Abstract: Finite (Galois) field arithmetic finds applications in cryptography, error correction codes, signal processing, etc. Multiplication usually lies at the core of all Galois field computations and is a high-complexity operation. This paper addresses the problem of formal verification of hardware implementations of modulo-multipliers over Galois fields of the type \(\mathbb{F}_2\), using a computer-algebra/algebraic-geometry based approach. The multiplier circuit is modeled as a polynomial system in \(\mathbb{F}_2[x_1, x_2, \cdots, x_d]\) and the verification test is formulated as a Nullstellensatz proof over the finite field. A Gröbner basis engine is used as the underlying computational framework. The efficiency of Gröbner basis computations depends heavily upon the variable (and term) ordering used to represent and manipulate the polynomials. We present a variable (and term) ordering heuristic that significantly improves the efficiency of Gröbner basis engines. Using our approach, we can verify the correctness of up to 96-bit multipliers, whereas contemporary BDDs/SAT/SMT-solver based methods are infeasible.

I. INTRODUCTION

Finite (Galois) field theory is extensively applied in Elliptic Curve Cryptography, error correction codes, digital signal processing, etc. Therefore, dedicated hardware implementations of Galois field arithmetic abound. Multiplication lies at the core of most Galois field computations – where two \(k\)-bit inputs \(A, B\) are multiplied modulo an irreducible polynomial \(P(x)\) over the field \(\mathbb{F}_2\). Incorrect (buggy) multiplication can lead to full leakage of the secret key in cryptosystems. Therefore, it is of utmost importance to verify the correctness of hardware implementations of finite field multipliers residing at the core of such systems. This paper addresses formal verification of multiplier circuits over (binary) Galois fields of the type \(\mathbb{F}_2\) using computer algebra techniques.

Problem Statement: Let \(F\) represent a word-level (multiplier) specification such that \(F = A \cdot B \mod P(x)\), where \(A\) and \(B\) are elements in \(\mathbb{F}_2\) (\(k\)-bit inputs), and \(P(x)\) is an irreducible polynomial of \(\mathbb{F}_2\). Let \(G\) be a gate-level circuit implementation of \(F\). Our objective is to formally prove that the circuit \(G\) correctly computes the multiplication for all possible values of the inputs \(A, B\). Otherwise, we have to produce a counter-example that excites the bug in the design.

Approach and Contributions: We model the multiplier circuit as a polynomial system in \(\mathbb{F}_2[x_1, x_2, \cdots, x_d]\) and then formulate the equivalence test using the theory of Hilbert’s weak Nullstellensatz over Galois Fields \(\mathbb{F}_2\) (a proof-by-refutation approach). A Gröbner basis engine \([3]\) is used as the computational framework to solve the underlying polynomial decision problems.

A straight-forward application of our approach allows to verify only up to 48-bit multipliers. The source of this limitation is in the efficiency of Gröbner basis engines – which is highly susceptible to the effects of variable and term orderings used to represent and manipulate the polynomials. Therefore, to improve the efficiency and scalability of our approach, we present a new heuristic to derive a variable and term order for polynomial representation. Using this new term ordering, we are able to significantly enhance the efficiency of the Gröbner basis engine, and consequently scale our technique to verify up to 96-bit multipliers. Our experiments also demonstrate that contemporary approaches (BDDs, SAT- and SMT-solver based) are unable to verify the correctness of multipliers that are larger than 16-bits wide.

Our approach is generic enough to verify the implementation of any Galois field arithmetic circuit against its given polynomial specification. However, for this paper, we concentrate only on modulo multipliers over \(\mathbb{F}_2\) fields. Our approach can prove correctness or detect the presence of bugs. It cannot, however, find input vectors that excite the bugs in flawed designs (counter-examples). In such cases, we show that SMT-solvers are able to generate the counter-examples, complementing our technique.

Motivation for this work: The main motivation behind this work is to formally prove the correctness of Galois field multipliers for applications in Elliptic Curve Cryptography. In many organizations, cryptography implementations have to be certified correct before they can be deployed for secure communications. While SMT-solvers are able to catch bugs quickly, none of the contemporary decision procedures are able to formally prove the correctness of modulo-multiplier implementations (as demonstrated in our experiments). Our approach provides an automatic formal verification engine for correctness proof.

II. PRELIMINARIES: GALOIS FIELDS & MULTIPLICATION

We briefly describe the relevant finite field concepts (details can be found in the textbook \([4]\)) and modular multiplier design over such fields \([5]\ [6]\ [7]\ [8]\).

A finite field, also called a Galois field, is a field with a finite number of elements. The number of elements \(q\) of the
finite field is a power of a prime integer – i.e. $q = p^k$, where $p$ is a prime integer, and $k \geq 1$. Galois fields are denoted as $\mathbb{F}_q$ and also $GF(q = p^k)$. We are interested in fields where $p = 2$ and $k > 1$; i.e. binary Galois extension fields $\mathbb{F}_{2^k}$, as employed in Elliptic Curve Cryptography implementations.

To construct $\mathbb{F}_{2^k}$, we take the polynomial ring $\mathbb{F}_2[x]$ and an irreducible polynomial $P(x) \in \mathbb{F}_2[x]$ of degree $k$, and construct $\mathbb{F}_{2^k}$ as $\mathbb{F}_2[x] \mod P(x)$. For example, $\mathbb{F}_8 = \mathbb{F}_2[x]$ \mod $(x^3 + x + 1)$. All the field operations are performed modulo the irreducible polynomial $P(x)$ and the coefficients are reduced modulo $p = 2$. Any element $A \in \mathbb{F}_{2^k}$ can be represented in polynomial form as $A = a_0 + a_1x + \cdots + a_{k-1}x^{k-1}$, where $a_i \in \mathbb{F}_2 = \{0,1\}$ and $x$ is the root of the irreducible polynomial.

We will employ an important property of Galois fields [4]: For all elements $A \in \mathbb{F}_q, A^q = A$, and hence $A^q - A = 0, \forall A \in \mathbb{F}_q$.

We now illustrate multiplication over $\mathbb{F}_{2^k}$ through the following example.

**Example II.1:** Let us consider the field $\mathbb{F}_{2^4}$. We take as inputs: $A = a_0 + a_1x + a_2x^2 + a_3x^3$ and $B = b_0 + b_1x + b_2x^2 + b_3x^3$, along with the primitive polynomial $P(x) = x^4 + x^3 + 1$. We have to perform the multiplication $F = A \times B \mod P(x)$. The coefficients of $A = \{a_0, a_1, a_2, a_3\}, B = \{b_0, b_1, b_2, b_3\}$ are in $\mathbb{F}_2 = \{0,1\}$. So we can perform this multiplication as shown below:

\[
\begin{array}{cccc}
 & a_3 & a_2 & a_1 & a_0 \\
\hline
a_3 & b_3 & a_2b_3 & a_1b_3 & a_0b_3 \\
\hline
a_2 & b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
\hline
a_1 & b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
\hline
a_0 & b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
\hline
\end{array}
\]

In polynomial expression, we have the result as: $S = s_0 + s_1x + s_2x^2 + s_3x^3 + s_4x^4 + s_5x^5 + s_6x^6$, where, $s_0 = a_0b_0, s_1 = a_0b_1 + a_1b_0, s_2 = a_0b_2 + a_1b_1 + a_2b_0$, and so on. Here the multiply “$\cdot$” and add “$+$” operations are performed modulo 2, so they can be implemented in a circuit using AND and XOR gates. Note that unlike integer multipliers, there are no carry-chains in the design, as the coefficients are always reduced modulo $p = 2$. However, the result is yet to be reduced modulo the primitive polynomial $P(x) = x^4 + x^3 + 1$. This is shown below:

\[
\begin{array}{cccc}
& x_3 & x_2 & x_1 & x_0 \\
\hline
x_3 & 0 & 0 & x_3 & \text{(mod $P(x)$)} = x_3 \cdot (x^3 + 1) \\
x_2 & 0 & x_0 & \text{(mod $P(x)$)} = x_0 \cdot (x^3 + x + 1) \\
x_1 & 0 & x_2 & \text{(mod $P(x)$)} = x_2 \cdot (x^3 + x + 1) \\
x_0 & 0 & x_1 & \text{(mod $P(x)$)} = x_1 \cdot (x^3 + x + 1) \\
\hline
g_3 & g_2 & g_1 & g_0 \\
\hline
\end{array}
\]

The final result (output) of the circuit is: $G = g_0 + g_1x + g_2x^2 + g_3x^3$, where $g_0 = s_0 + s_4 + s_5 + s_6; \ g_1 = s_1 + s_4 + s_6; \ g_2 = s_2 + s_4; \ g_3 = s_3 + s_4 + s_5 + s_6$.

The above multiplier design is called the Mastrovito multiplier [6] [5]. In cryptosystems, multiplication is often performed repeatedly – e.g., for exponentiation. For such applications, Montgomery multiplier architectures [7] [8] over Galois fields are employed for faster computation (at the expense of area).

In our experiments, we verify custom implementations of both Mastrovito and Montgomery multipliers. Because of their obvious complexity, Montgomery multipliers are harder to verify than the ones based on Mastrovito design style.

### III. Limitations of Previous Work

Contemporary graph-based canonical DAG representations of Boolean functions such as BDDs [9], OKFDDs [10], BMDs [11], etc. are ill-suited for verification of such modular-multiplication applications, particularly over large finite fields (as also demonstrated in [12]). This verification problem is also very hard for SAT solvers, due to the large size and the presence of AND-XOR terms in the design. Contemporary Satisfiability Modulo Theory (SMT) solvers employ a mixture of theories for reasoning – however, none of them employ polynomial equation solving over finite fields (which is a very hard problem in itself). Therefore, for our applications, we use the SMT solvers to model the circuit constraints using the theory of fixed-size bit-vectors (QF-BV). Our experiments, described in later sections, reveal that BDDs/SAT/SMT solvers cannot prove equivalence beyond 16-bit multipliers.

The theorem-proving approach of [13] comes closest to ours, as they also verify a finite field $\mathbb{F}_{2^k}$ implementation against a given specification. They devise a decision procedure based on polynomial division, variable elimination, term re-writing, etc., to verify Galois field arithmetic. However, their correctness criterion is $2^k$-field-size independent. If this condition is not satisfied, then their approach requires decision over $\mathbb{F}_2$, and that really limits the scalability of their approach.

Gröbner basis based algebraic techniques have been utilized for SAT solving and formal verification: [14] [15] [16] [17]. However, these methods are employed for verification by modeling constraints over $\mathbb{F}_2 = \mathbb{Z}_2$ which leads to the inefficiency of these approaches. The recent work of [18] employs computer algebra over finite integer rings $\mathbb{Z}_{2^k}$ for verification of finite word-length integer datapaths. None of the above works have successfully solved the modulo-multiplication verification problem over $\mathbb{F}_{2^k}$. The work of [19] verifies a given composite field $\mathbb{F}_{2^{2n}}$ multiplier against its specification using a Gröbner basis approach which is similar to the setup in this paper. However, that approach has the limitation that it requires that the circuit decomposition hierarchy be made available to the verification engine.

### IV. Verification Setup and Polynomial Modeling

Our verification setup is formulated as an equivalence check between the polynomial specification and the circuit implementation. The setup (minter) is depicted in Fig. 1. Given the specification polynomial $F = A \cdot B \mod P(x)$, and the circuit implementation $G$ (as shown in Example II.1), we want to prove that for all possible inputs, $F$ is always equal to $G$ over $\mathbb{F}_{2^k}$. This can, conversely, also be solved as proving that $F \neq G$ has no solutions. The existence of no solutions to $F \neq G$ proves their equivalence. Whereas, a solution to $F \neq G$ implies a bug.

To test whether $F \neq G$ has no solutions: (i) we represent all
the constraints corresponding to our setup of Fig. 1 as a
polynomial system over the Galois field $\mathbb{F}_q$; and subsequently (ii) we use techniques from symbolic computer algebra, specifically Hilbert’s Nullstellensatz and Gröbner bases [2], to de-
duce whether or not the polynomial system has any solutions
over the given field.

The specification is provided in word-level polynomial form as follows: $A = a_0 + a_1 \cdot x + \cdots + a_{k-1} \cdot x^{k-1}$, $B = b_0 + b_1 \cdot x + \cdots + b_{k-1} \cdot x^{k-1}$ and $F = A \cdot B \pmod{P(x)}$, where $A, B \in \mathbb{F}_2$ ($a_i, b_i \in \mathbb{F}_2$) symbolically represent the inputs, and $F \in \mathbb{F}_2$ represents the result of the computa-
tion.

The implementation is given as a circuit netlist. We map the gate-level Boolean operators (AND, OR, NOT, XOR) to polynomials over $\mathbb{F}_2(\mathbb{F}_2)$ (as shown in Example II.1) using the following equalities:

\[
\begin{align*}
  f : B & \rightarrow F_2 \\
  a \leftrightarrow a + 1 & \pmod{2}; \quad a \vee b \leftrightarrow a + b + a \cdot b \pmod{2} \quad \text{(mod 2)} \\
  a \wedge b & \leftrightarrow a \cdot b \pmod{2} \quad \text{(mod 2)} \\
  a \oplus b & \leftrightarrow a + b \pmod{2} \quad \text{(mod 2)}
\end{align*}
\]

Therefore, we can represent the implementation also in
polynomial form over $\mathbb{F}_2$. Let $G$ symbolically denote word-
level result of the implementation (output of the circuit).

The $F \neq G$ constraint is also modeled as a polynomial over
$\mathbb{F}_2$ as follows: $t(F - G) = 1$, where $t \in \mathbb{F}_2$. The correctness
of the above constraint modeling can be shown as follows: (i) When $F = G, F - G = 0$, so $t \cdot 0 = 1$ has no solutions. (ii) When $F \neq G, (F - G) \neq 0$. Over any field, every non-zero element has a multiplicative inverse. Let $t^{-1} = (F - G)$. Then $t \cdot t^{-1} = 1$ will always have a solution.

To summarize, the constraints corresponding to the entire
verification instance (miter) – the specification, implementa-
tion, and the $F \neq G$ constraint – can be modeled as a poly-
nomial equation system in $\mathbb{F}_2$. Subsequently, we can reason
whether or not solutions exist to this polynomial system using
a computer algebra approach – which is described next.

V. VERIFICATION USING HILBERT’S NULLSTELLENSATZ
AND GRÖBNER BASES

Let $\mathbb{F}$ be any field and $\mathbb{F}[x_1, \ldots, x_d]$ the polynomial
ring over $\mathbb{F}$ with indeterminates $x_1, \ldots, x_d$. An ideal
$\mathfrak{I}$ generated by $f_1, \ldots, f_r \in \mathbb{F}[x_1, \ldots, x_d]$ is $\mathfrak{I} = \langle f_1, \ldots, f_r \rangle = \{ h = \sum_{i=1}^r g_i \cdot f_i : g_i \in \mathbb{F}[x_1, \ldots, x_d] \}$. Suppose that we are given a set of polynomials $f_1, f_2, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_d]$ and that we wish to find solutions to the polynomial system $f_1 = f_2 = \cdots = f_s = 0$. The set of all

solutions to a given system of polynomial equations is called
the variety, and is denoted by $V(f_1, \ldots, f_s)$.

Theorem VI: Weak Nullstellensatz over $\mathbb{F}_q$ [20]: Let
$f_1, \ldots, f_k \in \mathbb{F}_q[x_1, \ldots, x_d]$ generate an ideal $\mathfrak{I} = \langle f_1, \ldots, f_k \rangle \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$. Let ideal $\mathfrak{I}_0 = \langle x_1^q - x_1, x_2^q - x_2, \ldots, x_d^q - x_d \rangle$ and denote $(I, I_0) = \langle f_1, \ldots, f_k, x_1^q - x_1, \ldots, x_d^q - x_d \rangle$. Let $V_{\mathbb{F}_q}(I)$ denote the variety of $I$ over the Galois field $\mathbb{F}_q$. Then $V_{\mathbb{F}_q}(I) = \emptyset$ if and only if $1 \in (I, I_0)$.

A detailed proof of this result is given in [20]. We briefly
describe the significance of this result and how it applies to
our work.

- We represent the polynomial constraints corresponding to
  our verification setup of Fig. 1 as the ideal $I = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$, where $q = 2^k$.
- Then we generate another set of polynomials representing
  ideal $I_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$, for all the
  variables $\{x_1, \ldots, x_d\}$ in our system.
- Subsequently, we append the polynomials of $I_0$ to those of $I$ and
  generate the ideal $(I, I_0) = \langle f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_d^q - x_d \rangle$.
- Now we have to test if the constant polynomial 1 is a member
  of the ideal $(I, I_0)$.
- If indeed $1 \in (I, I_0)$, then $V_{\mathbb{F}_q}(I) = \emptyset$; i.e. the
  polynomial system $f_1 = \cdots = f_s = 0$ has no solution over the given field
  $\mathbb{F}_q$. This implies that our miter constraints are infeasible, and
  the implementation ($G$) is equal to the specification ($F$).
- On the other hand, if $1 \notin (I, I_0)$, then it means that there exist
  a finite number of solutions within the field $\mathbb{F}_q$, ($V_{\mathbb{F}_q}(I) \neq 0$),
  and that there are definitely bug(s) in the implementation [20].

The significance of ideal $I_0$: Recall from Section II, for all
elements $A$ of a Galois field $\mathbb{F}_q$, $A^q = A$, or $A^q - A = 0$. In
other words, $A^q - A$ vanishes $\forall A \in \mathbb{F}_q$. The ideal $I_0 = \langle x_1^q - x_1 \rangle$ is, therefore, also called the ideal of all vanishing polynomials
of the field $\mathbb{F}_q$. In general, the Weak Nullstellensatz deduces
the variety over an algebraically closed field. Galois fields are,
however, not algebraically closed. These vanishing polynomi-
als $(x_i^q - x_i)$ restrict the variety to $\mathbb{F}_q$, discarding the solutions
over the algebraic closure of $\mathbb{F}_q$.

To solve our verification problem, we have to test if $1 \in
(I, I_0)$. For this test, it is required to compute a Gröbner basis
of $(I, I_0)$, for which, Buchberger’s algorithm is used, as shown
in Algorithm 1.

![Fig. 1: The Equivalence Verification Setup: Miter](image-url)

Input: $F = \{f_1, \ldots, f_s\}$
Output: $G = \{g_1, \ldots, g_r\}$
$G := F$;
REPEAT
    $G' := G$
    For each pair $\{f, g\}, f \neq g$ in $G'$ DO
        $S(f, g) \rightarrow x \quad r$
        IF $r \neq 0$ THEN $G := G \cup \{r\}$
UNTIL $G = G'$

Algorithm 1: Buchberger’s Algorithm

For Gröbner basis computation, a monomial (term) or-
dering is fixed to ensure that polynomials are manipulated
in a consistent manner. Buchberger’s algorithm then takes pairs of polynomials in the basis and combines them into “S-polynomials” to cancel leading terms. An S-polynomial is defined as:

\[ S(f,g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g \]

where \( L = \text{LCM}(lm(f), lm(g)) \), where \( lm(f) \) is the leading monomial of \( f \), and \( lt(f) \) denotes the leading term of \( f \). The S-polynomial is then reduced (divided) by all elements of \( G' \) to a remainder \( r \), denoted as \( S(f,g) \xrightarrow{G}\rightarrow r \). Multivariate polynomial division is used for this reduction step. This process is repeated for all unique pairs of polynomials, including those created by newly added elements, until no new polynomials are generated; ultimately constructing the Gröbner basis.

A Gröbner basis can be “reduced” further (details given in [3]). A reduced Gröbner basis is a canonical representation of the ideal w.r.t. the imposed monomial order, and it provides a decision procedure to test if \( \mathcal{I} \subseteq \mathcal{I}_0 \).

**Theorem V.2.** (From [3]) Let \( \mathcal{I} = \{f_1, \ldots, f_s\} \subset \mathbb{F}[x_1, \ldots, x_d] \) be an ideal. Let \( G \) denote the reduced Gröbner basis of \( \mathcal{I} \). Then \( V(\mathcal{I}) = \emptyset \) if and only if \( G = \{1\} \).

Thus, to test if \( \mathcal{I} \subseteq \mathcal{I}_0 \), we need to compute the reduced Gröbner basis \( G \) of the ideal \( \{\mathcal{I}, \mathcal{I}_0\} \) and check if \( G = \{1\} \).

Using this formulation, we have performed verification for correct and buggy implementations of both Mastrovito and Montgomery designs. We use the SINGULAR computer algebra tool [21] to compute the Gröbner basis. The experiments are described in Section VII, and we discuss them later. However, we make the following observations here: To compute a Gröbner basis, an ordering needs to be imposed on the monomial terms of the polynomials. Conventionally, lexicographic (lex), degree-lexicographic (deg-lex) and degree-reverse-lexicographic (degrevlex) orderings are used [2]. Term orderings have a significant impact on the efficiency of Gröbner basis computations. When we use these conventional lex, deglex and degrevlex orderings, we are able to verify only up to 48-bit circuits in \( \mathbb{F}_{2^{48}} \); beyond which SINGULAR runs out of memory. To improve the scalability of our approach, we now present a variable (and term) ordering heuristic that allows to verify up to 96-bit circuits (\( \mathbb{F}_{2^{96}} \)).

**VI. TERM ORDERING TO IMPROVE GRÖBNER BASIS COMPUTATION**

The critical computation in Gröbner bases algorithm is that of S-polynomials and their subsequent reduction: \( S(f,g) \xrightarrow{f+} r \). A rudimentary implementation of Gröbner bases algorithm has the limitation that too many S-polynomials might be created, many of which may actually reduce to 0, thus wasting computation time. Leading monomials play a major role in computing \( S(f,g) \). Different monomial orderings may lead to different S-polynomials and hence a different Gröbner basis. This is demonstrated below:

**Example VI.1:** Consider the polynomials: \( f_1 = x_0 x_1 + x_2 \), \( f_2 = x_1 x_2 + x_3 \). The above representation is obtained if the lex term order, with variable order \( x_0 > x_1 > x_2 > x_3 \), is chosen.

Here, the leading monomials \( lm(f_1) = x_0 x_1 \), \( lm(f_2) = x_1 x_2 \), and \( f_3 = f_1 f_2 = -x_0 x_3 + x_2^2 \). The Gröbner basis computed with this order contains these three polynomials \( \{f_1, f_2, f_3\} \). If, however, a lex order with \( x_1 > x_0 > x_2 > x_3 \) is used, then \( f_1 = x_2 + x_1 \cdot x_0 \), with \( lm(f_1) = x_2 \) and \( f_2 = x_1 + x_3 \cdot x_1 \) with \( lm(f_2) = x_3 \) and \( S(f_1, f_2) = x_3 (x_1, x_3) - x_3^2 x_1 \). We see the benefit of this order as this S-poly reduces to 0 modulo \( \{f_1, f_2\} \), i.e. \( S(f_1, f_2) \nmid f_3 \), implying that \( \{f_1, f_2\} \) is itself a Gröbner basis with this term order.

We wish to heuristically derive a variable and term order such that fewer S-polynomials are computed in the Gröbner basis algorithm. Our variable ordering heuristic is inspired by Buchberger’s Product Criterion [22]:

**Lemma VI.1 (Product Criterion)** Let \( f, g \in \mathbb{F}[x_1, \ldots, x_d] \) be polynomials. If the equality \( lm(f) \cdot lm(g) = \text{LCM}(lm(f), lm(g)) \) holds, then \( S(f,g) \xrightarrow{G}\rightarrow 0 \).

The above result states that when the leading monomials of \( f, g \) are relatively prime, then \( S(f,g) \) always reduces to 0 modulo \( G' \). Thus \( S(f,g) \) need not be considered. Modern computer algebra engines always perform this check before computing \( S(f,g) \). Note that, in Example VI.1, the lex term order with \( x_3 > x_2 > x_1 > x_0 \) makes \( lm(f_1) \) and \( lm(f_2) \) relatively prime, which obviates the need to compute \( S(f_1, f_2) \) and thus \( G = \{f_1, f_2\} \) itself constitutes a Gröbner basis. Our variable ordering heuristic exploits the above result to derive a variable and term order such that leading monomials of most polynomials become relatively prime, and that speeds-up the computation.

Algorithm 2 presents our WVAO (Weighted Variable Activity Order) ordering heuristic. For every variable, we introduce a measure (or count), called Weighted Variable Activity (WVA). Intuitively, WVA corresponds to the frequency of a variable’s occurrence among polynomials, weighted according to the size of the term in which the variable appears. Moreover, if a variable appears in more than one polynomial, its WVA measure is further enhanced by a multiplicative factor. The algorithm inputs the polynomials and iteratively updates the WVA measure for each variable. Finally, the variables are sorted according to ascending order of their WVA count. Using this variable order, a lexicographic (lex) term ordering is imposed on the polynomials.

The motivation behind this approach is that if a variable has lower WVA count, it appears infrequently in the terms. If terms with variables that have lower WVA count are made leading terms, then they are more likely to be relatively prime; and thus avoid S-polynomials computations. We illustrate our algorithm using the example below:

**Example VI.2:** Consider the following polynomials extracted from a 2-bit multiplier over \( \mathbb{F}_2^2 \): \( f_1 = a_0 \cdot b_0 + r_0 \); \( f_2 = a_0 \cdot b_1 + a_1 \cdot b_0 + r_1 \); \( f_3 = a_1 \cdot b_1 + r_2 \); \( f_4 = r_0 + r_2 + y_0 \); \( f_5 = r_1 + r_2 + y_1 \). Polynomial \( f_1 \) is input first; it contains two terms \( a_0 \cdot b_0 \) and \( r_0 \). Since variables \( a_0, b_0, r_0 \) are encountered for the first time, (line 5 in the algorithm), their WVAs are assigned according to the size of their terms (Line 6 – 9): WVA(\(a_0\))=2, WVA(\(b_0\))=2, WVA(\(r_0\))=1. Next, \( f_2 \) is input;
Input: polys: polynomials extracted from given circuit
Output: Static Variable Order
1: WVA= Initialize to zero for input variables;
2: for (i=0; i < number of polynomials; i++) do
3:   term_list[i]=Extract_Term(polys[i]);
4:   for all var, term ∈ term_list[i] do
5:     occurrence=Search(var, var_list[0, ..., j - 1]);
6:   if occurrence==j then
7:     /*First occurrence of var*/
8:     WVA[var]=Var_Num_In_Term(term);
9:   end if
10: if occurrence < j && Same_Poly(var)==1 then
11:   /*var has occurred in same polynomial*/
12:   WVA[var]+=2 Var_Num_In_Term(term);
13: end if
14: if occurrence < j && Same_Poly(var)==0 then
15:   /*var already appeared in a different polynomial*/
16:   WVA[var]+=4 Var_Num_In_Term(term);
17: end if
18: end for
19: end for
20: /*Sort variables in terms of WV A in ascending order*/
21: Sort(WVA[0, ..., j]);
22: return WVA;

Algorithm 2: WVA Ordering Algorithm

it contains three terms $a_0 \cdot b_1$, $a_1 \cdot b_0$, and $r_1$. Since $a_0$ has already appeared in another polynomial $f_1$, its WVA is updated according to lines (14 – 17): WVA($a_0$) = 10. Similarly, WVA($b_0$) = 10. Since $a_1$ and $b_1$ appear for the first time, their WVAs are both 2. Similarly, WVA($r_1$) = 1. In this fashion, the WVA counts for all variables are updated until all polynomials are analyzed. This is illustrated in Table I. The variables are sorted in ascending order of their WVA counts, which results in $y_0 > y_1 > r_0 > r_1 > r_2 > a_0 > a_1 > b_0 > b_1$. Finally, a lex term order, with the above variable order, is used to represent the polynomials. When the polynomials are represented with the generated term order, we can see that $lm(f_1) = r_0$, $lm(f_2) = r_1$, $lm(f_3) = r_2$, $lm(f_4) = y_0$, and $lm(f_5) = y_1$. The leading terms of all polynomials become relatively prime, and no $S$-polynomial needs to be computed – and $\{f_1, f_2, f_3, f_4, f_5\}$ is itself a Gröbner basis using this order.

TABLE I: WVAO Algorithm execution for Example VI.2.

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$b_0$</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$y_0$</th>
<th>$y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_3$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$f_4$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>$f_5$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

Obtained WVAO in Ascending Order

WVAO $y_0$ $y_1$ $r_0$ $r_1$ $r_2$ $a_0$ $a_1$ $b_0$ $b_1$

VII. EXPERIMENTAL RESULTS

We have conducted experiments to verify custom implementations of both Mastrovito and Montgomery multipliers, which were derived from [6] [5] [7] [8]. We use the computer algebra tool SINGULAR [v. 3-1-2] [21] to compute the Gröbner basis using the slimgb command. Our experiments are conducted on a desktop with 2.40GHz Intel Core(TM)2 Quad CPU and 2GB memory running 64-bit Linux.

Experiments with Singular: When our circuits are correctly designed, we do observe that the reduced Gröbner basis $GB(I, I_0) = \{1\}$, thus proving the equivalence. Results of the verification of Mastrovito multipliers using SINGULAR are shown in Table II. The results are shown for various variable orderings. For lp (lex), dp (degrevlex) and Dp (deglex), the best variable order found was “primary inputs $>$ intermediate variables $>$ primary outputs”. Of these, the lex order performs best and we can verify up to 48-bit multipliers. Beyond that, the Gröbner basis creates too many polynomials and SINGULAR runs out of memory. However, using our proposed order (WVAO), we are able to extend the verification for up to 96-bit multipliers. Moreover, our term order also leads to faster reduced Gröbner basis computation. For example, for 32-bit multipliers, our term order completes verification within 52.76s, as compared to 2509.57s required for the Dp order.

The experiments for Montgomery multiplier verification are shown in Table III. The implementation of Montgomery multiplier was verified against the specification. Using the same lp, Dp and dp orders, we can verify only up to 44-bit multipliers, whereas our WVA order extends the verification up to 64-bit fields. Beyond the 64-bit multipliers, our WVA order also results in memory overflow in the Gröbner basis computation.

Table IV shows the verification results using BDDs, SAT and SMT solvers. To verify that the multiplier implementation is bug-free, we use BDDs/SAT/SMT-solvers to prove that the miter of Fig. 1 is infeasible. None of BDDs, SAT or SMT solvers can verify the correctness of circuits that are larger than 16-bits wide.

TABLE II: Verification of correct Mastrovito Multipliers using SINGULAR.

<table>
<thead>
<tr>
<th>Stats</th>
<th>Word size of the operands k-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>#variables</td>
<td>#terms</td>
</tr>
<tr>
<td>lex</td>
<td>dp</td>
</tr>
</tbody>
</table>

Experiments with Bugs: Our method can only detect the presence or absence of bugs (depending upon whether $1 \in (I, I_0)$). It cannot generate counter-examples that excite the bug. However, SMT solvers can identify bugs and gener-
ate counter-examples quickly. We created buggy implementations by incorrectly connecting some signals in the design. Bug-catching results are shown in Table V. The SMT-solver YICES outperforms all others for bug catching.

VIII. Conclusions

This paper has presented a formal approach to model and verify multiplier circuits over Galois fields \( \mathbb{F}_2^m \) using a computer-algebra based approach. We model the verification test as a Nullstellensatz proof over \( \mathbb{F}_2^m \), using a Gröbner basis engine. We analyze the verification constraints and derive a term order for efficient Gröbner basis computation. Using our approach, we are able to verify the correctness of up to 96-bit multipliers over \( \mathbb{F}_2^96 \), whereas conventional techniques based on SAT/SMT/BDD solvers are infeasible.

REFERENCES