Toward Formal Design of Cryptographic Processors Based on Galois Field Arithmetic

Naofumi Homma
Tohoku University, Japan
Formal processor design?

- Formal description and verification at front-end design
- Goal: circuit description whose function is completely guaranteed

### Diagram

**Front-end design**

**Back-end design**

**Specification**
- System/Architecture
- Functional simulation

**Logic synthesis**
- Place and route
- Clock tree synthesis

...
References


Outline

- Why formal design?
- Galois-Field Arithmetic Circuit Graph: GF-ACG
- Formal verification using computer algebra
- Application to AES processor design
- Conclusions
Arithmetic circuits over Galois fields

Demands of high security and reliable systems
- Cryptography, Error correction code
  - Arithmetic operations over Galois Fields (GF)
  - ASIC implementation

GFs used in the ISO/IEC18033 standard cryptosystems

<table>
<thead>
<tr>
<th>Public key ciphers</th>
<th>Symmetric key ciphers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECIES-KEM</td>
<td>Block ciphers</td>
</tr>
<tr>
<td>PSEC-KEM</td>
<td>AES</td>
</tr>
<tr>
<td>GF(2^{160})~</td>
<td>Camellia</td>
</tr>
<tr>
<td>GF(p) (log_{2}p&gt;160)</td>
<td>MUGI</td>
</tr>
<tr>
<td>GF(2^{8})</td>
<td>GF(2^{8})</td>
</tr>
<tr>
<td>GF(2^{8})</td>
<td>GF((2^{4})^{2})</td>
</tr>
<tr>
<td>GF(2^{8})</td>
<td>GF((2^{8})^{4})</td>
</tr>
</tbody>
</table>

Difficulties in designing GF arithmetic circuits

GSIS, TOHOKU UNIVERSITY
Design issues

- Lowest level description using logical expressions
  - GF arithmetic is not supported in high-level design environments
    - Describe any basic operation by gates
    - Huge AND-XOR expressions

- Huge simulation times due to long operands
  - Impossible to simulate all input patterns in practical applications
  - Monte Carlo Test assuming no corner-case

- Test bench program might include a bug
  - Not intuitive, Hard to visual confirmation
Necessity of complete verification

Any corner case is not allowed in critical systems
- Software in critical system is usually verified formally

Complete verification is in high demand for cryptographic hardware
- Variety of circuit architectures
  - Optimizations depending on specifications
  - Countermeasures against physical attacks
    e.g. Architectures for SubBytes and Mixcolumns in AES:
    Table S-box (Enc), Composite-field S-box (Enc), Composite-field inversion (Enc), Composite S-box (Enc+Dec), Mix+InvMix, Composite-field InvMix+InvAffine, S-box with random masking, Mix with random masking…

- Bugs (or initial failures) in crypto processors could severely compromise the security [CRYPTO ‘08]
Formal verification of arithmetic circuits

- **Formal verification**: Mathematically check the equivalence between specification and circuit description.

- Conventional methods for integer arithmetic circuits:
  - Decision diagrams
  - Moment diagrams
  - Word-level DDs, *BMDs
  - Not suitable or not applicable for describing GF arithmetic circuits.

![Binary Decision Diagram (BDD)](image1)

![*Binary Moment Diagram (*BMD)](image2)
Example of BDD for $GF(2^4)$ multiplier

$Z[3:0] = X[3:0] \times Y[3:0]$
Conventional works for GF arithmetic circuits

- **Formal verification of** $GF(2^m)$ circuits [Morioka ‘01]
  - Positive Polarity Reed-Muller (PPRM) representation for equivalence checking
  - Successful verification of decoder in ECC circuit

- **Extension to** $GF((2^m)^n)$ arithmetic [Mukhopadhyay ‘07]
  - Hierarchical design approach

- **Formal verification of AES** software [Slobodova ‘08]
  - Verification on extended CPU instruction set
This work

- Formal design of $GF(p^m)$ arithmetic circuits and its application to cryptographic hardware design
  - Arithmetization + Hierarchical design approach
  - GF-ACG: Galois-Field Arithmetic Circuit Graph
  - Formal verification using computer algebra

![Graph showing verification time for $GF(2^m)$ multipliers]

Verification time for $GF(2^m)$ multipliers

Logic Simulation

Proposed method
Outline

- Why formal design?
- Galois-Field Arithmetic Circuit Graph: GF-ACG
- Formal verification using computer algebra
- Application to AES processor design
- Conclusions
Extension fields

- Galois field of order $p^m$: $GF(p^m)$  $p$: prime number
- Each element is a polynomial over $GF(p)$
- Addition and multiplication are performed modulo irreducible polynomial $IP$ of degree $m$
  - e.g., $GF(2^2) = \{ 0, 1, \beta, \beta + 1 \}$  $IP = \beta^2 + \beta + 1$

### Addition over $GF(2^2)$

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>$\beta$</th>
<th>$\beta + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$\beta$</td>
<td>$\beta + 1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\beta + 1$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>$\beta + 1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\beta + 1$</td>
<td>$\beta + 1$</td>
<td>$\beta$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Multiplication over $GF(2^2)$

<table>
<thead>
<tr>
<th>$\times$</th>
<th>0</th>
<th>1</th>
<th>$\beta$</th>
<th>$\beta + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$\beta$</td>
<td>$\beta + 1$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0</td>
<td>$\beta$</td>
<td>$\beta + 1$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta + 1$</td>
<td>0</td>
<td>$\beta + 1$</td>
<td>1</td>
<td>$\beta$</td>
</tr>
</tbody>
</table>
Basic ideas to represent GF arithmetic circuits

- **Arithmetization** of all internal sub-functions in a GF circuit by variables and equations over GFs

  $$ z = x \times y $$

  \[ x, y, z \in (GF(2^4), (1,0)) \]

  - Any function including logic functions can be represented by arithmetic equations over GFs

- **Hierarchical design approach**

  - Arithmetic circuits usually consist of sub-circuits that themselves compute arithmetic functions
How can we represent GFs formally?

- **Integer** (e.g. binary number system)
  
  e.g., \{0, 1, 2, 3, \ldots\} = \{(00)_2, (01)_2, (10)_2, (11)_2, \ldots\}
  
  - Represented with **Weight and Digit-set vector**
  
    \( (\ldots, 2^2, 2^1, 2^0) \ (\ldots, \{0, 1\}, \{0, 1\}, \{0, 1\}) \)

- **Galois field** (e.g. \( GF(2^2) \))
  
  e.g., \{0, 1, \beta, \beta + 1\} = \{(00)_{GF}, (01)_{GF}, (10)_{GF}, (11)_{GF}\}
  
  - Represented with **Basis and Coefficient-set vector**
  
    \( (\beta^{m-1}, \ldots, \beta^1, \beta^0) \ (\{0, 1\}, \ldots, \{0, 1\}, \{0, 1\}) \)

- **Irreducible polynomial** is required to define operations
Variables associated with Galois fields

- **Galois field**: $GF=(B, C, IP)$
  - $B$: basis, $C$: coefficient-set vector, $IP$: irreducible polynomial

- GF variables defined by $GF$ and degree range $(h, l)$

**e.g., GF variable $x$ over $GF(2^4)$**

$$
GF(2^4) = \left( (\beta^3, \beta^2, [\beta^1, \beta^0]), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), \beta^4 + \beta^1 + \beta^0 \right)
$$

degree range

$(h, l) = (1, 0)$

$$
x \in \{0,1, \beta, \beta + 1\} = (GF(2^4), (1,0))
$$
GF-ACG: Galois Field Arithmetic Circuit Graph

GF-ACG: $G = (N, E)$

- **$N$: set of nodes**
  - Node: $n = (F, G')$
    - $F$: function (GF equation)
    - $G'$: internal structure
      (GF-ACG)

- **$E$: set of directed edges**
  - Directed edge: $e = (n_s, n_d, x)$
    - $n_s$: source node
    - $n_d$: destination node
    - $x$: GF variable
Hierarchical design approach

- Find hierarchical structure in arithmetic circuits

**e.g., GF(2^2) multiplier**

![Diagram of hierarchical design approach]

- Highest level
- Lowest level

GSIS, TOHOKU UNIVERSITY
Logic gates by GF-ACG

- Pseudo logic variable
  - GF variable on a GF(2)

\[
GF(2) = ((\beta^0), \{0, 1\}, \text{nil})
\]

- Representation of logic functions

\[
\begin{align*}
\text{NOT}(u) &= 1 - u \\
\text{OR}(u, v) &= u + v - uv \\
\text{AND}(u, v) &= uv \\
\text{XOR}(u, v) &= u + v - 2uv
\end{align*}
\]

**Idempotent conditions** (property of logic signals):
\[
v^2 = v, \ u^2 = u
\]
Examples of logic gates

- **AND gate**

  \[ GF(2) = ((\beta^0), \{0,1\}, \text{nil}) \]
  \[ x, y, z \in (GF(2), (0,0)) \]

- **XOR gate**

  \[ GF(2) = ((\beta^0), \{0,1\}, \text{nil}) \]
  \[ x, y, z \in (GF(2), (0,0)) \]

Logic circuit has no internal structure because the function is guaranteed by LSI manufacturer
Encoding function for $GF(p)$ arithmetic

- Mapping from GF variables to logic variables

<table>
<thead>
<tr>
<th>Example of $x \in GF(2)$</th>
<th>Example of $x \in GF(3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GF(2)</strong></td>
<td><strong>Logic</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$x = L_0$

$L_0, L_1$: Pseudo logic variables

There are lowest-level nodes having encoding functions to transform GF and logic variables

$$x = (L_1 - 1) L_0 + 2L_1 (L_0 - 1)$$

$$L_1 L_0 = 0$$
Outline

- Why formal design?
- Galois-Field Arithmetic Circuit Graph: GF-ACG
- Formal verification using computer algebra
- Application to AES processor design
- Conclusions
**Functional verification of arithmetic circuits**

- **Formal verification**: Formally (mathematically) check the equivalence between specification and implementation.

![Diagram showing circuit specification, formula manipulation, and circuit implementation with "equal?" in the middle.](image)

- "Function" in node
- **Formula manipulation**
- "Internal structure" in node
Formal verification of GF-ACGs

- Function is correct if the same function is derived from internal structure

\[ z = x \times y \]

Solve simultaneous algebraic equations for each node

\[ \begin{align*}
  t_0 + t_1 &= x \times y \\
  z &= t_0 + t_1
\end{align*} \]
Example of verification

Hierarchical verification for each function and internal structure
Method for solving simultaneous equations

Computer algebra

Set of polynomials
\[ P = \{p_0, p_1, \ldots, p_{n-1}\} \]

Ideal membership problem
\( f \) is included in Ideal \( I \) generated by \( P \)?

Equation
\[ f(x_0, x_1, \ldots, x_{m-1}) = 0 \]

Polynomial
\[ f(x_0, x_1, \ldots, x_{m-1}) \]

Simultaneous equations
\[
\begin{align*}
  p_0(x_0, x_1, \ldots, x_{m-1}) &= 0 \\
  p_1(x_0, x_1, \ldots, x_{m-1}) &= 0 \\
  &\vdots \\
  p_{n-1}(x_0, x_1, \ldots, x_{m-1}) &= 0
\end{align*}
\]
Ideal

**Ideal:** Set of polynomials generated by a finite set of polynomials \( P = \{p_0, p_1, \ldots, p_{n-1}\} \)

\[
I = \{a_0p_0 + a_1p_1 + \cdots + a_{n-1}p_{n-1} \mid a_0, a_1, \ldots, a_{n-1} \in R[x]\}
\]

\( R[x] \): set of entire polynomials, \( P \) : basis of ideal

- Equivalent to solutions of simultaneous equations in \( P \)

**Solution for ideal membership problem using polynomial reduction**

1. Divide a polynomial \( f \) by the element of \( P \) repeatedly to get remainder \( r \)

\[
f = q_0p_0 + \cdots + q_{n-1}p_{n-1} + r \quad (q_0, \ldots, q_{n-1} : \text{quotients})
\]

2. \( f \) is an element of an ideal \( I \) if \( r = 0 \)
Polynomial reduction

- Eliminate maximal term (or head term) repeatedly according to term ordering

**Example**

\[ f = x^2 + 3xy + 2y^2 - z \]

\[ P = \{ p_0, p_1, p_2 \} \]

\[ p_0 = ab - z \]

\[ p_1 = x + y - a \]

\[ p_2 = x + 2y - b \]

**Ordering:** \( x > y > a > b > z \)

Reduction result depends on reduction procedure
Gröbner basis

- Basis with good property (Church-Rosser property)
  - Reduction result is canonical
    - $f$ is an element of an ideal $\iff$ Reduction result is 0
  - Obtained from arbitrarily basis by finite steps (Buchberger’s algorithm)

Ideal membership problem is solved by polynomial reduction on computer
The reduction result can be uniquely determined.

If the result is 0, f can be represented by a combination of elements in P.
Proposed verification method

Input: function $f$ and internal structure $P$

1. Calculate Gröbner basis $GB$ from $P$ using Buchberger’s algorithm.
2. Reduce the polynomial $f$ by $GB$.
3. Reduce result is $0$?
   - Function is correct: YES
   - Function is wrong: NO

Each node in GF-ACG is verified independently by the verification process.
Evaluation of verification time

Comparison

- Proposed method using computer algebra
  - Software: Mathematica version 6.0
- Conventional method using logic simulation
  - HDL descriptions converted from GF-ACGs
  - Simulator: Verilog-XL

Experimental Condition

- Linux PC (Intel Xeon 3.00GHz, Memory 32GB)
- Mastrovito multiplier over GF($2^m$)
- Extension degree (Operand length): 2-128
Verification time of \( GF(2^m) \) multipliers

\( GF(2^{128}) \) multiplier were verified about 10 minutes

\( GF(2^{16}) \) multiplier could not be verified
Outline

- Why formal design?
- Galois-Field Arithmetic Circuit Graph: GF-ACG
- Formal verification using computer algebra
- Application to AES processor design
- Conclusions
AES (Advanced Encryption Standard)

- Most popular block cipher
- Round function is described by GF arithmetic
  - Many modern ciphers are affected by AES
Functions in Round

- **SubBytes**
  \[ s_{i,j} = \sum_{k=0}^{7} c_k \cdot a_{i,j}^{-2^k} + c_8 \]

- **ShiftRows**
  \[ t_{i,j} = s_{j,i + j \mod 4} \]

- **MixColumns**
  \[ m_{i,j} = \sum_{k=0}^{3} v_{i+k \mod 4} \cdot t_{k,j} \]

- **AddRoundKey**
  \[ b_{i,j} = k_{i,j} + m_{i,j} \]

- **Round**
  \[ b_{i,j} = k_{i,j} + \sum_{k=0}^{3} v_{i+k \mod 4} \cdot \left[ \sum_{l=0}^{7} c_l \cdot a_{i,i + j \mod 4}^{-2^l} + c_8 \right] \]
Application to 128-bit AES processor design
GF-ACG for AES datapath

\[
n_0
\]

(AES datapath)
GF-ACG for AES datapath
GF-ACG for data randomization part

AES sub-datapath
## Evaluation of verification time

<table>
<thead>
<tr>
<th>Graph name</th>
<th>Num.</th>
<th>Verification time [sec]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Composite field</td>
<td>Extension field</td>
<td></td>
</tr>
<tr>
<td>AES datapath</td>
<td>1</td>
<td>697.15</td>
<td>711.67</td>
<td></td>
</tr>
<tr>
<td>AddInitKey</td>
<td>1</td>
<td>1.47</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>KeySchedule</td>
<td>1</td>
<td>0.58</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>Rand datapath</td>
<td>1</td>
<td>0.49</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td>Rand sub-datapath</td>
<td>4</td>
<td>0.23</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>SubBytes</td>
<td>4</td>
<td>158.01</td>
<td>3.24</td>
<td></td>
</tr>
<tr>
<td>ShiftRows</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MixColumns</td>
<td></td>
<td>0.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AddRoundKey</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>113</td>
<td><strong>858.79</strong></td>
<td><strong>718.19</strong></td>
<td></td>
</tr>
</tbody>
</table>

83 variables!

Composite field $GF(((2^2)^2)^2)$
Discussions

- Most time-consuming part is “AES datapath” at the highest level
  - 83 variables (16 1-byte inputs, 16 1-byte round keys, 16 1-byte outputs, 16 1-byte key outputs, 16 1-byte internal signals and 3 1-bit/1-byte control signals)

- Inversion over composite field \( GF(((2^2)^2)^2) \) can be verified
  - Other structures such as Table and \( GF(2^8) \) multiplier are also possible

- Complete verification of 128-bit AES datapath
  - Common loop architecture with 128-bit inputs
Future prospective

- Extension of GF-ACGs to a wider variety of GFs
  - Normal bases, Dual bases, etc.
- Hybrid verification approach with conventional DD-based approach
  - Combination of PPRM-based method and our method
- Applications to other cryptosystems
  - Public-key (e.g. ECC) and block ciphers (e.g. CLEFIA)
  - Countermeasures against physical attacks
    - AES with random masking
- Automatic generation of GF arithmetic circuits
GF($2^m$) multiplier generator on the Web

- Automatic generation of GF($2^m$) multipliers for any irreducible polynomial
- Generate only formally-proofed HDL codes

AMG: Arithmetic Module Generator

Design specification
Irreducible polynomial

Designers
Verified HDL codes

Generation & Verification based on GF-ACGs
Conclusions

- Importance of formally-proofed GF arithmetic circuits is increasing as the application to security primitives increases.

- Formal approach to designing GF arithmetic circuits based on GF-ACGs
  - Formal verification using computer algebra
  - Design and verification of a 128-bit AES datapath

- There are many works to do in the future
  - Research on formal method has a long history, but interest and demand for its application to cryptographic hardware have just increased.
Thank you for your attention