formal verification based on DDs or BMDs requires reference circuits (i.e., golden models), which often have not been prepared for sophisticated cases.

As a pioneering work, a high-level description and verification method for arithmetic circuits over GFs was proposed in [7], [8]. The proposed method uses a high-level mathematical graph based on variables and arithmetic equations over GFs, which is called the GF Arithmetic Circuit Graph (GF-ACG). Using GF-ACGs, we can describe any GF arithmetic circuit in a hierarchical manner as a combination of arithmetic sub-circuits (i.e., sub-graphs). A circuit description is formally verified by checking whether its function is satisfied under the set of specification functions of the subcircuit components. This verification is done without any reference model. In [7], an automatic algebraic verification procedure based on a Gröbner basis (GB) and polynomial reduction technique [9] was shown. This procedure can be applied to any kind of circuit description. Verification of 256-bit GF multipliers and a 128-bit Advanced Encryption Standard (AES) encryption datapath using the GF-ACG representation and algebraic techniques was also shown in [7], [8]. However, the

• *The authors are with Tohoku University, 6-6-05 Aramaki Aza Aoba, Aoba-ku, Sendai-shi 980-8579, Japan.*
E-mail: {ueno, homma, yukihiro}@aoki.ecei.tohoku.ac.jp, aoki@ecei.tohoku.ac.jp.

Manuscript received 18 Nov. 2015; revised 3 July 2016; accepted 12 Aug. 2016. Date of publication 28 Aug. 2016; date of current version 16 Feb. 2017. Recommended for acceptance by E. Savas.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2603979

verification time increases rapidly depending on the algebraic degree and number of variables in the circuit functions. (Note here that the algebraic degree is different from the extension degree used in the function.) The drawbacks of algebraic verification have made it difficult to verify entire cryptographic and ECC processors, which have high-degree functions such as complex functions. For example, the conventional GB-based method has difficulty in verifying Reed-Solomon (RS) decoders based on $GF(2^8)$ arithmetic (e.g., an ECC used for CD), whose algebraic degree is more than 1,000. Moreover, AES encryption and decryption, which are also based on $GF(2^8)$ arithmetic, are also represented by functions with the algebraic degree of more than 254. In particular, the conventional method also has difficulty in verifying the AES decryption datapath because the AES decryption function contains multivariate polynomials of algebraic degree of 508.

Other types of algebraic verification methods were also proposed in [10], [11], [12]. In [10], a hierarchical verification method was applied to multipliers over $GF((2^m)^n)$. In [11], an extended method for the variable ordering was presented to perform the computation of a GB in Buchberger's algorithm in an efficient manner. In [12], the GB was calculated without using Buchberger's algorithm by a term order derived from a circuit topology analysis. As a result, a multiplier over $GF(2^{163})$ was verified effectively. These methods [10], [11], [12] were successfully applied to GF multipliers. On the other hand, the applications to the larger and tower-field multipliers commonly used in ECC and cryptographic processors (e.g., the multiplicative inversion on $GF(((2^2)^2)^2)$ in AES and multiplier over $GF(2^{233})$ and $GF(2^{283})$ in elliptic curve cryptography) were not represented. In addition, an application of such methods to the whole of ECC and cryptographic or ECC processor datapaths with numerous variables and high-degree functions has not been discussed in the literature.

To address the above problem, this paper presents an efficient approach to verifying high-degree GF arithmetic circuits based on GF-ACG. The basic idea is to combine the conventional algebraic method with a new method based on natural deduction for predicate logic. The verification time of the new method does not depend on the degree of the function. Using this feature, the proposed method efficiently verifies GF arithmetic circuits of higher degrees than attempted before. While the idea of the new method and a preliminary evaluation were presented in [13], in this paper, we enhance our method significantly. First, we improve and generalize the verification algorithm based on natural deduction so that it may be applied to a wider range of applications. The major contribution is that we newly propose the entire verification process, combining the algebraic and natural deduction methods in a systematic manner. The efficiency of the natural-deduction method is also newly evaluated using accumulators with inversion circuits. The advantage of the proposed method is then demonstrated through the design and verification of ECC and cryptographic processors. In particular, we show the design and verification of the AES encryption and decryption datapaths in addition to the ECC decoders. To the best of our knowledge, this is the first result in which the AES decryption datapaths are successfully verified in a formal manner.

2 GF-ACG and the Computer Algebra-Based Verification Method

2.1 GF-ACG

This section briefly describes the graph-based representation of GF arithmetic circuits, which are referred to as GF-ACGs.

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 by its functional assertion and internal structure. The directed edge represents the flow of data between nodes and defines the data dependency. We assume that every node has at least one edge connection.

A node (each element of $\boldsymbol{N}$) is defined by $(\boldsymbol{F}, G')$, where F is the functional assertion, given as a set of equations over GFs (*GF equations*), and G' is the internal structure, given as a lower-level GF-ACG. A node at the lowest level of abstraction, which does not have this internal structure, is described as $(\boldsymbol{F}, nil)$. A functional assertion is represented as a relation $F_l = F_r$, where F_l and F_r are the output and input expressions, respectively, and each expression is given by variables, constants or combinations of two or more of these expressions connected by arithmetic operations $+$, $-$, and $\times$.

A directed edge (each element of $\boldsymbol{E}$) is defined as (n_s, n_d, v), where n_s and n_d represent the start and end nodes, respectively, and v represents the variables indicating a variable of GF. If either n_s or n_d is nil, the directed edge represents an external input or output for the given GF-ACG. Each variable is associated with a Galois field. A Galois field GF is represented as $(\boldsymbol{B}, \boldsymbol{C}, IP)$, where $\boldsymbol{B}$ denotes the basis, $\boldsymbol{C}$ is the coefficient vector, and IP is the irreducible polynomial. More precisely, $\boldsymbol{B}$, $\boldsymbol{C}$, and IP are given as

$$\boldsymbol{B} = \left(\beta^{m-1}, \beta^{m-2}, \ldots, \beta^0\right), \tag{1}$$

$$\boldsymbol{C} = (C_{m-1}, C_{m-2}, \ldots, C_0), \tag{2}$$

$$IP = \beta^m + c_{m-1}\beta^{m-1} + \cdots + c_0\beta^0, \tag{3}$$

where β is the indeterminate element, C_i is the coefficient set of degree i, m is the degree of field extension, and c_i is the element of the coefficient set C_i. When GF represents a prime field, we let IP be nil. Thus, the above description can handle both prime and extension fields. Let h $(0 \leq h \leq m-1)$ and l $(0 \leq l \leq h)$ be the most and least significant degrees, respectively. A variable is represented as $v = (GF, (h, l))$, where the tuple (h, l) is called the degree range. Using the above notation, we handle a specific variable v_d of degree d.

A variable can be decomposed/composed into an expression with sub-variables at a lower/higher level of abstraction. Using the decomposition/composition nodes, we can change the level of abstraction in the edge representation. These nodes are implemented by wiring and have no internal structures. GF-ACG can be used also for representing any logic circuit. We refer the reader to [8] for more details.

For example, Fig. 1 shows the GF-ACGs for the conventional Mastrovito multiplier over $GF(2^2)$, where the GF-ACGs are represented by four levels of abstraction. Here, an

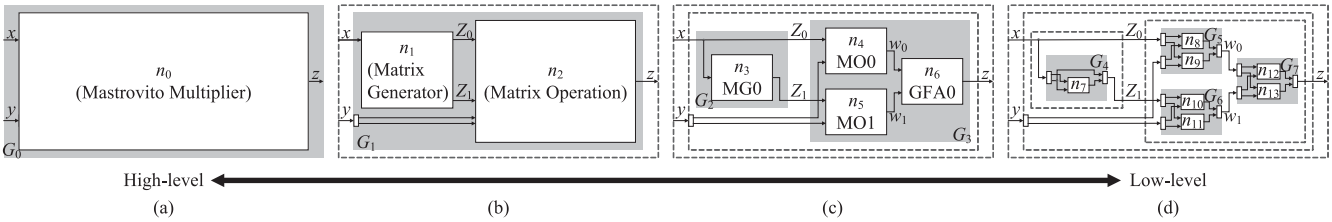


Fig. 1. GF-ACGs for $GF(2^2)$ Mastrovito Multiplier: (a)-(d) GF-ACGs at four levels of abstraction from high to low.

example of $GF(2^2)$ multiplier is presented to show the hierarchical structure for simplicity. Note that the multiplier structure of a larger extension degree (e.g., $m = 163$ or $m = 233$ used in elliptic curve cryptography) is simply extended in accordance with the value of m. The nodes in Figs. 1a, 1b, and 1c correspond to the shaded parts in Figs. 1b, 1c, and 1d, respectively. Table 1 shows the nodes, GFs, and variables. Note that the decomposition/composition nodes are not shown in Table 1. Fig. 1a shows the top-level node that represents whole of the multiplier. Fig. 1b shows the second-level of abstraction that consists of a Mastrovito matrix generator and operator, and Fig. 1c shows the internal structure of the generator (i.e., G_2) which consists of a constant multiplier represented by n_3 and, that of the operator (i.e., G_3) which consists of partial product generators (i.e., n_4 and n_5) and an accumulator (i.e., n_6). Finally, Fig. 1d shows the lowest-level of abstraction that represents the logic circuit corresponding to the multiplier. Note again that Mastrovito multipliers for extension degrees are described in the same manner.

2.2 GB-Based Verification

The functions of GF arithmetic circuits, hierarchically given by GF-ACGs, can be verified by checking the equivalence of the functional assertions and internal structure. In [8], an algebraic method using the GB and a polynomial reduction technique was shown for this purpose. The basic idea of verification is to solve an ideal membership problem instead of equivalence checking. In other words, function f ($\in \boldsymbol{F}$) is correctly derived from its internal structure if f is included in the solution space (i.e., ideal) given by a set of polynomials (sub-functions) of internal structures. The GB and polynomial reduction are used for solving the ideal membership problem in a systematic manner [9]. In summary, the GB-based verification method is formally described as follows:

(Step 1) Describe the target circuit in the form of GF-ACG $(\boldsymbol{N}, \boldsymbol{E})$.

(Step 2) For each node n given by $(\boldsymbol{F}, G) \in \boldsymbol{N}$, generate an ideal $\boldsymbol{I}$ from the polynomial set of functional assertions in G, and prove $\boldsymbol{F} \subset \boldsymbol{I}$.

Algorithm 1 shows the GB-based verification method used in Step 2. In Lines 2–4, we derive a set of polynomial $\boldsymbol{P}$ from an internal structure, and function "GröbnerBasisOf(P)" in Line 5 transforms $\boldsymbol{P}$ to the corresponding Gröbner basis $\boldsymbol{GB}$ by using Buchberger's algorithm [14]. Note that $\boldsymbol{GB}$ generates an ideal $\boldsymbol{I}$ equal to that of $\boldsymbol{P}$. ($\boldsymbol{GB}$ becomes an unit set {1} if $\boldsymbol{P}$ has no root. Because this means that the node includes one or more bugs, the algorithm returns false in such a case.) In Line 7, we calculate the normal form $NF_{\boldsymbol{GB}}(f)$ ($f \in \boldsymbol{F}$), which is given by reducing f by $\boldsymbol{GB}$. This procedure is called polynomial reduction, which is an extension of modulo to multivariate polynomials. $NF_{\boldsymbol{GB}}(f)$ becomes 0 if and only if f is included in the ideal $\boldsymbol{I}$ generated by $\boldsymbol{GB}$. Thus, we can verify the correctness of any target function from the normal form. Note that the computation of the GB and normal form always terminates in a finite number of steps. This means that this verification algorithm can be applied to any kind of node functions in GF-ACG. See [8] for more details and examples of the verification method.

Algorithm 1. GB-Based Verification Method

Input: Node $(\boldsymbol{F}, G)$
Output: Verification result $res \in \{\text{true}, \text{false}\}$
1: **Function** GB-BasedMethod$((\boldsymbol{F}, G))$
2: **set** $\boldsymbol{P} \leftarrow \emptyset$; **Bool** $res \leftarrow$ true;
3: **for** each $(\boldsymbol{F}', G') \in \boldsymbol{N} (\in G)$ **do**
4: $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \boldsymbol{F}'$;
5: **end for**
6: $\boldsymbol{GB} \leftarrow$ GröbnerBasisOf$(\boldsymbol{P})$;
7: **for** each $f \in \boldsymbol{F}$ **do**
8: **if** $((NF_{\boldsymbol{GB}}(f) = 0)$ & $(\boldsymbol{GB} \neq \{1\}))$ **then**
9: $res \leftarrow res$ & true;
10: **else then**
11: $res \leftarrow res$ & false;
12: **end if**
13: **return** res;
14: **end**

TABLE 1
Nodes, GFs and GF Variables in Fig. 1

Nodes
[Mastrovito Multiplier] $n_0 = (\{z = x \times y\}, G_1)$
[Matrix Generator] $n_1 = (\{Z_0 = x \cdot \beta^0, Z_1 = x \cdot \beta^1\}, G_2)$
[MG0] $n_3 = (\{Z_1 = Z_0 \times \beta\}, G_4)$
$n_7 = (\{Z_{1,1} = Z_{0,0} + Z_{0,1}\}, nil)$
[Matrix Operation] $n_2 = (\{z = \sum_{i=0}^{1} Z_i \times (y_i \cdot \beta^{-i}), G_3)$
[MOi] $n_{i+4} = (\{w_i = Z_i \times (y_i \cdot \beta^{-i})\}, G_{i+5}),\ (0 \le i \le 1)$
$n_{2i+j+8} = (\{w_{i,j} = Z_{i,j} \times y_{i,i}\}, nil),$
$(0 \le i \le 1,\ 0 \le j \le 1)$
[GFA0] $n_6 = (\{z = w_0 + w_1\}, G_7)$
$n_{i+12} = (\{z_i^p = w_{0,i} + w_{1,i}\}, nil),\ (0 \le i \le 1)$
GFs
$GF(2^2) = ((\beta^1, \beta^0), (\{0,1\}, \{0,1\}), \beta^2 + \beta^1 + \beta^0)$
$GF(2) = ((\beta^0), (\{0,1\}), nil)$
GF variables
$x, y, z = (GF(2^2), (1,0))$
$y_i = (GF(2^2), (i,i)),\ (0 \le i \le 1)$
$y_{i,i}, z_i = (GF(2), (0,0)),\ (0 \le i \le 1)$
$Z_i, w_i = (GF(2^2), (1,0)),\ (0 \le i \le 1)$
$Z_{i,j}, w_{i,j} = (GF(2), (0,0)),\ (0 \le i \le 1,\ 0 \le j \le 1)$

$$\frac{}{\tau = \tau} \qquad \frac{P[\tau_0/var] \quad \tau_0 = \tau_1}{P[\tau_1/var]}$$

(a) (b)

Fig. 2. Proof figures for inference rules of proposed method: (a) axiom of equal sign, and (b) Rule of equal sign.

The computation time of GB-based method depends on the number of variables and/or degree of equations. For example, the GB calculation is close to Gaussian elimination if the set of polynomials consists of linear polynomials. In this case, the computation cost becomes $O(\kappa^3)$, where κ is the number of variables. Here, the hierarchical description of GF-ACG helps reduce the number of variables belonging to a polynomial. As a result, the verification time (i.e., computation of GBs) is reduced significantly. Therefore, the algebraic method is especially effective for verifying linear or low-degree functions.

On the other hand, the GB-based method sometimes takes a long time to compute the GB and normal form if the polynomials have high degrees. In addition, the GB-based method requires us to handle all polynomials in the expanded form. As a result, the GB-based method has difficulty verifying functions that have a high-degree polynomial with a large number of terms (e.g., $(\sum_{i=0}^{n-1} a_i)^{254}$) because of the memory limitations.

3 Proposed Verification Method

3.1 ND-Based Verification

A major advantage of GF-ACG is that we can describe sophisticated datapaths, such as those found in ECC and cryptographic processors. However, they sometimes include high-level nodes with higher-degree arithmetic functions (e.g., exponentiation of polynomials). GB-based verification has difficulty handling such functions, as mentioned above. On the other hand, the functional assertion of such nodes is often given as a simple equation, substituting input variables for internal structure (function). This is because many higher-level functions of ECC and cryptography, which consist of several sub-functions, are implemented by simple serial and/or parallel combinations of sub-functions. The correctness of such substitution can be verified deductively without algebraic manipulation. In other words, we use only substitution to verify whether the connection of the internal nodes is valid for the function.

To verify nodes whose functional assertions are efficiently given by substitutions alone, we introduce a natural deduction technique for first-order predicate logic with equal sign [15]. Accordingly, we treat the equations (i.e., functional assertions and simultaneous equations of internal structure) as symbol strings in order to apply the inference rules of natural deduction, which derives an equation from other equations. In predicate logic, if formula A can be derived from a set of formulae $\boldsymbol{T}$ by applying inference rules a finite number of times, it can be said that "A is provable from $\boldsymbol{T}$." This means that A is correct if all of the elements of $\boldsymbol{T}$ are correct. Using the proof method, we solve simultaneous algebraic equations of mathematical substitutions by natural deduction. In general, we generate a proof figure to give the natural deduction proof. The proof figure has a tree structure consisting of formulae, and shows that

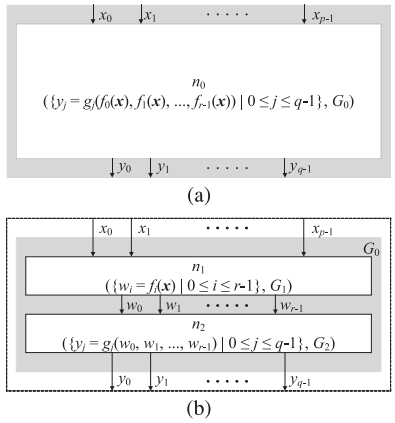


Fig. 3. Example of GF-ACG for ND-based method: (a) top-level and (b) second-level description.

the formula at the bottom (i.e., the conclusion) is provable from the formulae at the top (i.e., the assumptions). We apply inference rules repeatedly to the formulae from the bottom up in order to generate such a proof figure. The inference rules themselves are given as a proof figure. Fig. 2 shows proof figures for the two inference rules of natural deduction used in the proposed method. The set of rules are equivalent to equality axioms that give the attributes of the general equal sign. Fig. 2a shows one inference rule, that $\tau = \tau$ for any term τ is true without any assumption or/and any proof. Fig. 2b shows the other inference rule, which defines mathematical substitution, where $P[\tau_0/var]$ and $P[\tau_1/var]$ indicate the formulae P where every variable var is replaced by term τ_0 and term τ_1, respectively. The rule illustrated in Fig. 2b says that an equation $P[\tau_1/var]$ is provable from two equations, $P[\tau_0/var]$ and $\tau_0 = \tau_1$. The natural deduction system with these two rules guarantees that the conclusion at the bottom of the proof figure is true if all of the assumptions are true.

The new ND-based method treats a functional assertion (an equation) and internal structure (i.e., simultaneous equations) as a "conclusion" and "assumptions" in the proof figure, respectively. When we successfully generate a proof figure that has the functional assertion at the bottom and consists of inference rules and/or assumptions at the higher layers, we can say that the node function is correctly derived from the internal structure.

Note here that the new method exploits arithmetic logic and essentially differs from the existing verification methods based on DDs. Our new method describes predicates (i.e., GF equations) by word-level variables whose domain consists of elements of GF and gives a proof by the application of predicates (i.e., inference rules), while DD-based methods describe a circuit specification by using logic variables and quantifiers $\forall$ and $\exists$ and check the equivalence between target and reference circuits. Thus, the new method is easily applied to GF-ACGs.

Fig. 3 shows an example of GF-ACG which is suitable for the new verification method. Let n_0 in Fig. 3a be a node that has p inputs and q outputs. The functional assertion is given by

$$y_i = g_i(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{r-1}(\boldsymbol{x})), \ 0 \le i \le q-1, \qquad (4)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{y_j = y_j} \quad y_j = g_j(w_0, w_1, \ldots, w_{r\text{-}1})}{y_j = g_j(w_0, w_1, \ldots, w_{r\text{-}1})} \quad w_0 = f_0(\boldsymbol{x})}{y_j = g_j(f_0(\boldsymbol{x}), w_1, \ldots, w_{r\text{-}1})} \quad w_1 = f_1(\boldsymbol{x})}{y_j = g_j(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, w_{r\text{-}1})} \quad w_{r\text{-}1} = f_{r\text{-}1}(\boldsymbol{x})}{y_j = g_j(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{r\text{-}1}(\boldsymbol{x}))}$$

Fig. 4. Proof figure verification of n_0 in Fig. 3.

where $\boldsymbol{x}$ represents p variables $x_0, x_1, \ldots, x_{p-1}$, and $f_0, f_1, \ldots, f_{r-1}$ represent functions of p variables, respectively. In addition, let n_1, n_2 in Fig. 3b be nodes of the internal structure of n_0 with functional assertions

$$w_i = f_i(\boldsymbol{x}),\ 0 \le i \le r-1, \tag{5}$$

$$y_j = g_j(w_0, w_1, \ldots, w_{r-1}),\ 0 \le j \le q-1, \tag{6}$$

respectively. Fig. 4 shows a proof figure to verify the functional assertion of n_0, which is generated by the verification method based on natural deduction. We generate such proof figures for all j $(0 \le j \le q-1)$. The double line in Fig. 4 represents the application of the same inference rules more than once. Such a proof figure is certainly generated if the function is correct and the simultaneous algebraic equations are given as mathematical substitutions. Fig. 3 illustrates an example where the internal structure is composed of two nodes connected in series, but the verification method can be applied to any internal structure if the simultaneous equations are given as mathematical substitutions.

Algorithm 2 shows the proposed ND-based method, where the node given by $(\boldsymbol{F}, G)$ is the input and the verification result $res \in \{\text{true, false}\}$ is the output. The function "GetEquations" in Line 1 obtains a list of equations (i.e., functional assertions) that forms the internal structure G. The function "SortEquationsByLength" in Line 2 sorts the obtained equations by the length of rhs in descending order to perform the following substitutions uniquely. Using the sorted list, we then apply the inference rule in Fig. 2b repeatedly to each equation in $\boldsymbol{F}$. The function "IsSubstitutable" in Line 6 examines whether the ith element of L can be substituted for any term in S. If it is possible, the function "Substitution" at line 7 performs mathematical substitution by using the rule of equal sign, where the output S, the inputs S, and $L[i]$ correspond to $P[\tau_0/var]$, $P[\tau_1/var]$, and $\tau_0 = \tau_1$ in Fig. 2b, respectively. After the substitution, the equation used in "Substitution" is deleted from list L' since an equation is never used again to give proof of an equation and the value of i is reset to examine the substituted equation again from the first equation in L'. After all substitutions, the function "IsAxiom" in Line 13 examines whether the obtained equation is the same as axiom $(\tau = \tau)$. Finally, we return true if and only if all the equations in $\boldsymbol{F}$ are transformed to the axiom.

The verification time of this algorithm is basically proportional to the number of performed substitution operations, that is, the number of equations that comprise the functional assertion and the internal edges in the internal structure Note that each substitution operation is performed by a symbolic computation. Therefore, the verification method is efficient when the GF-ACG has a higher-degree function given by substitutions.

Algorithm 2. ND-Based Verification Method

Input: Node $(\boldsymbol{F}, G)$
Output: Verification result $res \in \{\text{true, false}\}$
1: **Function** ND-BasedMethod($(\boldsymbol{F}, G)$)
2: **list** $L \leftarrow$ GetEquations(G); **Bool** $res \leftarrow$ true;
3: $L \leftarrow$ SortEquationsByLength(L);
4: **for** each $f \in \boldsymbol{F}$ **do**
5: **str** $S \leftarrow f$; **int** $i \leftarrow 0$; **list** $L' \leftarrow L$;
6: **while** $i <$ length(L') **do**
7: **if** IsSubstitutable($S, L'[i]$) = true **then**
8: $S \leftarrow$ Substitution($S, L'[i]$);
9: **delete** $L'[i]$; $i \leftarrow 0$;
10: **else then**
11: $i \leftarrow i + 1$;
12: **end if**
13: **end while**
14: $res \leftarrow res$ & IsAxiom(S)
15: **end for**
16: **return** res;
17: **end**

The algorithm stops when "IsSubstitutable($S, L'[i]$) = false" for every equation. The termination condition of this algorithm is that the internal structure (i.e., GF-ACG) does not include directed cycle graphs, that is, substitutions are not repeated more than once. This property is also implemented by **delete** $L'[i]$ on Line 8. The "false" suggests that the number of equations obtained from the internal structure is finite. In addition, Algorithm 2 simply consists of substitution operations to examine a proof figure. The correctness of Algorithm 2 is given from the proven soundness of the natural deduction system [15], which means that the conclusion (i.e., functional assertion) is correct if the assumptions (i.e., internal structure) are correct in the proof figure.

The above algorithm returns false if the node is wrong or the proof must consider arithmetic rules (e.g., addition and multiplication) because it does not exploit arithmetic rules and other inference rules of natural deduction (e.g., rules about $\vee, \wedge$ and $\supset$). In other words, the algorithm sometimes returns false even if the circuit is correct when the function is not derived deductively. Therefore, it is better to apply the ND-based method only to nodes derived deductively and those that would required to take an enormous amount of time to verify by the GB-based method.

In its preliminary version [13], the ND-based method was first applied to each node and the GB-based method was then applied to the nodes where the ND-based method returns false. However, it is difficult to verify a high-degree function by such an approach when the node includes bugs. Nodes that have a high-degree function should not be input to the GB-based method.

3.2 GB-ND Hybrid Method

Considering the above condition, we propose an entire verification flow based on the two different methods (i.e., GB-based and ND-based methods) that was not discussed in the preliminary version in [13]. We first divide arithmetic functions into seven layers to distinguish functions derived deductively, and then apply the ND-based method to functions belonging to the top two layers and the GB-based

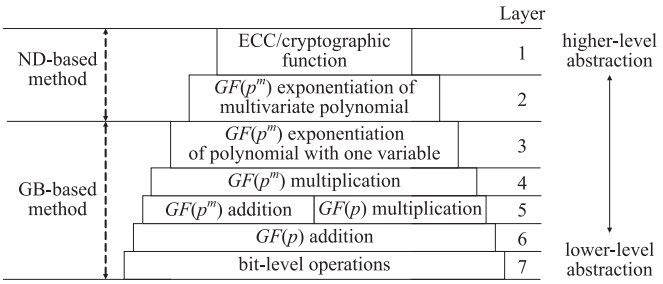


Fig. 5. Hierarchical structure of arithmetic functions.

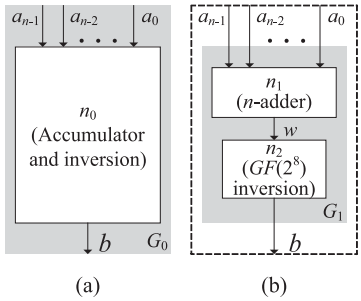


Fig. 6. GF-ACG for circuit connecting n-input adder and inversion in serial.

method to the rest. Fig. 5 shows the hierarchical structure of arithmetic functions that is commonly given by GF-ACG. This hierarchy consists of the ECC/cryptographic function, $GF(p^m)$ exponentiation of multivariate polynomials, $GF(p^m)$ exponentiation of polynomials with one variable, $GF(p^m)$ multiplication, $GF(p^m)$ addition, $GF(p)$ multiplication, $GF(p)$ addition, and bit-level operations from higher to lower levels of abstraction. The top layer includes the node for the whole ECC/cryptographic processor function and the second layer includes the nodes for exponentiation of multivariate polynomials. These functions are necessarily implemented by simple serial or/and parallel combinations of subfunctions (e.g., adders and multipliers). Therefore, such functions can be derived deductively and are suitable for the ND-based method. Note again that the GB-based method has difficulty verifying functions such as those that belong to the top two layers since these functions usually have higher degrees. On the other hand, the rest of the functions, which belong to the lower layers, perform basic arithmetic operations, and most of them are low-degree functions even though there are many implementations of such functions. In other words, most of them are not derived deductively. Therefore, these functions should be verified by the GB-based method. In summary, our design method consists of three steps, as follows:

(Step 1) Describe the target circuit in the form of GF-ACG $(\boldsymbol{N}, \boldsymbol{E})$.

(Step 2) For each node n given by $(\boldsymbol{F}, G) \in \boldsymbol{N}$, determine which verification method should be applied.

(Step 3) Verify each node by either the GB-based or ND-based method.

In step 1, it is essential to describe the circuit hierarchically according to Fig. 5 to ensure fast verification.

Algorithm 3 shows the entire verification flow used in Steps 2 and 3, which is obtained by extending the conventional algorithm in [7] for combining the GB- and ND-based formula evaluation methods. Given a GF-ACG, the formula evaluation based on either GB- or ND-based method is applied to all the nodes having functional assertions and internal structures. In Line 4, for nodes with internal structures (i.e., $G \neq nil$), we recursively apply "Verify" to G and verify the node using either GB- or ND-based method. On the other hand, for nodes without any internal structure (i.e., $G = nil$), "Verify" always returns true since the node represents a logic element (i.e., a pre-defined gate in a standard cell library). Line 6 corresponds to Step 2. "Layer" returns the layer level of the function, as shown in Fig. 5. "Layer" can be implemented easily by looking at the form of the function and whether G consists of decomposition/composition nodes. Consequently, each node is verified properly in Lines 7 or 9. If there are nodes whose functions are fully given as mathematical substitutions with higher degrees, we can reduce the total verification time.

Algorithm 3. GB-ND Hybrid Verification Method

Input: GF-ACG $(\boldsymbol{N}, \boldsymbol{E})$
Output: Verification result $res \in \{\text{true}, \text{false}\}$
1: **Function** Verify($(\boldsymbol{N}, \boldsymbol{E})$)
2: **Bool** $res \leftarrow$ true;
3: **for** each $(\boldsymbol{F}', G) \in \boldsymbol{N}$ **do**
4: **if** $G \neq nil$ **then**
5: $res \leftarrow res$ & Verify(G);
6: **if** Layer($(\boldsymbol{F}, G)$) ≤ 2 **then**
7: $res \leftarrow res$ & ND-BasedMethod$(\boldsymbol{F}, G)$;
8: **else then**
9: $res \leftarrow res$ & GB-BasedMethod$(\boldsymbol{F}, G)$;
10: **end if**
11: **end if**
12: **end for**
13: **return** res;
14: **end**

3.3 Experimental Evaluation

To evaluate the efficiency of the ND-based method, we show the design and verification of the $GF(2^8)$ circuits consisting of an n-input adder and an inversion which are sometimes used in ECC decoders and cryptographic processors.

Fig. 6 shows a GF-ACG for the target circuit and Table 2 shows the nodes, GFs, and GF variables in Fig. 6. The node

TABLE 2
Nodes, GF, and GF Variables in Fig. 6

Nodes
[Accumulator and inversion]
$n_0 = (\{b = \left(\sum_{i=0}^{n-1} a_i\right)^{254}\}, G_1)$
[Accumulator] $n_1 = (\{w = \sum_{i=0}^{n-1} a_i\}, G_2)$
[Inversion] $n_2 = (\{b = w^{254}\}, G_3)$
GF
$GF(2^8) = ((\beta^7, \beta^6, \ldots, \beta^0), (\{0,1\}, \{0,1\}, \ldots, \{0,1\}), \beta^8 + \beta^4 + \beta^3 + \beta^2 + 1)$
GF variables
$a_i = (GF(2^8), (7,0)), 0 \leq i \leq n-1$
$b, w = (GF(2^8), (7,0))$

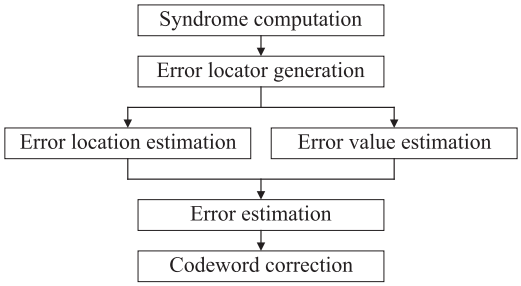


Fig. 7. Typical flow of RS decoder.

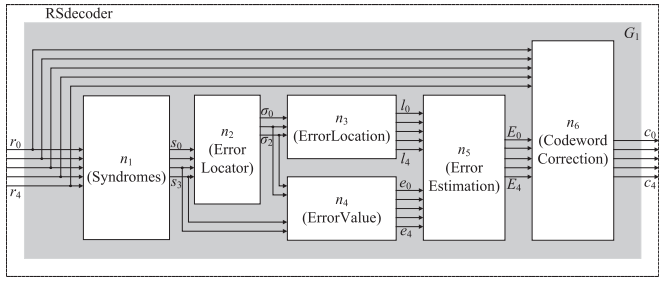


Fig. 8. GF-ACG for RS decoder.

"Accumulator and inversion" in Fig. 6a is the top-level node. Its functional assertion is given by

$$b = \left(\sum_{i=0}^{n-1} a_i \right)^{254}. \tag{7}$$

This function is included in the second layer in Fig. 5 because it performs an exponentiation of a multivariate polynomial. Fig. 6b shows the internal structure of the "Accumulator and inversion" node, which consists of two nodes: "n-adder" and "$GF(2^8)$ inversion." The n-adder node is described by two-input adders (i.e., bit-parallel XOR) and $GF(2^8)$ inversion is described by an addition-chain exponentiation to compute $b = w^{254}$. (See [8] for details.)

The verification time of such circuits for $2 \leq n \leq 8$ was measured by open-source computer algebra system Risa/Asir on a Linux PC with Intel Xeon E5450 3.00 GHz processor and 32 GB RAM. Table 3 shows the results using the conventional RTL simulation, GB-based method, and ND-based method (the proposed method). Note that Table 3 shows the verification time only for the top-level node. The RTL simulation could not verify the circuits completely if the number of operands was more than five. This is because the verification requires 2^{40} test patterns when $n = 5$. The GB-based method also failed to verify the circuits when $n \geq 5$ because of the memory limitations caused by the large number of terms. More precisely, the number of terms becomes ${}_{254+n-1}C_{n-1}$ ($\approx 2^{62}$ in the case of $n = 5$). On the other hand, the ND-based method successfully verified such circuits in a fixed time independently of n because the verification time only depends on the number of equations representing the internal structure. Thus, we confirmed that the proposed method is efficient in verifying GF arithmetic circuits of high degrees.

4 Application to ECC Decoders

4.1 Reed-Solomon Decoder

This section shows a formal design and verification of Reed-Solomon code decoder [16] using GF-ACGs as an application of the proposed method described in Section 3. RS code is a kind of ECC suitable for correcting consecutive bit errors. It treats a unit of m bits, which is called a symbol, as an element of $GF(2^m)$, where m indicates the symbol length. In many cases, m is 4 (a nibble), 8 (a byte), or 16 (a word). The number of symbols constructing one code word is called the code length. When an RS code has a code length of n and can correct less than t (symbols) errors, a single code word sends $n - 2t$ $(= k)$ information symbols at a time, where t and k indicate the error correction capacity and the information symbol length, respectively.

TABLE 3
Verification Time of n_0 (s)

# of variables n	2	3	4	5	6	7	8
RTL simulation	0.56	30.16	10,032.98	N/A	N/A	N/A	N/A
GB-based	0.33	0.49	130.84	N/A	N/A	N/A	N/A
GB-ND hybrid	0.37	0.32	0.28	0.27	0.35	0.33	0.32

An ECC decode derives a correct codeword $\boldsymbol{C} = (c_{n-1}, c_{n-2}, \ldots, c_0)$ from a received word $\boldsymbol{r} = (r_{n-1}, r_{n-2}, \ldots, r_0)$ that may include errors $\boldsymbol{E} = (E_{n-1}, E_{n-2}, \ldots, E_0)$, where c_i, r_i, and E_i are symbols.

Fig. 7 shows the typical architecture of the RS decoder designed in this paper. The decoder consists of six sub-functions: (i) syndrome computation, (ii) error locator polynomial generation, (iii) error location estimation, (iv) error value estimation, (v) error estimation, and (vi) corrected codeword computation. There are several algorithms for implementing each stage. Among them, we select the algorithms suitable for hardware implementation [17]. For the GF-ACG description, we derive an algebraic description of each stage. In particular, (iii) error location estimation and (iv) error values estimation are represented as high-degree functions because we solve a high-degree equation $X^{2^m-1} - 1$ in (iii) and calculate inversion (i.e., to the power of $2^m - 2$) in (iv), where X indicates an indeterminate element. See our previous work [13] for more details.

4.2 Formal Design and Verification Based on GF-ACGs

Fig. 8 shows the GF-ACGs of an RS decoder for $m = 8$, $n = 5$, and $t = 2$ at the top two levels of abstraction, and Table 4 shows the nodes, GFs, and variables of the internal structure of the "RSdecoder" node. (Note here that the figures of GF-ACGs for the lower levels of abstraction are omitted for the simplicity.) The node "RSdecoder" in Fig. 8 is the top-level node representing the entire decoder function with n inputs and n outputs. Each input or output (r_i or c_i) represents an element of $GF(2^8)$ which is an 8-bit signal. Its functional assertion is given in the form of mathematical substitutions derived from sub-functions in Fig. 7 because the outputs of former functions are fed into the inputs of the latter functions. As a result, the degrees of the functional assertions become quite high. The "Syndromes," "ErrorLocator," "ErrorLocation," "ErrorValue," "ErrorEstimation," and "CodewordOutput" nodes perform the sub-functions corresponding to (i)–(vi). As mentioned before, "ErrorLocation" and "ErrorValue" have high-degree functions.

TABLE 4
Nodes, GF, and Variables in Second Level Description of RS Decoder

Nodes
[Syndromes]
$n_1 = (\{s_j = \sum_{i=0}^{n-1} \beta^{ij} \times r_i \mid 0 \le j \le 2t-1\}, G_2)$
[ErrorLocator]
$n_2 = (\{\sigma_0 = s_2^2 + s_1 \times s_3,$ $\sigma_1 = s_1 \times s_2 + s_0 \times s_3,$ $\sigma_2 = s_1^2 + s_0 \times s_2\}, G_3)$
[ErrorLocation]
$n_3 = (\{L_i = \left(\sum_{u=0}^{t} \beta^{iu} \times \sigma_u\right)^{2^m-1} + 1 \mid 0 \le i \le n-1\}, G_4)$
[ErrorValue]
$n_4 = (\{E_i = \beta^{-2it} \times \left(\sum_{u=0}^{t'} (\beta^i)^{2u+1} \times \sigma_{2u+1}\right)^{2^m-2} \times \left(\sum_{u=0}^{t-1} \sum_{v=0}^{u} \beta^{iu} \times s_{2t-v-1} \times \sigma_{t-u+v-1}\right) \mid 0 \le i \le n-1\}, G_5)$
[ErrorEstimation]
$n_5 = (\{e_i = L_i \times E_i \mid 0 \le i \le n-1\}, G_6)$
[CodewordCorrection]
$n_6 = (\{c_i = r_i + e_i \mid 0 \le i \le n-1\}, G_7)$
GF
$GF(2^8) = ((\beta^7, \beta^6, \ldots, \beta^0), (\{0,1\}, \ldots, \{0,1\}), \beta^8 + \beta^4 + \beta^3 + \beta^2 + \beta^0)$
GF variables
$r_i = (GF(2^8), (7,0)),\ (0 \le i \le n-1)$
$c_i = (GF(2^8), (7,0)),\ (0 \le i \le n-1)$
$s_j = (GF(2^8), (7,0)),\ (0 \le j \le 2t-1)$
$\sigma_u = (GF(2^8), (7,0)),\ (0 \le u \le t)$
$L_i = (GF(2^8), (7,0)),\ (0 \le i \le n-1)$
$E_i = (GF(2^8), (7,0)),\ (0 \le i \le n-1)$
$e_i = (GF(2^8), (7,0)),\ (0 \le i \le n-1)$

Table 5 shows the verification times of the RS decoder for $m = 4, n = 5$, and $t = 2$, by the GB-based method [8] and GB-ND hybrid method newly presented in this paper. The experimental setup is the same as that in the Section 3. The proposed method applied the ND-based method to "RSdecoder," "ErrorLocation," and "ErrorValue" since "RSdecoder" belongs to ECC/cryptographic function layer and "ErrorLocation" and "ErrorValue" belong to the $GF(p^m)$ exponentiation of a multivariate polynomial layer, as illustrated in Fig. 5. As shown in Table 5, the proposed method makes it possible to verify the RS decoder in 14 s while the GB-based method required about 4 h to verify it. Table 6 shows the verification times of larger RS decoders

TABLE 5
Verification Time of RS Decoder ($m = 4, n = 5, t = 2$)

Nodes	Verification Time (s)	
	GB-based	GB-ND hybrid
RSdecoder	14,016.037	0.168
Syndromes	4.221	3.980
ErrorLocator	2.266	2.248
ErrorLocation	2.913	2.618
ErrorValue	5.110	4.249
ErrorEstimation	0.205	0.204
CodewordCorrection	0.383	0.205
Total	14,041.940	13.673

TABLE 6
Verification Time of RS Decoders ($m = 8, t = 2$)

Code length n	16	32	64	128	255
Verification time (s)	31.1	55.3	97.5	167.6	293.9

with $m = 8$ and $t = 2$ for different n. Note here that the GB-based method has difficulty verifying such practical decoders for any code length because it takes an enormous amount of time to verify the top-level nodes that have a high degree function. Logic simulation also fails to verify these circuits completely because these are performed with operands longer than 128-bits. In contrast, the proposed method can verify such sophisticated circuits even if n becomes the largest value ($n = 255$).

5 Application to Cryptographic Processors

5.1 AES

The AES is a symmetric key cryptography stipulated in ISO/IEC 18033-3. AES is a 128-bit block cipher. The initial key length is 128, 196, or 256 bits and round keys are generated by a key scheduling function. In the AES algorithm, 128-bit data is considered as 4×4 matrix whose elements are given by bytes (i.e., a variable of $GF(2^8)$). AES encryption with a 128-bit key repeats the data randomization function 10 times, which is called a round.

Fig. 9 shows the encryption and decryption flow of AES with a 128-bit key. AES encryption mainly consists of four subfunctions: SubBytes, ShiftRows, MixColumns, and AddRoundKey. In addition, KeyScheduling is a function for round key generation. These operations are performed over $GF(2^8)$ with an irreducible polynomial $\beta^8 + \beta^4 + \beta^3 + \beta + 1$. Algebraic descriptions for GF-ACG can be found in [7]. SubBytes is given by a high-degree function. In addition, MixColumns and AddRoundKey consist of multi-input addition. Such functions are essential to modern ciphers to ensure resistance against

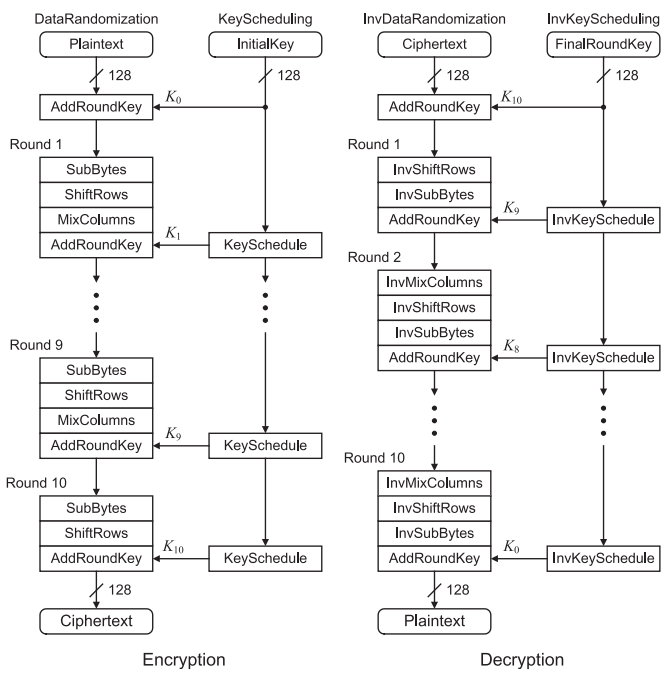


Fig. 9. AES encryption and decryption flow.

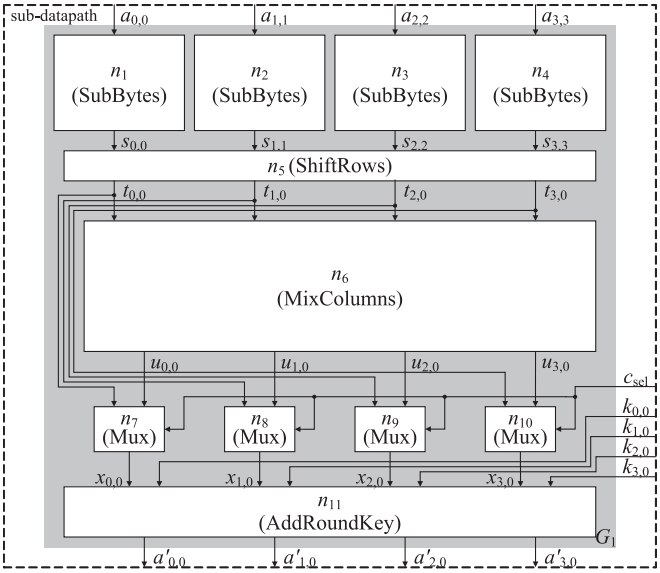


Fig. 10. GF-ACG for sub-datapath of AES encryption processor.

cryptoanalysis. Note that AES decryption is performed by the inverse encryption functions in reverse order.

5.2 Formal Design and Verification Based on GF-ACG

This paper focus on the design and verification of AES encryption and decryption processors which perform one round per clock cycle. Note that we do not consider the key scheduling function to generate round keys. However, AES processors with round key generation on-the-fly can also be designed in the same manner.

The AES processor datapath is divided into four sub-datapaths, and each of them encrypts (or decrypts) four bytes independently. Figs. 10 and 11 show the GF-ACGs of the sub-datapaths for the AES encryption and decryption processors at the second and third levels of abstraction, respectively. The top-level nodes (i.e., the entire processor datapaths) are composed of sub-datapaths connected in parallel. (The figures and descriptions of the top-level nodes

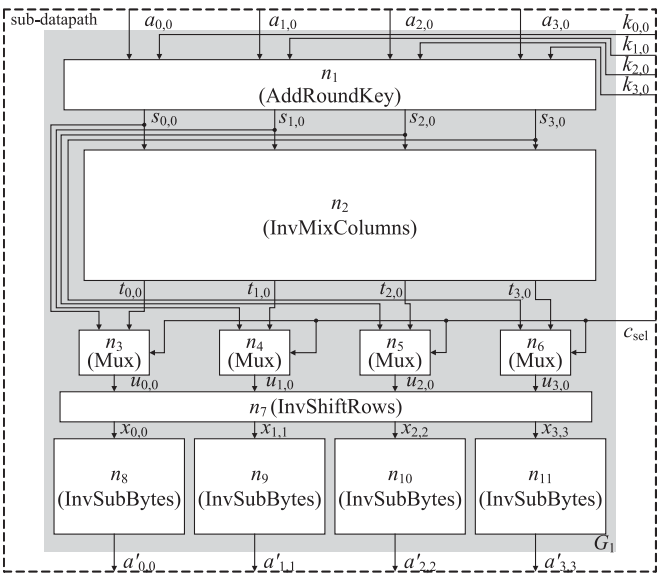


Fig. 11. GF-ACG for sub-datapath of AES decryption processor.

TABLE 7
Nodes, GFs, and Variables in AES Encryption Processor (Fig. 10)

Nodes
[sub-datapath]
$n_0 = (\{a'_{i,0} = k_{i,0} + c_{sel}(\sum_{d_r=0}^{7} c_{d_r} a_{i,i}^{-2^{d_r}} + c_8) + (1 - c_{sel}) \sum_{e_r=0}^{3} v_{e_r-i}(\sum_{d_r=0}^{7} c_{d_r} a_{e_r,e_r}^{-2^{d_r}} + c_8) \mid 0 \le i \le 3\}, G_1)$
[SubBytes]
$n_{i+1} = (\{s_{i,i} = \sum_{d_r=0}^{7} c_{d_r} a_{i,i}^{-2^{d_r}} + c_8\}, G_{i+2})$
[ShiftRows]
$n_5 = (\{t_{i,0} = s_{i,i} \mid 0 \le i \le 3\}, G_6)$
[MixColumns]
$n_6 = (\{u_{i,0} = \sum_{e_r=0}^{3} v_{e_r-i} t_{e_r,0} \mid 0 \le i \le 3\}, G_7)$
[Mux]
$n_{i+7} = (\{x_{i,0} = c_{sel} t_{i,0} + (1 - c_{sel}) u_{i,0}\}, G_{i+8})$
[AddRoundKey]
$n_{11} = (\{a'_{i,0} = k_{i,0} + x_{i,0} \mid 0 \le i \le 3\}, G_{12})$
GFs
$GF(2^8) = ((\beta^7, \beta^6, \ldots, \beta^0), (\{0,1\}, \{0,1\}, \ldots, \{0,1\}), \beta^8 + \beta^4 + \beta^3 + \beta^1 + \beta^0)$
$GF(2) = \big((\beta^0), (\{0,1\}), nil\big)$
GF variables
$a_{i,i}, a'_{i,0}, s_{i,i}, t_{i,0}, u_{i,0}, k_{i,0}, x_{i,0} = (GF(2^8), (7,0)),\ (0 \le i \le 3)$
$c_{sel} = (GF(2^8), (0,0))$

are omitted for simplicity.) Tables 7 and 8 show the nodes of the AES encryption and decryption processor sub-datapaths, respectively.

Here, we explain the details of the encryption processor. Node "sub-datapath" is the second-level node, whose

TABLE 8
Nodes, GFs, and Variables in AES Decryption Processor (Fig. 11)

Nodes
[sub-datapath]
$n_0 = (\{a'_{i,0} = (\sum_{d_r=0}^{7} c_{d_r}(c_{sel}(k_{i,0} + a_{i,0}) + (1 - c_{sel}) \times \sum_{e_r=0}^{3}(v'_{e_r-i}(k_{e_r,-i} + a_{e_r,-i})))^{2^{d_r}} + c_8)^{254} \mid 0 \le i \le 3\}, G_1)$
[AddRoundKey]
$n_{11} = (\{s_{i,-i} = k_{i,-i} + a_{i,-i} \mid 0 \le i \le 3\}, G_{12})$
[InvMixColumns] $(0 \le i \le 3)$
$n_6 = (\{t_{i,-i} = \sum_{e_r=0}^{3} v'_{e_r-i} s_{e_r,0}, G_7)$
[InvShiftRows]
$n_5 = (\{x_{i,0} = u_{i,-i} \mid 0 \le i \le 3\}, G_6)$
[InvSubBytes] $(0 \le i \le 3)$
$n_{i+1} = (\{a'_{i,0} = (\sum_{d_r=0}^{7} c_{d_r} x_{i,0}^{2^{d_r}} + c_8)^{254}\}, G_{i+2})$
[Mux] $(0 \le i \le 3)$
$n_{i+7} = (\{x_{i,0} = c_{sel} t_{i,0} + (1 - c_{sel}) u_{i,0}\}, G_{i+8})$
GFs
$GF(2^8) = ((\beta^7, \beta^6, \ldots, \beta^0), (\{0,1\}, \{0,1\}, \ldots, \{0,1\}), \beta^8 + \beta^4 + \beta^3 + \beta^1 + \beta^0)$
$GF(2) = \big((\beta^0), (\{0,1\}), nil\big)$
GF variables
$a_{i,-i}, a'_{i,0}, s_{i,-i}, t_{i,0}, u_{i,0}, k_{i,0}, x_{i,0} = (GF(2^8), (7,0)),\ (0 \le i \le 3)$
$c_{sel} = (GF(2^8), (0,0))$

TABLE 9
Verification Time of AES Encryption Processor

Nodes	Verification Time (s)	
	GB-based	GB-ND hybrid
AESencryption	0.44	0.07
sub-datapath	0.25	0.07
SubBytes	3.72	3.72
ShiftRows	0	0
MixColumns	0.84	0.69
Mux	0.63	0.35
AddRoundKey	0.24	0.21
Total	6.12	5.07

inputs are 4-byte data, a 4-byte round key, and 1-bit control signal c_{sel}. The functional assertion, which is derived deductively from its four sub-functions and Mux, is very high degree. Nodes "SubBytes," "ShiftRows," "MixColumns," and "AddRoundKey" in the internal structure have functional assertions corresponding to the sub-functions. As mentioned above, "SubBytes" has a high-degree function with one variable and "MixColumns" has four linear functions with four variables. Therefore, their combination also has high-degree functions with a large number of terms. Lower-level abstractions are detailed in [7]. The decryption processor is also designed in the same manner as the encryption processor. "InvSubBytes," "InvShiftRows," and "InvMixColumns" perform the inverse functions of "SubBytes," "ShiftRows," and "MixColumns," respectively. Note that "AddRoundKey" has the same function as in the encryption since it is a simple key addition (i.e., bit-parallel XOR with a round key).

For comparison, the AES encryption and decryption processors were verified by the GB-based method and the proposed GB-ND hybrid method. Tables 9 and 10 show the verification times of the encryption and decryption processors, respectively, where "AESencryption" and "AESdecryption" indicate the top-level nodes of the encryption and decryption processors, respectively. The experimental setup was the same as Section 3. In this verification, the ND-based method was applied to "AESencryption" and "sub-datapath" since "AESencryption" belongs to the ECC/cryptographic layer and "sub-datapath" belongs to the $GF(p^m)$ exponentiation of multivariate polynomial layer in Fig. 5. Note that "ShiftRows" and "InvShiftRows" were not verified since they were implemented only by wiring. As a result, the GB-based method failed to verify

TABLE 10
Verification Time of AES Decryption Processor

Nodes	Verification Time (s)	
	GB-based	GB-ND hybrid
AESdecryption	N/A	0.07
sub-datapath	N/A	0.07
AddRoundKey	0.24	0.21
InvMixColumns	1.71	1.71
Mux	0.63	0.35
InvShiftRows	0	0
InvSubBytes	4.58	4.58
Total	N/A	6.99

TABLE 11
Classification of Modern Ciphers

	Applicable		Not applicable
GF	$GF(2^8)$	$GF(2^4)$	$GF(2)$
Ciphers	AES, Camellia, MUGI, SNOW, KCipher-2	Piccolo, LED, TWINE	PRESENT, PRINCE, SIMON

"AESdecryption," but could verify "AESencryption." This was caused by the order of SubBytes and MixColumns (InvSubBytes and InvMixColumns). The function of "AESencryption" does not have fewer terms than "AESdecryption" since the encryption processor performs SubBytes before MixColumns. Therefore, when verifying "AESdecryption," the GB-based method required an enormous amount of memory and computational time because of the huge number of terms in the polynomials. On the other hand, the proposed method successfully verified "AESdecryption" in less than a second since the ND-based method could verify the function without expanding the polynomials. This result confirms the efficiency of the proposed method in verifying cryptographic processors.

6 Discussion

In this section, we discuss the applicability of the proposed method.

The AES encryption processor designed in this paper performs the subfunctions in the order of SubBytes, ShiftRows, MixColumns, and AddRoundKey, which is a naive implementation. Such naive hardware can be verified even by the conventional GB-based method as shown in the previous paper [7]. On the other hand, in many previous works on AES hardware implementation, resister-retiming and subfunction-reordering techniques heve been proposed to reduce the circuit delay and area [18], [19], [20], [21]. We should consider these optimized techniques for designing practical AES processors. The conventional GB-based method would require huger time to verify such practical AES encryption datapaths that perform AddRouneKey or MixColumns prior to SubBytes because the function contains the 254th power of multivariate polynomials as a result of the reordering. On the other hand, the GB-ND hybrid method can efficiently verify such practical datapaths where the function can be derived deductively even after the resister-retiming and subfunction-reordering. Thus, the applications of GF-ACG-based design can be extended by the proposed method. Note here that it is still difficult to verify highly bit-level optimized circuits (e.g., masked inversion circuits [22], [23] known as a countermeasure against side-channel attacks [24]) even by the proposed method. This is because the masked inversion circuit does not have any hierarchical structure as in Fig. 5.

A further application of the proposed method to other symmetric key algorithms would be possible because many modern ciphers are represented by $GF(2^m)$ $(m \geq 4)$ arithmetic. Table 11 shows a classification of some modern ciphers by GF used. Some of the ISO/IEC 18033 standard ciphers with $GF(2^8)$ arithmetic, such as Camellia [25], SNOW [26], MUGI [27], and KCipher-2 [28], can be

designed and verified using the proposed method. Some of lightweight cipher with $GF(2^4)$ arithmetic, such as LED [29], Piccolo [30], and TWINE [31], are also possible applications. On the other hand, it would be difficult to apply for ciphers which are not based on $GF(2^m)$ arithmetic and/or cannot be described in a hierarchical manner such as PRESENT [32], PRINCE [33], and SIMON [34].

7 Conclusion

This paper presented an efficient approach to verifying GF arithmetic circuits of higher degrees based on GF-ACGs. The proposed method combines the conventional algebraic method with a newly-proposed verification method based on natural deduction for predicate logic. The new method enables us to verify higher-degree arithmetic circuits (e.g., exponentiation of polynomials) frequently used in ECC or cryptographic processors. In addition, this paper proposed an overall verification flow using the two different methods properly. The efficiency of the proposed method was demonstrated through experimental designs and verifications of general arithmetic circuits, RS decoders, and AES encryption/decryption processors. The results confirmed that the proposed method could verify the circuits with higher degrees in a practical time while the conventional method either took an enormous amount of time or failed.

A further extension of the natural deduction method for other types of applications would remain in the future study. For example, a hierarchical description of GF-ACG might sometimes be unavailable for some circuits highly optimized in the logic level even if the function includes the exponentiation of a multivariate polynomial. In such cases, we would have a difficulty in verifying the circuits by both the GB-based and ND-based methods. Another further extension of GF-ACGs would be necessary for such circuits.

Acknowledgments

We are particularly grateful for the assistance given by Kotaro Okamoto of HITACHI Co. Ltd., Japan. This work has been supported by JSPS KAKENHI Grant No. 25240006.

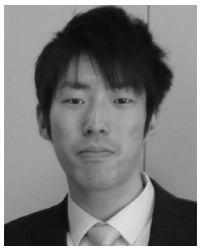
References

[1] E. Savas and K. C. Koc, "Finite field arithmetic for cryptography," *IEEE Circuits Syst. Mag.*, vol. 10, no. 2, pp. 40–56, Aug. 2010.
[2] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
[3] E. R. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Proc. 32nd Des. Autom. Conf.*, 1995, pp. 535–541.
[4] R. Drechsler, Ed., *Advanced Formal Verification*. Norwell, MA, USA: Kluwer, 2004.
[5] R. Stankovic and R. Drechsler, "Circuit design from Kronecker Galois field decision diagrams for multiple-valued functions," in *Proc. 27nd IEEE Int. Symp. Multiple-Valued Logic*, 1997, pp. 275–280.
[6] A. M. Jabir and D. K. Pradhan, "A graph-based unified technique for computing and representing coefficients over finite fields," *IEEE Trans. Comput.*, vol. 56, no. 8, pp. 1119–1132, Aug. 2007.
[7] 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 Secur.*, vol. 7, no. 1, pp. 3–13, Feb. 2012.
[8] 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.
[9] D. A. Cox, J. B. Little, and D. O'Shea, *Using Algebraic Geometry*. New York, NY, USA: Springer-Verlag, 1998.
[10] J. Lv, P. Kalla, and F. Enescu, "Verification of composite Galois field multipliers over $GF((2^m)^n)$ using computer algebra techniques," in *Proc. IEEE Int. High Level Des. Validation Test Workshop*, 2011, pp. 136–143.
[11] J. Lv and P. Kalla, "Formal verification of Galois field multipliers using computer algebra techniques," in *Proc. 25th Int. Conf. VLSI Des.*, 2012, pp. 388–393.
[12] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner basis reductions for formal verification of Galois field multipliers," in *Proc. Conf. Des. Autom. Test Europe*, 2012, pp. 899–904.
[13] R. Ueno, K. Okamoto, N. Homma, and T. Aoki, "An efficient approach to verifying Galois-field arithmetic circuits of higher degrees and its application to ECC decoders," in *Proc. IEEE 44th Int. Symp. Multiple-Valued Logic*, 2014, pp. 144–149.
[14] B. Buchberger, "Some properties of Gröbner-bases for polynomial ideals," *ACM SIGSAM Bulletin*, vol. 10, no. 4, pp. 19–24, 1976.
[15] D. Prowitz, *Natural Deduction A Proof-Theoretical Study*. Stockholm, Sweden: Almqvist & Wiksell, 1965.
[16] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM J. Appl. Mathematics*, vol. 8, pp. 300–304, 1960.
[17] W. W. Peterson, "Encoding and error correction procedures for the Bose-Chaudhuri codes," *IRE Trans. Inf. Theory*, vol. IT-6, no. 4, pp. 459–470, Sep. 1960.
[18] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Advances in Cryptology*. Berlin, Germany: Springer, 2001, pp. 239–254.
[19] I. Hammad, K. El-Sankary, and E. El-Masry, "High-speed AES encryptor with efficient merging techniques," *IEEE Embedded Syst. Lett.*, vol. 2, no. 3, pp. 67–71, Sep. 2010.
[20] S. K. Mathew, et al., "53 Gbps native $GF(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 767–776, Apr. 2011.
[21] R. Ueno, S. Morioka, N. Homma, and T. Aoki, "A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths—toward efficient CBC-mode implementation," Cryptology ePrint Archive, Tech. Rep. 2016/595, 2016. [Online]. Available: http://eprint.iacr.org/
[22] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the AES S-box," in *Fast Software Encryption*. Berlin, Germany: Springer, 2005, pp. 413–423.
[23] D. Canright and L. Batina, "A very compact "perfectly masked" S-box for AES," in *Applied Cryptography and Network Security*. Berlin, Germany: Springer, 2008, pp. 446–459.
[24] P. Kocher, "Differential power analysis," in *Advances in Cryptology*. Berlin, Germany: Springer, 1999, pp. 388–397.
[25] K. Aoki, et al., "Camallia: A 128-bit block cipher suitable for multiple platforms—design and analysis," in *Selected Areas in Cryptography*. Berlin, Germany: Springer, 2001, pp. 39–56.
[26] P. Ekdahl and T. Johansson, "A new version of the stream cipher SNOW," in *Selected Areas in Cryptography*. Berlin, Germany: Springer, 2001, pp. 47–61.
[27] Hitachi Ltd., "MUGI pseudorandom number generator specification ver 1.2," 2001. [Online]. Available: http://www.hitachi.com/rd/yrl/crypto/mugi/mugi_spe.pdf
[28] K. Shinsaku, T. Toshiaki, and S. Kouichi, "K2: A stream cipher algorithm using dynamic feedback control," in *Proc. Int. Conf. Secur. Cryptography*, 2007, vol. 1, pp. 204–213.
[29] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The LED block cipher," in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, 2011, pp. 326–341.
[30] K. Shibutani, T. Isobe, H. Hiwatiri, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, 2011, pp. 342–357.
[31] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE: A lightweight block cipher for multiple platforms," in *Selected Areas in Cryptography*. Berlin, Germany: Springer, 2012, pp. 339–354.
[32] A. Bogdanov, et al., "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, 2007, pp. 450–466.

[33] J. Borghof, et al., "PRINCE—a low-latency block cipher for pervasive computing applications," in *Advances in Cryptography*. Berlin, Germany: Springer, 2012, pp. 208–225.
[34] E. Pavlenko, et al., "STABLE: A new QF-BV SMT solver for hard verification problems combining Boolean reasoning with computer algebra," in *Proc. Conf. Des. Autom. Test Europe*, 2011, pp. 1–6.

Rei Ueno received the BE degree in information engineering, and the MS degree in information sciences from Tohoku University, Sendai, Japan, in 2013 and 2015, respectively. He is currently working toward the doctoral degree at Tohoku University. Since 2016, he has been a research fellow in Japan Society for the Promotion of Science. His research interests include arithmetic circuits, cryptographic implementations, formal verification, and hardware security. He is a student member of the IEEE.

Naofumi Homma received the BE degree in information engineering, and the MS and PhD degrees in information sciences from Tohoku University, Sendai, Japan, in 1997, 1999, and 2001, respectively. He is currently a professor in the Research Institute of Electrical Communication, Tohoku University. Form 2002 to 2006, he also joined Japan Science and Technology Agency as a researcher for the PRESTO project. His research interests include computer arithmetic, EDA methodology, high performance/secure VLSI computing, and hardware security. He received the IP Award at the 2005 LSI IP Design Award, the Best Paper Award at the Workshop on Synthesis And System Integration of Mixed Information Technologies in 2007, the Best Symposium Paper Award at the 2013 IEEE International Symposium on Electromagnetic Compatibility, and the Best Paper Award at the Workshop on Cryptographic Hardware and Embedded Systems 2014. He is a member of the IEEE.



Yukihiro Sugawara received the BE degree in information engineering, and the MS degree in information sciences from Tohoku University, Sendai, Japan, in 2014 and 2016, respectively. His research interests include arithmetic circuits and formal verification.

Takafumi Aoki received the BE, ME, and DE degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1988, 1990, and 1992, respectively. He is currently a professor in the Graduate School of Information Sciences, Tohoku University. Form 1997 to 1999, he also joined Japan Science and Technology Corp. for the PRESTO project. His research interests include theoretical aspects of computation, VLSI computing structures for signal and image processing, multiple-valued logic, and biomolecular computing. He received the Outstanding Paper Award at the 1990, 2000, 2001, and 2006 IEEE International Symposia on Multiple-Valued Logic, the Outstanding Transactions Paper Award form the Institute of Electronics, Information and Communication Engineers (IEICE), Japan, in 1989 and 1997, the IEE Mountbatten Premium Award in 1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems, the IP Award at the Seventh LSI IP Design Award in 2005, the Best Paper Award at the 14th Workshop on Synthesis and System Integration of Mixed Information Technologies, and the Best Paper Award at the Workshop on Cryptographic Hardware and Embedded Systems 2014. He is a member of the IEEE.