EFFICIENT GRÖBNER BASIS REDUCTIONS FOR FORMAL VERIFICATION OF GALOIS FIELD ARITHMETIC CIRCUITS

Jinpeng Lv[1], Priyank Kalla[2], Florian Enescu[3]
[1]Cadence Design Systems, Conformal Group, San Jose, USA
[2]Electrical and Computer Engineering, Univ. of Utah, USA
[3]Mathematics and Statistics, Georgia State University, USA
Contact: kalla@ece.utah.edu

***Abstract*—Galois field arithmetic is a critical component in communication and security-related hardware, requiring dedicated arithmetic circuit architectures for greater performance. In many Galois field applications, such as cryptography, the data-path size in the circuits can be very large. Formal verification of such circuits is beyond the capabilities of contemporary verification techniques. This paper addresses formal verification of combinational arithmetic circuits over Galois fields of the type $\mathbb{F}_{2^k}$ using a computer-algebra/algebraic-geometry based approach.**

The verification problem is formulated as membership testing of a given specification polynomial in a corresponding ideal generated by the circuit constraints. Ideal membership testing requires the computation of a Gröbner basis, which is computationally very expensive. To overcome this limitation, we analyze the circuit topology and derive a term order to represent the polynomials. Subsequently, using the theory Gröbner bases over $\mathbb{F}_{2^k}$, we show that this term order renders the set of polynomials itself a minimal Gröbner basis of this ideal. Consequently, the verification test reduces to a much simpler case of Gröbner basis reduction via polynomial division, significantly enhancing verification efficiency.

To further improve our approach, we exploit the concepts presented in the $F4$ algorithm for Gröbner basis, and show that our verification test can be formulated as Gaussian elimination on a matrix representation of the problem. Finally, we demonstrate the ability of our approach to verify the correctness of, and detect bugs in, up to 163-bit circuits in $\mathbb{F}_{2^{163}}$ — whereas verification utilizing contemporary techniques proves infeasible.

Keywords — Formal verification, Galois fields, arithmetic circuits, computer algebra, Gröbner bases.

I. INTRODUCTION

With the spread of Internet and mobile devices, transferring information safely and securely has become more important than ever. Galois fields have widespread applications in such domains, such as in cryptography, error correction codes, signal processing, etc. Therefore, dedicated hardware (VLSI) implementations of Galois field arithmetic abound [1] [2] [3] [4] [5]. In most practical applications, the field size — and therefore the word-lengths of the operands — can be very large. For example, the U.S. National Institute for Standards and Technology (NIST) recommends the use of Galois fields corresponding to data-path sizes of 163-bits or more for elliptic curve cryptography. The high complexity of arithmetic operations over such large fields requires circuits to be (semi-) custom designed — increasing the likelihood of errors/bugs in the implementation. Such bugs not only cause unintended operations, but they also manifest themselves as *security vulnerabilities* open for exploitation. It has been shown in [6] that arithmetic bugs in crypto-systems can lead to full leakage of the secret key. Formal verification of Galois field arithmetic circuits is therefore imperative.

This paper addresses the problem of *formal verification of combinational circuits that implement Galois field arithmetic computations.* We consider Galois fields of the type $\mathbb{F}_{2^k}$ — i.e. binary Galois extension fields — as these are often the fields of choice for efficient hardware implementations. Galois field arithmetic circuits are implementations of some specification polynomial f. The specification f may not be limited to simple polynomial computations such as multiplication ($f = A \cdot B$) or squaring ($f = X^2$), but may also specify entire systems, such as point-addition on elliptic curves over $\mathbb{F}_{2^k}$. Given f as a specification, and an arithmetic circuit as its implementation, the purpose of the verifier is to ensure that the circuit implementation is equivalent to the specification. More formally, the verification problem is stated as follows:

- Given a Galois field $\mathbb{F}_{2^k}$, i.e. given k, along with the irreducible polynomial $P(x)$ used for field construction.
- The *specification* is given as a multi-variate polynomial f with coefficients from $\mathbb{F}_{2^k}$.
- The *implementation* is given as a gate-level *combinational* circuit C.

Our *objective* is to prove that the circuit C correctly implements the polynomial f. Otherwise we have to generate a counter-example that excites the bug in the design. This paper targets verification of only combinational Galois field circuits. Verification of sequential Galois field circuits is a different problem, and is beyond the scope of this paper.

A. Approach and Contributions

Our technique utilizes concepts from computer-algebra and algebraic geometry as the core verification framework. This enables us to formulate the verification problem as an ideal membership test using the *Strong Nullstellensatz* [7] over Galois fields. Subsequently, *Gröbner basis* techniques are employed for this ideal membership test. As Galois field arithmetic circuits perform computations that are algebraic in nature, computer-algebra based formulations and decision procedures provide an efficient and scalable means to address the verification problem. Our approach and contributions can be outlined as follows:

- Using polynomial abstractions, the specification and the implementation circuit are modeled as elements of a multivariate polynomial ring with coefficients from $\mathbb{F}_{2^k}$.

- Using the concepts of *Strong Nullstellensatz* [7] over Galois fields, we deduce that the verification problem can be formulated as *membership testing* of the specification polynomial f in a corresponding (radical) *ideal* generated by the circuit constraints.
- Ideal membership testing requires the computation of a *Gröbner basis* [8]. Buchberger's algorithm [9], employed for Gröbner basis computation, exhibits high computational complexity — which is critically tied to the *term ordering* used to represent and manipulate the polynomials. To overcome this limitation, we show that a specialized term ordering can be derived by analyzing the topology of the given circuit. Subsequently, we prove that this term ordering renders the set of polynomials itself a Gröbner basis — *thus obviating the need for Buchberger's algorithm*. As a consequence of our deductions, the verification (ideal membership) test reduces to a much simpler case of Gröbner basis reduction via polynomial division.
- We further demonstrate how this Gröbner basis can be transformed into a *minimal* Gröbner basis directly by construction, simplifying the reduction procedure.
- Our approach only requires a polynomial reduction (divisions), enabling any general-purpose computer algebra tool (*e.g.* SINGULAR [10]) to be employed for such purposes. Efficient polynomial reduction techniques, such as those based on the $F4$ [11] algorithm, have been proposed in literature. We show that *our term ordering can be further exploited to engineer an efficient $F4$-style polynomial reduction procedure implemented as Gaussian elimination on a (dense) matrix representation of the problem.*
- We implement the technique as a standalone, custom verification tool for Galois field arithmetic circuits. Experiments conducted over various custom-designed arithmetic circuits demonstrate the efficiency and scalability of our methods. We are able to verify the correctness of, and detect bugs in, up to 163-bit circuits in $\mathbb{F}_{2^{163}}$, whereas contemporary techniques are infeasible beyond 16-bit circuits.

Paper Organization: The rest of the paper is organized as follows. In the next section, we review the relevant concepts of Galois fields $\mathbb{F}_{2^k}$ and describe the architectures of the digital circuits that we have designed and verified through our approach. Section III reviews related previous work. Section IV reviews preliminary computer algebra concepts of ideals, varieties and Nullstellensatz, and how they apply over Galois fields. Section V describes our problem formulation using Strong Nullstellensatz and Gröbner bases. Section VI shows how a term ordering is derived from the circuit that renders the set of polynomials corresponding to the verification instance itself a (minimal) Gröbner basis. In Section VII, we show how $F4$-style Gröbner basis reduction can be devised on a matrix for our specific problem. Section VIII describes the experiments conducted and analyzes the results. Finally, Section IX concludes the paper.

II. GALOIS FIELDS & HARDWARE DESIGN

We briefly describe the relevant concepts related to Galois fields $\mathbb{F}_{2^k}$; for more details, interested readers may refer to the textbook [12]. We also review some VLSI architectures used for Galois field computations [1] [2] [13] [3] [5] [14]. In our experiments, we have verified custom designs based on these architectures.

A Galois field is a field with a finite number of elements. The number of elements q of the field is a power of a prime integer — i.e. $q = p^k$, where p is a prime integer, and $k \geq 1$ is a positive integer. 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 they are widely employed in hardware implementations of cryptography primitives.

To construct $\mathbb{F}_{2^k}$, we take the polynomial ring $\mathbb{F}_2[x]$, where $\mathbb{F}_2 = \{0, 1\}$, and an irreducible polynomial $P(x) \in \mathbb{F}_2[x]$ of degree k, and construct $\mathbb{F}_{2^k}$ as $\mathbb{F}_2[x] \pmod{P(x)}$. For example, $\mathbb{F}_8 = \mathbb{F}_2[x] \pmod{x^3 + x + 1}$.

The *characteristic* of any finite field with unity element 1 is the least integer n such that $1 + \cdots + 1$ *(n times)* $= 0$. The characteristic of fields of the type $\mathbb{F}_{p^k}$ is the prime integer p. Since in our case $p = 2$, all fields of the type $\mathbb{F}_{2^k}$, for any given k, have characteristic 2. As a result, all field operations are performed modulo the irreducible polynomial $P(x)$ and the coefficients are reduced modulo $p = 2$; due to which $-1 = +1$ over $\mathbb{F}_{2^k}$.

Any element $A \in \mathbb{F}_{2^k}$ can be represented in polynomial form as $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$, where $a_i \in \mathbb{F}_2, i = 0, \ldots, k-1$, and α is the root of the irreducible polynomial, *i.e.* $P(\alpha) = 0$. The field $\mathbb{F}_{2^k}$ can therefore be construed as a k-dimensional vector space over $\mathbb{F}_2$.

An important property of Galois fields is that for all elements $A \in \mathbb{F}_q, A^q = A$, and hence $A^q - A = 0$. Therefore, the polynomial $x^q - x$ *vanishes* on all points in $\mathbb{F}_q$. Such *vanishing polynomials* will form an important part of our ideal membership formulation.

A. Hardware Implementations of Galois Field Arithmetic

In many Galois field applications, primitive computations such as ADD, MULT, INVERSE etc., are implemented in hardware, and application algorithms are then implemented in software (e.g. cryptoprocessors [14]). In other cases, the entire design can be implemented in hardware — such as the point-addition circuitry [5] used in elliptic curve cryptosystems.

As modular multiplication over $\mathbb{F}_{2^k}$ is at the heart of most public-key cryptosystems, efficient VLSI architectures have been introduced for this computation. These include the Mastrovito multiplication, Montgomery reduction [2] and the Barrett reduction [3].

Conceptually, the multiplication $Z = A \times B \pmod{P(x)}$ in $\mathbb{F}_{2^k}$ consists of two steps. First, $A \times B$ is computed, and then the result is reduced $\pmod{P(x)}$. A Mastrovito implementation [15] [1] is shown in the example below:

Example 2.1: Consider the field $\mathbb{F}_{2^4}$. We take as inputs: $A = a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3$ and $B = b_0 +$

$b_1 \cdot \alpha + b_2 \cdot \alpha^2 + b_3 \cdot \alpha^3$, *along with the irreducible polynomial* $P(x) = x^4 + x^3 + 1$. *We have to perform the multiplication* $Z = A \times B \pmod{P(x)}$. *The coefficients of* $A = \{a_0, \ldots, a_3\}, B = \{b_0, \ldots, b_3\}$ *are in* $\mathbb{F}_2 = \{0, 1\}$. *Multiplication can be performed as:*

				a_3	a_2	a_1	a_0
$\times$				b_3	b_2	b_1	b_0
				$a_3 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_0 \cdot b_0$
			$a_3 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_0 \cdot b_1$	
		$a_3 \cdot b_2$	$a_2 \cdot b_2$	$a_1 \cdot b_2$	$a_0 \cdot b_2$		
	$a_3 \cdot b_3$	$a_2 \cdot b_3$	$a_1 \cdot b_3$	$a_0 \cdot b_3$			
s_6		s_5	s_4	s_3	s_2	s_1	s_0

The result $Sum = s_0 + s_1 \cdot \alpha + s_2 \cdot \alpha^2 + s_3 \cdot \alpha^3 + s_4 \cdot \alpha^4 + s_5 \cdot \alpha^5 + s_6 \cdot \alpha^6$, *where,* $s_0 = a_0 \cdot b_0$, $s_1 = a_0 \cdot b_1 + a_1 \cdot b_0$, $s_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0$, *and so on. Here the multiply "·" 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:*

s_3	s_2	s_1	s_0	
s_4	0	0	s_4	$s_4 \cdot \alpha^4 \pmod{P(\alpha)} = s_4 \cdot (\alpha^3 + 1)$
s_5	0	s_5	s_5	$s_5 \cdot \alpha^5 \pmod{P(\alpha)} = s_5 \cdot (\alpha^3 + \alpha + 1)$
s_6	s_6	s_6	s_6	$s_6 \cdot \alpha^6 \pmod{P(\alpha)} = s_6 \cdot (\alpha^3 + \alpha^2 + \alpha + 1)$
z_3	z_2	z_1	z_0	

The final result (output) of the circuit is: $Z = z_0 + z_1\alpha + z_2\alpha^2 + z_3\alpha^3$; *where* $z_0 = s_0 + s_4 + s_5 + s_6$; $z_1 = s_1 + s_5 + s_6$; $z_2 = s_2 + s_6$; $z_3 = s_3 + s_4 + s_5 + s_6$.

In cryptosystems, multiplication is often performed repeatedly – e.g., for exponentiation. For such applications, Montgomery and Barrett architectures [2] [13] [16] [3] over Galois fields are employed for faster computation.

Montgomery Reduction: Montgomery reduction (MR) computes:

$$MR(A, B) = A \cdot B \cdot R^{-1} \pmod{P(x)}$$

where A, B are k-bit inputs, R is suitably chosen as $R = \alpha^k$, R^{-1} is multiplicative inverse of R in $\mathbb{F}_{2^k}$, and $P(x)$ is the irreducible polynomial. Since Montgomery reduction cannot directly compute $A \cdot B \pmod{P(x)}$, we need to pre-compute $A \cdot R$ and $B \cdot R$, as shown in Figure 1.

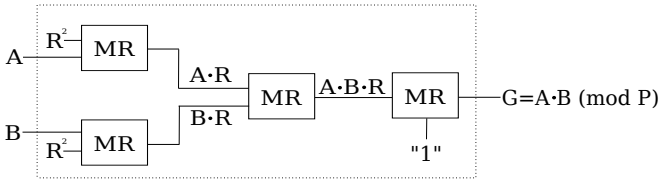

Fig. 1: *Montgomery* multiplication over $\mathbb{F}_{2^k}$ using four Montgomery reductions.

Each *MR* block in Fig. 1 represents a Montgomery reduction step, which is a hardware implementation of the algorithm shown in Algorithm 1. The algorithm is referred from [2].

Barrett Reduction: Similar to Montgomery reduction, traditional Barrett reduction [16] needs a pre-computed value of the reciprocal/inverse of modulus $P(x)$. The recent approach of [3] avoids such a pre-computation of inverses and simplifies the hardware implementation.

ALGORITHM 1: Montgomery Reduction Algorithm [2]

Input: $A, B \in \mathbb{F}_{2^k}$; irreducible polynomial $P(x)$.
Output: $Z = A \cdot B \cdot x^{-k} \pmod{P(x)}$.
Z :=0
for ($i = 0$; $i \leq k - 1$; ++i) **do**
  $Z := Z + A_i \cdot B$ /*A_i is the i^{th} bit of A*/;
  $Z := Z + Z_0 \cdot P(x)$
  /*Z_0 is the least significant bit of Z*/;
  $Z := Z/x$ /*Right shift Z by 1 bit*/;
end

Based on Barrett reduction, a multiplier can be designed in two steps: multiplication $R = A \times B$ and a subsequent Barrett reduction $G = R \pmod{P}$. In our experiments, we have verified custom implementations of each of the Mastrovito, Montgomery and Barrett multipliers.

Point Addition over Elliptic Curves: The main operations of encryption, decryption and authentication in elliptic curve cryptography (ECC) rely on *point additions* and *doubling* operations on elliptic curves designed over Galois fields. In general, this requires computation of multiplicative inverses over the field - which is expensive. Modern approaches represent the points in projective coordinate systems, *e.g.*, the L*ó*pez-Dahab (LD) projective coordinate [5], which eliminates the need for multiplicative inverses and improves the efficiency of these operations.

Example 2.2: Consider point addition in López-Dahab (LD) projective coordinate. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ *over* $\mathbb{F}_{2^k}$, *where* X, Y, Z *are* k*-bit vectors that are elements in* $\mathbb{F}_{2^k}$ *and similarly,* a, b *are constants from the field. Let* (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1) *represent point addition over the elliptic curve. Then* X_3, Y_3, Z_3 *can be computed as follows:*

$$\begin{aligned} A &= Y_2 \cdot Z_1^2 + Y_1 \\ B &= X_2 \cdot Z_1 + X_1 \\ C &= Z_1 \cdot B \\ D &= B^2 \cdot (C + aZ_1^2) \\ Z_3 &= C^2 \\ E &= A \cdot C \\ X_3 &= A^2 + D + E \\ F &= X_3 + X_2 \cdot Z_3 \\ G &= X_3 + Y_2 \cdot Z_3 \\ Y_3 &= E \cdot F + Z_3 \cdot G \end{aligned}$$

Example 2.3: Consider point doubling in LD projective coordinate system. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$. *Let* ($X_3$, Y_3, Z_3) = 2(X_1, Y_1, Z_1), *then*

$$X_3 = X_1^4 + b \cdot Z_1^4$$
$$Z_3 = X_1^2 \cdot Z_1^2$$
$$Y_3 = bZ_1^4 \cdot Z_3 + X_3 \cdot (aZ_3 + Y_1^2 + bZ_1^4)$$

In the above computations, polynomial multiplication and squaring operations can be implemented in hardware using Montgomery or Barrett reductions over $\mathbb{F}_{2^k}$. In our experiments, we verify circuit implementations of point addition and doubling based on the above examples.

III. Review of Previous Work

The verification problem addressed in this paper is a combinational equivalence checking (CEC) problem, where the specification (polynomial) and the implementation (circuit) are given at different levels of abstraction. To make use of contemporary gate-level CEC tools, we can transform the polynomial specification into a specification circuit ("golden model") and check its equivalence against the implementation circuit. Canonical decision diagrams (BDDs [17] and their word-level variants [18]), implication-based methods [19], And-Invert-Graph (AIG) based reductions [20] [21], circuit-SAT solvers [22], etc., are among the many techniques that can be employed for this CEC. When one circuit is synthesized from the other, this problem can be efficiently solved using AIG-based reductions (e.g. the ABC tool [23]) and circuit-SAT solvers (e.g., CSAT [22]). Equivalence checking tools, such as [24] [25] [26], etc., are also offered by CAD vendors, which have even larger capacity than academic tools. Synthesized circuits generally contain many sub-circuit equivalences which AIG and CSAT based tools can identify and exploit for verification. However, *when the circuits are functionally equivalent but structurally very dissimilar*[1], none of the contemporary techniques, including ABC and CSAT, offer a practical solution. Automatic formal verification of large *custom-designed arithmetic circuits* largely remains unsolved today. Our experiments also demonstrate the inability of AIG/ABC and Circuit-SAT solvers to solve our problems.

Graph-based canonical DAG representations of Boolean functions such as BDDs [17], OKFDDs [27], BMDs [28] and MODDs [29], etc. are ill-suited for such modulo-arithmetic applications, particularly over large finite fields. While BMDs were proposed for verification of integer multipliers, the representation is not efficient for modulo arithmetic computations. MODDs [29] were presented as a canonical DAG representation for Galois field polynomials over $\mathbb{F}_{2^k}$. However, they also suffer from the size explosion problem. As every node in the MODD may have up to k children, the composition and reduction operations are rather complicated for MODDs and make verification over large fields infeasible. The work of [30] presents a DAG representation for synthesis and verification of multi-output polynomials over finite integer rings $\mathbb{Z}_{2^k}, k > 1$. Since their canonical reduction rules employ the theory of polynomial functions over finite integer rings [31], this approach is not directly applicable over finite fields $\mathbb{F}_{2^k}$. Similarly, the work of [32] is also only applicable for verification of *integer-modulo-arithmetic* over $\mathbb{Z}_{2^k}$ at word-level/RTL, and not over Galois field circuits.

This verification problem is also very hard for SAT solvers, due to the large circuit size, and the presence of AND-XOR structures. Contemporary Satisfiability Modulo Theory (SMT) solvers employ a mixture of theories for reasoning – however, none of them employ polynomial equation solving over Galois fields (which is itself a very hard problem). Therefore, in our experiments, we have used the quantifier-free bit-vector (QF-BV) theories of SMT solvers to verify Galois field circuits. As shown in our experiments, *none of BDDs, SAT, SMT solvers, and ABC can prove design correctness beyond* 16*-bit circuits.*

The theorem-proving approach of [33] verifies Galois field arithmetic algorithms over $\mathbb{F}_{2^k}$. The authors devise a decision procedure based on variable elimination and term re-writing and demonstrate a correctness proof of a sub-block of a Reed-Solomon decoder. The employed algebraic simplification rules can be beneficial only when the sentences are independent of the irreducible polynomial. Otherwise, their approach requires decision over $\mathbb{F}_2$ which is infeasible for large circuits. The work of [34] solves similar problems as those of [33]. They make use of OKFDDs [27] to canonically represent the circuit constraints. Moreover, instead of verifying circuits over $\mathbb{F}_{2^k}$ directly, [34] verifies the circuits over its equivalent composite field $GF((2^m)^n)$, where a *non-prime* $k = m \cdot n$. Their approach has no benefit if k is prime – say, when $k = 163$ for elliptic curves. Also, the size-explosion of FDDs limits their approach to 16-bit ($\mathbb{F}_{2^{16}}$) circuits, as shown in their experiments.

The paper [35] describes the high-level modeling language *Cryptol*, and its verification tool-set, that is designed for verification of cryptographic algorithms. A Cryptol description can be further synthesized into hardware. For verification, the decision procedures employed in their tools make use of AIG-based reductions (SAT-sweeping) and SAT/SMT-solving. For applications where AIGs/SAT/SMT-techniques fail, *Cryptol* tool-set has no benefit.

Symbolic computer algebra techniques have been used for verification of integer arithmetic circuits [36] [37] and also for decision procedures over Galois fields [38]. In [36], the authors verify integer arithmetic circuits hierarchically using polynomial algebra techniques. Their approach analyzes sub-circuit components and models the implementation by way of integer equations. The functionality of the sub-circuits is verified using Gröbner basis computations. However, the paper does not address any improvements to the core Gröbner basis computational engine.

The paper [37] addresses verification of finite precision integer datapath circuits using the concepts of Gröbner bases over the ring $\mathbb{Z}_{2^k}$. They model the circuit constraints by way of arithmetic-bit-level (ABL) polynomials ($\{G\}$), and formulate the verification test as an equivalent variety subset problem. To solve this, first they derive a term order that already makes $\{G\}$ a Gröbner basis. Then they compute a

[1]For example, the golden model may be a Mastrovito multiplier, and the implementation may be a Montgomery multiplier. This is also the case when the "abstraction-gap" between the specification and the implementation is very large – *e.g.* a polynomial specification versus a circuit implementation.

normal form f of the specification g w.r.t. $\{G\}$. They test if f is a vanishing polynomial over $\mathbb{Z}_{2^k}$ [32]; if so, circuit correctness is established. In [39], the authors further show that the vanishing polynomial test can be omitted by formulating the problem directly over $Q := \mathbb{Z}_{2^k}[X]/\langle x^2 - x : x \in X\rangle$.

The work of [38] shows how to use Gröbner bases techniques to count the zeros of a polynomial ideal over Galois fields. The authors then follow-up with an approach for *quantifier elimination* over Galois fields [40]. These papers address the mathematical problem formulation (theory) and algorithmic solutions; efficiency/improvements in Gröbner basis computation and application to design verification is beyond the scope of these works.

An important set of recent papers [41] [42] [43] on the BLUEVERI tool from IBM needs special mention: The authors present a methodology and toolset to verify Galois field circuits for error correcting codes against an algorithmic spec. The implementation consists of a set of (pre-designed and verified) circuit blocks that are interconnected to form the error correcting system. The spec is given as a set of design constraints on a "check file". Their objective is to prove the equivalence of the implementation against this check file. They model the verification instance as a data-flow graph, represent each sub-circuit block with its known (word-level) polynomial over $\mathbb{F}_q$, and formulate the verification problem using the *Weak Nullstellensatz* — i.e. to check if the *variety* of the algebraic system "*spec* $\neq$ *implementation*" is empty — for which they use a Gröbner basis engine. Their main contributions are: i) a "term re-writing" to specify the algorithmic description using polynomials (ideal); and ii) integrating an AIG-style [20] Boolean solver with their word-level decision procedure, with lazy signal computations and Boolean reasoning. For final verification, the polynomial system is then given to a computer algebra tool (SINGULAR [10]) to *compute* a reduced Gröbner basis. However, improvements to the core Gröbner basis computational engine are not the subject of their work.

In contrast, *our investigations go beyond the* BLUEVERI *work* by addressing further improvements to the Gröbner basis computation.

In our own previous work [44] [45] and [46], we have verified implementations of Galois field *multiplier* circuits – particularly, Mastrovito and Montgomery implementations. In [44] [45], we use the *Weak Nullstellensatz* formulation to prove that the "specification $\neq$ implementation" (miter) is infeasible, and use a Gröbner basis engine as the decision procedure. We *heuristically* derive a variable order to represent the monomial terms. Using our heuristic, we are able to verify only up to 96-bit circuits [45]. In the presence of bugs, our heuristic is inefficient in that the Gröbner basis computation runs into memory explosion beyond 16-bit circuits.

This paper is an extended version of our conference paper [46]. In this work, we draw inspirations from [38] [37] [39] [11], and build upon their results to develop automatic verification techniques for Galois field arithmetic circuits.

IV. COMPUTER ALGEBRA PRELIMINARIES

We review basic commutative algebra concepts related to ideals, varieties, Nullstellensatz and Gröbner bases, and their application over Galois fields; the material is referred from [7] [8] and [38].

Let $\mathbb{F}$ be a field and let $\mathbb{F}[x_1, \ldots, x_d]$ be the polynomial ring over $\mathbb{F}$ with indeterminates $x_1, \ldots, x_d$. A *monomial* in variables $x_1, \cdots, x_d$ is a product of the form $X = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_d^{\alpha_d}$, where $\alpha_i \geq 0, i \in \{1, \ldots, d\}$. A *polynomial* $f \in \mathbb{F}[x_1, \ldots, x_d], f \neq 0$, is written as a finite sum of terms $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$. Here $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ are monomials. To systematically manipulate the polynomials, a *monomial ordering* $>$ is imposed such that $X_1 > X_2 > \cdots > X_t$. It is a well-ordering on the set of all monomials such that multiplication with a monomial preserves the ordering[2]. Subject to such an ordering, $lt(f) = c_1X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. Similarly, tail(f) = $c_2X_2 + \cdots + c_tX_t$.

Polynomial reduction: Let f, g be polynomials. If a non-zero term cX of f is divisible by the leading term of g, then we say that f *reduces* to r modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{cX}{lt(g)} \cdot g$. Similarly, f can be reduced (divided) w.r.t. a set of polynomials $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r, denoted $f \xrightarrow{F}_+ r$, such that no term in r is divisible by the leading term of any polynomial in F.

Ideals and varieties: An *ideal* J generated by polynomials $f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_d]$ is:

$$J = \langle f_1, \ldots, f_s\rangle = \{\sum_{i=1}^{s} h_i \cdot f_i : \ h_i \in \mathbb{F}[x_1, \ldots, x_d]\}.$$

The polynomials $f_1, \ldots, f_s$ form the basis or generators of J.

Let $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{F}^d$ be a point, and $f \in \mathbb{F}[x_1, \ldots, x_d]$ be a polynomial. We say that f *vanishes* on $\mathbf{a}$ if $f(\mathbf{a}) = 0$.

For any ideal $J = \langle f_1, \ldots, f_s\rangle \subseteq \mathbb{F}[x_1, \ldots, x_d]$, the *affine variety* of J over $\mathbb{F}$ is:

$$V(J) = \{\mathbf{a} \in \mathbb{F}^d : \forall f \in J, f(\mathbf{a}) = 0\}.$$

In other words, the variety corresponds to the set of all solutions to $f_1 = \ldots f_s = 0$.

Definition 4.1: For any subset V of $\mathbb{F}^d$, the ideal of polynomials that vanish on V, called the *vanishing ideal of* V, is defined as:

$$I(V) = I_{\mathbb{F}}(V) = \{f \in \mathbb{F}[x_1, \ldots, x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}.$$

Proposition 4.1: If a polynomial f vanishes on a variety V, then $f \in I(V)$.

A. Radicals and Nullstellensatz

Definition 4.2: Let $J \subset \mathbb{F}[x_1, \ldots, x_d]$ be an ideal. The *radical of* J is defined as $\sqrt{J} = \{f \in \mathbb{F}[x_1, \ldots, x_d] : \exists m \in \mathbb{N}, f^m \in J\}$.

When $J = \sqrt{J}$, then J is said to be a *radical ideal*. Moreover, $I(V)$ is a radical ideal. The Strong Nullstellensatz establishes the correspondence between radical ideals and varieties.

[2]Lexicographic (*lex*), degree-lexicographic (*deglex*), degree-reverse-lexicographic (*degrevlex*) are examples of monomial orderings.

Theorem 4.1: Strong Nullstellensatz [8]: Let $\mathbb{F}$ be an algebraically closed field, and let J be an ideal in $\mathbb{F}[x_1, \ldots, x_d]$. Then we have $I(V(J)) = \sqrt{J}$.

We are concerned with Galois fields, which are not algebraically closed. When a field $\mathbb{F}$ is not algebraically closed, then the above result can be suitably applied over the algebraic closure of $\mathbb{F}$.

Corollary 4.1: Let $\mathbb{F}$ be an arbitrary field and J be an ideal in $\mathbb{F}[x_1, \ldots, x_d]$. Let $\overline{\mathbb{F}}$ denote the algebraic closure of $\mathbb{F}$, and let $V_{\overline{\mathbb{F}}}(J)$ denote the variety of J over $\overline{\mathbb{F}}$. Then $I_{\mathbb{F}}(V_{\overline{\mathbb{F}}}(J)) = \sqrt{J}$.

B. Strong Nullstellensatz over Galois Fields

Nullstellensatz admits a special form over Galois fields. We state the following results of Nullstellensatz over Galois fields, proofs of which can be found in [38].

Proposition 4.2: Let $\mathbb{F}_q$ be a Galois field of q elements. For all elements $A \in \mathbb{F}_q$, we have $A^q - A = 0$. Therefore, for a polynomial $x^q - x$, we have $V(x^q - x) = \mathbb{F}_q$.

The polynomials of the form $\{x^q - x\}$ are called the *vanishing polynomials* of the field. Let $F_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$, then $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ is the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$. Below, we use the concept of sum of ideals: given ideals $I_1 = \langle f_1, \ldots, f_s \rangle$ and $I_2 = \langle g_1, \ldots, g_t \rangle$, then ideal $I_1 + I_2 = \langle f_1, \ldots, f_s, g_1, \ldots, g_t \rangle$.

Lemma 4.1: From [38]: For any ideal $J \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$, $J + J_0 = J + \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ is radical. In other words, $\sqrt{J + J_0} = J + J_0$.

The above is a very powerful result, as it implies that any ideal $J \in \mathbb{F}_q[x_1, \ldots, x_d]$ can be easily turned into a radical ideal by adding J_0, without changing the zero-set $V(J)$ over $\mathbb{F}_q$. And, based on the above, the following result can be easily deduced:

Theorem 4.2: Strong Nullstellensatz over $\mathbb{F}_q$: For any Galois field $\mathbb{F}_q$, let $J \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal, and let $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ be the ideal of all vanishing polynomials. Let $V_{\mathbb{F}_q}(J)$ denote the variety of J over $\mathbb{F}_q$. Then, $I(V_{\mathbb{F}_q}(J)) = J + J_0 = J + \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$.

Proof: Let $\overline{\mathbb{F}_q}$ denote the algebraic closure of $\mathbb{F}_q$. Therefore, $\overline{\mathbb{F}_q} \supset \mathbb{F}_q$, and we have:

$$\begin{aligned} V_{\mathbb{F}_q}(J) &= V_{\overline{\mathbb{F}_q}}(J) \cap \mathbb{F}_q^d \\ &= V_{\overline{\mathbb{F}_q}}(J) \cap V_{\mathbb{F}_q}(J_0) \\ &= V_{\overline{\mathbb{F}_q}}(J) \cap V_{\overline{\mathbb{F}_q}}(J_0) \\ &= V_{\overline{\mathbb{F}_q}}(J + J_0) \end{aligned}$$

Therefore, $I(V_{\mathbb{F}_q}(J)) = I(V_{\overline{\mathbb{F}_q}}(J + J_0)) = \sqrt{J + J_0}$, from Corollary 4.1. Moreover, Lemma 4.1 says that $(J + J_0)$ is radical, so $\sqrt{J + J_0} = J + J_0$. Consequently, we have that $I(V_{\mathbb{F}_q}(J)) = J + J_0$. ∎

C. Gröbner Basis of Ideals

An ideal J may have many different generators: it is possible to have sets of polynomials $F = \{f_1, \ldots, f_s\}$ and $G = \{g_1, \ldots, g_t\}$ such that $J = \langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$ and $V(J) = V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$. Some generating sets are "better" than others, i.e. they are a better representation of the ideal. A *Gröbner basis* is one such representation which allows to solve many polynomial decision questions.

Definition 4.3: [**Gröbner Basis**] [From [8]] For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner basis for $J \iff \forall f \in J$, $f \neq 0$, there exists $i \in \{1, \cdots, t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f)$.

In our context, Gröbner bases theory provides a *decision procedure to test for membership in an ideal*. As a consequence of Definition 4.3, the set G is a Gröbner basis of ideal J, if and only if for all $f \in J$, dividing f by polynomials of G gives 0 remainder: $G = GB(J) \iff \forall f \in J, f \xrightarrow{G}_+ 0$.

Buchberger's algorithm [9], shown in Algorithm 2, computes a Gröbner basis over a field. Given polynomials $F = \{f_1, \ldots, f_s\}$, the algorithm computes the Gröbner basis $G = \{g_1, \ldots, g_t\}$. In the algorithm,

$$Spoly(f, g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g$$

where $L = \text{LCM}(lm(f), lm(g))$, where $lm(f)$ is the leading monomial of f, and $lt(f)$ is the leading term of f.

ALGORITHM 2: Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$
Output: $G = \{g_1, \ldots, g_t\}$
$G := F$;
repeat
  $G' := G$;
  for *each pair* $\{f, g\}, f \neq g$ *in* G' **do**
    $Spoly(f, g) \xrightarrow{G'}_+ r$;
    if $r \neq 0$ **then**
      $G := G \cup \{r\}$;
    end
  end
until $G = G'$;

We now describe our verification problem formulation using Strong Nullstellensatz over $\mathbb{F}_{2^k}$, and its solution using Gröbner bases and Buchberger's algorithm.

V. Verification Problem Formulation

We are given a Galois field $\mathbb{F}_q$, with $q = 2^k, k > 1$, along with the irreducible polynomial $P(x)$. Let α be the root of $P(x)$, i.e. $P(\alpha) = 0$. The specification is given as $Z = \mathcal{F}(A^1, A^2, \ldots, A^n)$, where each A^i represents a word-level input, Z is the word-level output; $Z, A^1, A^2, \ldots, A^n \in \mathbb{F}_q$, and $\mathcal{F}$ is the polynomial function describing the input-output relation. This specification can be modeled as a multivariate polynomial $f : Z - \mathcal{F}(A^1, A^2, \ldots, A^n)$; or equivalently as $f : Z + \mathcal{F}(A^1, A^2, \ldots, A^n)$, as $-1 = +1$ over $\mathbb{F}_{2^k}$.

Also given is a gate-level combinational circuit C. The bit-level primary inputs of the circuit are $\{a_0^j, a_1^j, \ldots, a_{k-1}^j\}$, for $j = 1, \ldots, n$, and the primary outputs are $\{z_0, \ldots, z_{k-1}\}$. Here $a_i^j, z_i \in \mathbb{F}_2, i = 0, \ldots, k-1,\ j = 1, \ldots, n$.

The word-level and bit-level correspondences are the following:

$$\begin{aligned} A^1 &= a_0^1 + a_1^1\alpha + \cdots + a_{k-1}^1\alpha^{k-1} \\ &\vdots \\ A^n &= a_0^n + a_1^n\alpha + \cdots + a_{k-1}^n\alpha^{k-1} \\ Z &= z_0 + z_1\alpha + z_2\alpha^2 + \cdots + z_{k-1}\alpha^{k-1} \end{aligned} \tag{1}$$

Our goal is to formally prove that $\forall A^j \in \mathbb{F}_q$, the circuit correctly implements the specification $f : Z + \mathcal{F}(A^1, A^2, \ldots, A^n) = 0$ over $\mathbb{F}_q$. Otherwise, we have to produce a counter-example that excites the bug in the design.

Problem Modeling: We analyze the given circuit C and model all the Boolean gate-level operators as polynomials over $\mathbb{F}_2$ ($\subset \mathbb{F}_{2^k}$), using the following one-to-one mapping over $\mathbb{B} \rightarrow \mathbb{F}_2$:

$$\begin{aligned} \neg a &\rightarrow a + 1 \pmod 2 \\ a \vee b &\rightarrow a + b + a \cdot b \pmod 2 \\ a \wedge b &\rightarrow a \cdot b \pmod 2 \\ a \oplus b &\rightarrow a + b \pmod 2 \end{aligned} \tag{2}$$

where $a, b \in \mathbb{F}_2 = \{0, 1\}$. To this set of Boolean polynomials extracted from the circuit, we append the polynomials corresponding to Eqns. (1) that relate the bit-level and word-level variables. We model these circuit constraints as the set of polynomials $F = \{f_1, \ldots, f_s\}$ over the ring $\mathbb{F}_q[x_1, \ldots, x_d]$, as $f_1, \ldots, f_s$ have coefficients in $\mathbb{F}_q$. We denote the ideal generated by these polynomials as $J = \langle f_1, \ldots, f_s \rangle$. Similarly, the specification polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_d]$.

Example 5.1: As an example, we describe the problem modeling for a 2-bit multiplier over $\mathbb{F}_{2^2}$. The multiplier specification is $Z = A \cdot B$, where $Z, A, B \in \mathbb{F}_{2^2}$, and $P(x) = x^2+x+1$, s.t. $P(\alpha) = 0$. The specification is modeled as a polynomial $f : Z + A \cdot B$. The circuit C is given in Fig. 2. Variables a_0, a_1, b_0, b_1 are primary inputs, z_0, z_1 are primary outputs, and c_0, c_1, c_2, c_3, r_0 are intermediate variables. The gate $\otimes$ corresponds to AND-gate, i.e. bit-level multiplication modulo 2. The gate $\oplus$ corresponds to XOR-gate, i.e. addition modulo 2.

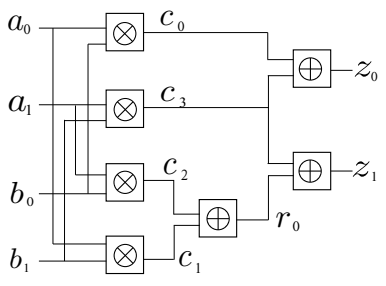

Fig. 2: A 2-bit multiplier over $\mathbb{F}(2^2)$.

Every gate in the circuit is modeled as a polynomial in $\mathbb{F}_2$. For example, the AND gate $c_0 = a_0 \wedge b_0$ is modeled as $f_1 : c_0 + a_0 \cdot b_0$. Similarly, $f_2 : c_1 + a_0 \cdot b_1$, $f_3 : c_2 + a_1 \cdot b_0$, $f_4 : c_3 + a_1 \cdot b_1$, $f_5 : r_0 + c_1 + c_2$, $f_6 : z_0 + c_0 + c_3$, $f_7 : z_1 + r_0 + c_3$. To this set, we add $f_8 : A + a_0 + a_1\alpha$, $f_9 : B + b_0 + b_1\alpha$, $f_{10} : Z + z_0 + z_1\alpha$. Then ideal $J = \langle f_1, \ldots, f_{10} \rangle$.

To prove that the specification (f) matches the implementation (J) at all points in the design space, we need to check *whether or not f vanishes on the variety $V_{\mathbb{F}_q}(J)$*. This is because for all point $p \in V_{\mathbb{F}_q}(J)$, if $f(p) = 0$, then $Z + \mathcal{F}(A^1, A^2, \ldots, A^n) = 0$ implies that $Z = \mathcal{F}(A^1, A^2, \ldots, A^n)$. On the other hand, if $f(p) \neq 0$ for some point p, then p corresponds to the bug in the design. Now if f vanishes on $V_{\mathbb{F}_q}(J)$, we know from Proposition 4.1 that f should be a member of $I(V_{\mathbb{F}_q}(J))$. The Strong Nullstellensatz over $\mathbb{F}_q$ (Theorem 4.2) tells us that $I(V_{\mathbb{F}_q}(J)) = J + J_0 = \langle f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_d^q - x_d \rangle$. Therefore, *we need to test whether or not f is a member of the ideal $J + J_0$*. If $f \in (J + J_0)$, correctness of the circuit is established. Otherwise, there is a bug in the design. To test if $f \in (J+J_0)$, it is required to compute a Gröbner basis of $(J+J_0)$, for which we can use Buchberger's algorithm.

We now have a complete approach to solve our problem: i) derive the set of polynomials $F = \{f_1, \ldots, f_s\}$ corresponding to the circuit instance; ii) append vanishing polynomials for all variables in our system $F_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$; iii) compute a Gröbner basis G of $\{F, F_0\}$ using Buchberger's algorithm; iv) reduce the specification polynomial f w.r.t. G. If $f \xrightarrow{G}_+ 0$, then the circuit is correct; otherwise there is definitely a bug in the design.

Gröbner basis Complexity: For our specific problem of computing a Gröbner basis for $J + J_0$ over $\mathbb{F}_q$, the following result is known [38]:

Theorem 5.1: Let $J = \langle f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_d^q - x_d \rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal. The time and space complexity of Buchberger's algorithm to compute a Gröbner basis of J is bounded by $q^{O(d)}$ assuming that the length of input $f_1, \ldots, f_s$ is dominated by $q^{O(d)}$.

In our case $q = 2^k$, and when k and d (all the variables in our system) are large, this complexity may make verification infeasible. In the next section, we show that a term order can be derived, by analyzing the topology of the given circuit, that makes $\{f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_d^q - x_d\}$ itself a Gröbner basis — obviating the need to apply Buchberger's algorithm.

VI. Obviating the need for Buchberger's Algorithm

To improve Buchberger's algorithm, variations of the chain and product criteria are applied.

Lemma 6.1: [Product Criterion [47]] Let $f, g \in \mathbb{F}[x_1, \cdots, x_d]$ be polynomials. If the equality $lm(f) \cdot lm(g) = LCM(lm(f), lm(g))$ holds, then $Spoly(f, g) \xrightarrow{G}_+ 0$.

The above result states that when the leading monomials of f, g are relatively prime, then $Spoly(f, g)$ always reduces to 0 modulo G. Thus $Spoly(f, g)$ need not be considered in Buchberger's algorithm. If we could analyze the given circuit and derive a term order such that every polynomial pair (f, g) in the generating set has relatively prime leading monomials, then $Spoly(f, g) \xrightarrow{G}_+ 0$. Consequently, the set of polynomials $\{f_1, \ldots, f_s\}$ extracted from the circuit

(corresponding ideal J) and represented using such a term order would itself constitute a Gröbner basis of J. In [37], the authors derive exactly such a term order, and the similar concept can be applied in our case.

Note that: i) since the circuit constraints $\{f_1, \ldots, f_s\}$ are modeled as polynomials in $\mathbb{F}_2$, they contain only multi-linear monomial terms; ii) the output of a gate is uniquely computed, and it always appears as a "single variable term" in the polynomials; iii) the circuit is acyclic. Let x_i be the output variable of any gate H_i in the circuit, and let $x_{p1}, \ldots, x_{pj}$ denote variables that are the inputs to the gate H_i. If we can represent the polynomials f_i such that $x_i >$ every monomial in the variables $x_{p1}, \ldots, x_{pj}$, then all $(f_i, f_j), i \neq j$ have relatively prime leading monomials and $\{f_1, \ldots, f_s\}$ is a Gröbner basis (this concept was described in Proposition 2 [37], which we rephrase below).

Proposition 6.1: Let C be any arbitrary combinational circuit. Let $\{x_1, \ldots, x_d\}$ denote the set of all variables (signals) in the circuit, i.e. the primary input, intermediate and primary output variables. Perform a *reverse topological traversal* of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lex term order to represent each gate as a polynomial f_i, s.t. $f_i = x_i + \text{tail}(f_i)$. Then the set of all polynomials $\{f_1, \ldots, f_s\}$ forms a Gröbner basis, as $lt(f_i)$ and $lt(f_j)$ for $i \neq j$ are relatively prime.

Example 6.1: Consider, again, the 2-bit multiplier over $\mathbb{F}_{2^2}$ shown in Fig. 2. Variables a_0, a_1, b_0, b_1 are primary inputs, z_0, z_1 are primary outputs, and c_0, c_1, c_2, c_3, r_0 are intermediate variables.

Perform a "reverse topological traversal" of the circuit. Starting from the primary outputs, traverse the circuit to the primary inputs, and order the gates according to the their (reverse) topological levels. The primary outputs z_0, z_1 are both at level-0, variables r_0, c_0, c_3 are at level-1, c_1, c_2 are at level-2, and the primary inputs a_0, a_1, b_0, b_1 are at level-3. We order the variables $\{z_0 > z_1\} > \{r_0 > c_0 > c_3\} > \{c_1 > c_2\} > \{a_0 > a_1 > b_0 > b_1\}$. Using this variable order, we impose a lex term order on the monomials. Then all the circuit polynomials have relatively prime leading terms, as shown below:

$$f_1 : c_0 + a_0 \cdot b_0,\ lm = c_0; \quad f_2 : c_1 + a_0 \cdot b_1,\ lm = c_1$$
$$f_3 : c_2 + a_1 \cdot b_0,\ lm = c_2; \quad f_4 : c_3 + a_1 \cdot b_1,\ lm = c_3$$
$$f_5 : r_0 + c_1 + c_2,\ lm = r_0; \quad f_6 : z_0 + c_0 + c_3,\ lm = z_0$$
$$f_7 : z_1 + r_0 + c_3,\ lm = z_1$$

In our formulation, we also have word-level variables $Z, A^1 \ldots, A^n \in \mathbb{F}_q$. They can also be accommodated in this term order by imposing $\{Z\} > \{A^1 > \cdots > A^n\} > \{z_0 > z_1\} > \{r_0 > c_0 > c_3\} > \{c_1 > c_2\} > \{a_0 > a_1 > b_0 > b_1\}$.

To perform the reverse topological traversal, the circuit is considered as a graph, where the gates are represented by nodes and the wires represented by edges. Topological sort of the graph is then performed from the primary outputs to the primary inputs to derive the variable order.

As a result of Proposition 6.1, the set of polynomials $F = \{f_1, \ldots, f_s\}$ is a Gröbner basis for J. Note that $F_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$ is a Gröbner basis for J_0. However, we have to compute a Gröbner basis of $J + J_0 = \langle F, F_0 \rangle = \langle f_1, \ldots, f_s,\ x_1^q - x_1, \ldots, x_d^q - x_d \rangle$. Not all polynomial pairs in $\{f_1, \ldots, f_s,\ x_1^q - x_1, \ldots, x_d^q - x_d\}$ have relatively prime leading monomials.

Consider a polynomial $f_i \in F$. Using our term order, we have $f_i = x_i + \text{tail}(f_i)$; i.e. the leading monomial of f_i is a single variable term x_i, corresponding to the output of a gate. Clearly, the pair $(x_i + \text{tail}(f_i),\ x_i^q - x_i),\ f_i \in F,\ x_i^q - x_i \in F_0$ do not have relatively prime leading monomials. In fact, the pairs $(x_i + \text{tail}(f_i), x_i^q - x_i)$ are the only ones to be considered for Gröbner basis computation, as all other pairs have relatively prime leading terms. This motivated us to investigate further the result of the reduction: $Spoly(x_i + \text{tail}(f_i), x_i^q - x_i) \xrightarrow{F, F_0}_+ r$. We state and prove the following:

Theorem 6.1: Let $q = 2^k$, and let $\mathbb{F}_q[x_1, \ldots, x_d]$ be a ring on which we have a monomial order $>$. Let I be a subset of $\{1, \ldots, d\}$. For all $i \in I$, let $f_i = x_i + P_i$ (where $P_i = \text{tail}(f_i)$) such that all indeterminates x_j that appear in P_i satisfy $x_i > x_j$. Then the set $G = \{f_i : i \in I\} \cup \{x_1^q - x_1, \ldots, x_d^q - x_d\}$ is a Gröbner basis.

Proof: According to Buchberger's Theorem (Theorem 1.7.4 in [8]), we need to show that for all $f, g \in G$, $Spoly(f, g) \xrightarrow{G}_+ 0$. Let $F = \{f_1, \ldots, f_s\}, F_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$, and $G = F \cup F_0$. Lemma 6.1 shows that if $f, g \in G$, have relatively prime leading terms, then $Spoly(f, g) \xrightarrow{G}_+ 0$. So the only case where Lemma 6.1 does not apply is when $f = x_i + P_i$ and $g = x_i^q - x_i$. Then $Spoly(f, g) = x_i^{q-1} f - g = P_i x_i^{q-1} + x_i$. In what follows, it is important to note that the indeterminates appearing in P_i are all less than x_i.

First of all, $P_i x_i^{q-1} + x_i - P_i x_i^{q-2}(x_i + P_i) = P_i^2 x_i^{q-2} + x_i$, which shows that $P_i x_i^{q-1} + x_i \xrightarrow{x_i + P_i} P_i^2 x_i^{q-2} + x_i$.

Next, $P_i^2 x_i^{q-2} + x_i - P_i^2 x_i^{q-3}(x_i + P_i) = P_i^3 x_i^{q-3} + x_i$. Continuing in this fashion, we get $P_i^{q-1} x_i + x_i - P_i^{q-1}(x_i + P_i) = x_i + P_i^q$, and finally $x_i + P_i^q - (x_i + P_i) = P_i^q - P_i$. Hence,

$$P_i x_i^{q-1} + x_i \xrightarrow{x_i + P_i} P_i^2 x_i^{q-2} + x_i \xrightarrow{x_i + P_i} P_i^3 x_i^{q-3} + x_i \xrightarrow{x_i + P_i} \cdots$$
$$\cdots \xrightarrow{x_i + P_i} P_i^q + x_i \xrightarrow{x_i + P_i} P_i^q - P_i.$$

In other words, $Spoly(x_i + \text{tail}(f_i), x_i^q - x_i) \xrightarrow{F}_+ P_i^q - P_i$. Over the Galois field $\mathbb{F}_q$, $P_i^q - P_i$ is a vanishing polynomial. Therefore, $P_i^q - P_i \in I(V(J_0)) = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$. By Lemma 6.1, $F_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$ is Gröbner basis. Therefore $P_i^q - P_i \xrightarrow{F_0}_+ 0$ which gives that $P_i^q - P_i \xrightarrow{G}_+ 0$, as $F_0 \subset G$. In conclusion, $\forall f, g \in G,\ Spoly(f, g) \xrightarrow{G}_+ 0$ and hence G is a Gröbner basis. ■

This Gröbner basis $G = F \cup F_0$ can be further simplified to a *minimal form*.

Definition 6.1: [From [8]] A Gröbner basis $G = \{g_1, \ldots g_t\}$ is called minimal if: (i) $\forall i$, $lc(g_i) = 1$; and (ii) $\forall i \neq j, lm(g_i)$ does not divide $lm(g_j)$.

In other words, for a Gröbner basis to be minimal, two conditions have to be satisfied: i) all polynomials in the basis are monic, i.e their leading coefficient is 1; and ii) the leading

monomial of any polynomial does not divide the leading monomial of any other polynomial in the basis.

Lemma 6.2: [From [8]] Let $G = \{g_1, \ldots, g_t\}$ be a Gröbner basis of ideal J. If $lm(g_2)$ divides $lm(g_1)$, then $\{g_2, \ldots, g_t\}$ is also a Gröbner basis of J.

Suppose $G = \{g_1, \ldots, g_t\}$ is a Gröbner basis such that $lm(g_2)$ divides $lm(g_1)$. If any polynomial f is such that $lm(f)$ is divisible by $lm(g_1)$, then $lm(f)$ is also divisible by $lm(g_2)$. Therefore, g_1 is a redundant element of the Gröbner basis and can be eliminated from G. Moreover, each remaining $g_i \in G$ can be made monic by dividing g_i by $lc(g_i)$. When all such redundant elements are removed, a minimal Gröbner basis is obtained [8]. Based on Definition 6.1, we can generate a minimal Gröbner basis G of $J + J_0$ directly by construction, as given below.

Corollary 6.1: Given the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_d], q = 2^k$, on which we have the monomial order $>$ specified in Proposition 6.1. Given the set of polynomials $F = \{f_1, \ldots, f_s\} \in \mathbb{F}_q[x_1, \ldots, x_d]$, representing the circuit C. Let $X_{PI} \subset \{x_1, \ldots, x_d\}$ denote the set of all primary input variables of the circuit. Let $F_0^{PI} = \{x_i^2 - x_i : x_i \in X_{PI}\}$, be the set of (bit-level) vanishing polynomials corresponding to all primary input variables. Then the set $G = F \cup F_0^{PI} = \{f_1, \ldots, f_s\} \cup \{x_i^2 - x_i : x_i \in X_{PI}\}$ is a minimal Gröbner basis.

Proof: We have already shown in Theorem 6.1 that $G = F \cup F_0$ is a Gröbner basis. Moreover, in our problem, all polynomials in G are monic. Furthermore, our ideal basis G consists of two sets of polynomials: i) polynomials of the form $f_i = x_i + \text{tail}(f_i), f_i \in F$; and ii) the vanishing polynomials $x_i^{2^k} - x_i \in F_0$ for $i = 1, \ldots, d$. Our monomial order ensures that in $f_i = x_i + \text{tail}(f_i)$, x_i corresponds to either a primary output variable or an intermediate variable of the circuit. Bit-level primary inputs of the circuit ($x_i \in X_{PI}$) never occur as leading terms of f_i because a primary input is not an output of any gate in the circuit. Therefore, $\forall x_i \in (\{x_1, \ldots, x_d\} - X_{PI})$, there always exists f_i with $lm(f_i) = x_i$ which will divide the leading monomial of the vanishing polynomial $x_i^{2^k} - x_i$. In such cases, $x_i^{2^k} - x_i,\ x_i \notin X_{PI}$ can be removed from the basis, due to Lemma 6.2. By eliminating all such vanishing polynomials corresponding to the non-primary-input variables, we will obtain $G = \{f_1, \ldots, f_s\} \cup \{x_i^{2^k} - x_i : x_i \in X_{PI}\}$.

Finally, since bit-level variable $x_i \in \mathbb{F}_2 \subset \mathbb{F}_{2^k}$, $x_i^2 - x_i = 0$, we obtain $G = \{f_1, \ldots, f_s\} \cup \{x_i^2 - x_i : x_i \in X_{PI}\}$ as the minimal Gröbner basis. ∎

While we can obtain a minimal Gröbner basis G directly by construction, unfortunately, we *cannot* obtain a *reduced* Gröbner basis without actually performing the reduction. This is because in a reduced Gröbner basis, tail(f_i) is also reduced w.r.t. $lt(f_j)$, for all $i \neq j$. However, a reduced Gröbner basis is not necessary for ideal membership testing.

A. Our Overall Approach

The verification problem is setup in $\mathbb{F}_{2^k}[x_1, \ldots, x_d]$, on which we impose the monomial order $>$ as derived above. The set of polynomials $F = \{f_1, \ldots, f_s\}$ is derived/extracted from the circuit. The set $F_0^{PI} = \{x_i^2 - x_i : x_i \in X_{PI}\}$ is generated as the set of all bit-level vanishing polynomials corresponding to the primary inputs. Then the set $G = F \cup F_0^{PI}$ forms a minimal Gröbner basis of the ideal $J + J_0 = \langle f_1, \ldots, f_s, x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$. We reduce the specification polynomial f w.r.t. G: $f \xrightarrow{G}_+ r$. If $r = 0$, then $f \in J + J_0$ and the circuit is correct. Otherwise, if $r \neq 0$, there is a bug in the design. Moreover, *if $r \neq 0$, then the monomial order ensures that r contains only the primary input variables.* To show this, assume that $r \neq 0$ and r contains either an intermediate or a primary output variable x_j. Since there always exists a polynomial f_j in G with $lm(f_j) = x_j$, r can be further reduced by f_j. Continuing in this fashion, all the terms with non-primary-input (intermediate or primary output) variables can be eliminated, and r contains only primary inputs. Finally, *in the presence of a bug, any assignment to the (primary-input) variables that makes $r \neq 0$, provides a counter-example for debugging*. A SAT-solver can find such an assignment in no time as r is simplified by Gröbner basis reduction. Our results therefore obviate the need to *compute* a Gröbner basis using Buchberger's algorithm. The set G is already a Gröbner basis, and verification can be performed by the reduction: $f \xrightarrow{G}_+ r$.

VII. Improving Polynomial Division using $F4$-style Reduction

The most intensive computational step in our approach is that of polynomial division $f \xrightarrow{F, F_0^{PI}}_+ r$. When the circuit C is very large, the polynomial set $\{F, F_0^{PI}\}$ also becomes extremely large. This division procedure then becomes the bottleneck in verifying the equivalence. In principle, this reduction can be performed using contemporary computer-algebra systems — *e.g.*, the SINGULAR [10] tool, which is widely used within the verification community [37] [39] [46]. In our work, we have also performed experiments with SINGULAR. However, as in any "general-purpose" computer algebra tool, the data-structures are not specifically optimized for circuit verification problems. Moreover, SINGULAR also limits the number of variables (d) that it can accommodate in the system to $d < 32767$; this limits its application to large circuits. Therefore, to further improve our approach, we exploit the relatively recent concept of $F4$-style polynomial reduction [11] — which implements polynomial division using row-reductions on a matrix — to develop a custom verification tool to perform this Gröbner basis reduction efficiently.

$Faugère$'s $F4$ approach [11] presents a new algorithm to compute a Gröbner basis. It uses the same mathematical principles as Buchberger's algorithm. However, instead of computing and reducing one S-polynomial at a time, it computes many S-polynomials in one step and reduces them simultaneously using sparse linear algebra on a matrix (triangulation). In our formulation, since we already have a Gröbner basis, no S-polynomials are computed. We only need to perform the subsequent Gröbner basis reduction, for which the $F4$ technique can be very efficient. We now show how our reduction problem $f \xrightarrow{F, F_0^{PI}}_+ r$ can be represented and solved on a matrix. First, let us consider the following example that

demonstrates the main concepts behind the reduction approach of $F4$.

Example 7.1: Consider the lex term order with $x > y > z$ on the ring $\mathbb{Q}[x, y, z]$. Given $F = \{f_1 = 2x^2 + y, f_2 = 3xy^2 - xy, f_3 = 4y^3 - 1\}$, consider one step of Buchberger's algorithm: $S(f_1, f_2) \xrightarrow{f_1,f_2,f_3}_+ r$. We have, $Spoly(f_1, f_2) = \frac{1}{3}x^2y + \frac{1}{2}y^3 = f_4$. The reduction $Spoly(f_1, f_2) \xrightarrow{f_1,f_2,f_3}_+ (-\frac{1}{6}y^2 + \frac{1}{8})$ is done as follows: Since $lt(f_1) \mid lt(f_4)$, $f_4 \xrightarrow{f_1} h$ is computed as:

$$h = f_4 - \frac{lt(f_4)}{lt(f_1)}f_1 = f_4 - \frac{1}{6}yf_1 = \frac{1}{2}y^3 - \frac{1}{6}y^2;$$

Now $lt(f_2)$ does not divide any term in h, but $lt(f_3) \mid lt(h)$, so $h \xrightarrow{f_3} r$:

$$r = h - \frac{lt(h)}{lt(f_3)}f_3 = \frac{1}{2}y^3 - \frac{1}{6}y^2 - \frac{1}{8}f_3 = -\frac{1}{6}y^2 + \frac{1}{8}$$

This reduction procedure can also be simulated on a matrix using Gaussian elimination. The reduction above requires the computation of $\frac{1}{6}yf_1$ and $\frac{1}{8}f_3$. Ignoring the coefficients $\frac{1}{6}, \frac{1}{8}$, we can generate all the monomials required in the reduction process: i.e. monomials of f_4, yf_1, f_3, and setup the problem of cancellation of terms as Gaussian elimination on a matrix. Monomials of f_4, yf_1, f_3 are, respectively, $\{x^2y, y^3\}, \{x^2y, y^2\}, \{y^3, 1\}$. Let the rows of a matrix M correspond to polynomials $[f_4, yf_1, f_3]$, and columns correspond to all the monomials (in lex order) $\left[x^2y, y^3, y^2, 1\right]$. Then the matrix M shows the representation of these polynomials where the entry $M(i, j)$ is the coefficient of monomial of column j present in the polynomial of row i.

$$M = \begin{array}{c} \\ f_4 \\ yf_1 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccc} x^2y & y^3 & y^2 & 1 \end{array} \\ \begin{pmatrix} \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 4 & 0 & -1 \end{pmatrix} \end{array}$$

Now, reducing M to a row echelon form using Gaussian elimination gives:

$$M = \begin{array}{c} \\ f_4 \\ h = f_4 - \frac{1}{6}yf_1 \\ r = h - \frac{1}{8}f_3 \end{array} \begin{array}{c} \begin{array}{cccc} x^2y & y^3 & y^2 & 1 \end{array} \\ \begin{pmatrix} \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & -\frac{1}{6} & 0 \\ 0 & 0 & -\frac{1}{6} & \frac{1}{8} \end{pmatrix} \end{array}$$

The last row $(0, 0, -\frac{1}{6}, \frac{1}{8})$ accounts for polynomial $-\frac{1}{6}y^2 + \frac{1}{8}y$ which is equal to the reduction result r obtained before.

This approach generates all the monomial terms that are required in the division process, and the coefficients required for cancellation of terms are accounted for by elementary row reductions in the subsequent Gaussian elimination. Based on the above concepts, a matrix can be constructed for our problem: $f \xrightarrow{F,F_0^{PI}}_+ r$.

Definition 7.1: Let $L = [f_1, \ldots, f_m]$ be a list of m polynomials. Let M_L be an ordered list of monomials of elements of L and let n be the number of elements in M_L. Define M as the $m \times n$ matrix which associates the polynomials of L to rows and monomials of M_L to columns. Entry in row i, column j is the coefficient of the j^{th} element of M_L in f_i.

ALGORITHM 3: Generating the Matrix for Polynomial Reduction

Input: $f, F = \{f_1, \ldots, f_s\}$, term order $>$
Output: A matrix M representing $f \xrightarrow{f_1,\ldots,f_s}_+ r$
/*$L =$ set of polynomials, rows of M*/;
L:={f} ;
i:=1;
/*$M_L =$ the set of monomials, columns of M */;
M_L:={ monomials of f} ;
mon:= the i^{th} monomial of M_L;
while *mon* $\notin PrimaryInputs$ **do**
  Identify $f_k \in F$ satisfying: $lm(f_k)$ can divide mon ;
  /*add polynomial f_k to L as a new row in M */;
  $L := L \cup \frac{mon}{lm(f_k)} \cdot f_k$;
  /*Add monomials to M_L as new columns in M */;
  M_L:=$M_L \cup$ {monomials of $\frac{mon}{lm(f_k)} \cdot f_k$} ;
  $i := i + 1$;
  mon:= the i^{th} monomial of M_L;
end
Gaussian Elimination on M;
return $r =$ last row of M;

Algorithm 3 describes our procedure to generate the matrix M of polynomials corresponding to our verification instance. The main idea is to setup the rows and columns of the matrix in a way that polynomial division can be subsequently performed by applying Gaussian elimination on M. In the algorithm, the set of polynomials $F = \{f_1, \ldots, f_s\}$ correspond to the circuit constraints. The term ordering derived from the topological analysis of the circuit is imposed to represent the polynomials. The specification polynomial f is to be reduced w.r.t. $F = \{f_1, \ldots, f_s\}$. Initially, $L = \{f\}$ is inserted as the first row of the matrix and M_L constitutes the (ordered) list of monomials of f. Then, in every iteration i, a polynomial $f_k \in F$ is identified such that $lm(f_k)$ divides the i^{th} monomial (mon) of M_L; this is to enable cancellation of the corresponding monomial term. The computation $L := L \cup \frac{mon}{lm(f_k)} \cdot f_k$ in the while-loop, generates the polynomials required for reduction[3]. The list M_L is updated to include monomials of $\frac{mon}{lm(f_k)} \cdot f_k$. Finally, the iteration in the loop terminates when monomial mon consists solely of primary input variables. This is because primary inputs are never a leading term of any polynomial; therefore, no polynomial $f_k \in F$ exists which can divide mon. Moreover, due to our term order, once mon consists of only primary inputs, all remaining monomials will also contain only primary input variables. Clearly, no more polynomials f_k need to be generated in L, and the loop terminates.

Using the set L as rows and M_L as columns, a matrix M is constructed and Gaussian elimination is applied to reduce it to row-echelon form. The last row in the reduced matrix corresponds to the reduction result r. Let us describe the approach using an example.

Example 7.2: Consider the reduction related to verification of the $\mathbb{F}_{2^2}$ multiplier circuit of Fig. 2. Given specification f :

[3]Recall that the division $\frac{f_i}{f_k} = f_i - \frac{lt(f_i)}{lt(f_k)} \cdot f_k = f_i - \frac{lc(f_i)}{lc(f_k)} \cdot \frac{lm(f_i)}{lm(f_k)} \cdot f_k$. In the algorithm, the computation $\frac{mon}{lm(f_k)} \cdot f_k$ corresponds to $\frac{lm(f_i)}{lm(f_k)} \cdot f_k$ used in the division.

$$M = \begin{array}{c} \\ f \\ f_3 \\ Bf_1 \\ a_0f_2 \\ a_1f_2 \\ f_5 \\ f_6 \\ f_4 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} Z & AB & Ba_0 & Ba_1 & z_0 & z_1 & r_0 & a_0b_0 & a_0b_1 & a_1b_0 & a_1b_1 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \alpha & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \alpha \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

(a) Matrix M generated by Algorithm 3

$$M = \begin{array}{l} \\ row_1 = f \\ row_2 = f_3 - row_1 \\ row_3 = Bf_1 - row_2 \\ row_4 = a_0f_2 - row_3 \\ row_5 = \alpha a_1f_2 - row_4 \\ row_6 = f_5 - row_5 \\ row_7 = \alpha f_6 - row_6 \\ r = \alpha f_4 - row_7 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} Z & AB & Ba_0 & Ba_1 & z_0 & z_1 & r_0 & a_0b_0 & a_0b_1 & a_1b_0 & a_1b_1 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \alpha & 1 & \alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha & 1 & \alpha & 0 & 1 & \alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \alpha & 0 & 1 & \alpha & \alpha & \alpha^2 \\ 0 & 0 & 0 & 0 & 0 & \alpha & 0 & 0 & \alpha & \alpha & \alpha^2+1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \alpha & 0 & \alpha & \alpha & \alpha^2+\alpha+1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2+\alpha+1 \end{pmatrix} \end{array}$$

(b) M reduced to row echelon form via Gaussian Elimination

Fig. 3: F_4-style Polynomial reduction on a matrix for Example 7.2.

$Z + AB$, and the circuit polynomials $f_1 : A + a_0 + a_1\alpha,\ f_2 : B + b_0 + b_1\alpha,\ f_3 : Z + z_0 + z_1\alpha,\ f_4 : r_0 + a_0b_1 + a_1b_0,\ f_5 : z_0 + a_0b_0 + a_1b_1,\ f_6 : z_1 + r_0 + a_1b_1$. Here $P(x) = x^2 + x + 1$, and $P(\alpha) = 0$. We have to compute $f \xrightarrow{f_1,\ldots,f_6}_+ r$. Note that, for simplicity, variables c_0, c_1, c_2, c_3 from Fig. 2 have been substituted by functions on primary inputs. The imposed ordering corresponds to the one obtained due to Proposition 6.1: i.e. lex with $Z > A > B > z_0 > z_1 > r_0 > a_0 > a_1 > b_0 > b_1$. The algorithm constructs the matrix as follows:

1) *Initialization: $L = \{f\} = \{Z + AB\}$. $M_L = \{Z, AB\}, i = 1, mon = Z$ (i^{th} monomial of M_L).*
2) *Iteration 1: Identify a polynomial $f_k \in F$ s.t. $lm(f_k) \mid mon$. Clearly, $f_k = f_3 = Z + z_0 + z_1\alpha$. Then, $L = L \cup \frac{mon}{lt(f_k)} \cdot f_k = L \cup f_3$. Therefore, $L = \{f, f_3\}$ and $M_L = \{Z, AB, z_0, z_1\}$, $i = 2$ and $mon = AB$.*
3) *Iteration 2: $f_k = f_1 = A + a_0 + a_1\alpha$ because $lm(f_1) \mid mon$. Therefore, $L = L \cup \frac{AB}{A} \cdot f_1 = L \cup Bf_1 = \{f, f_3, Bf_1\}$ and $M_L = \{Z, AB, Ba_0, Ba_1, z_0, z_1\}, i = 3, mon = Ba_0$.*
4) *Iteration 3: $f_k = f_2 = B + b_0 + b_1\alpha$ as $lt(f_2) \mid mon$. Therefore, $L = L \cup \frac{mon}{lt(f_k)} \cdot f_k = L \cup \frac{Ba_0}{B} \cdot f_2 = L \cup a_0f_2$. So, $L = \{f, f_3, Bf_1, a_0f_2\}$ and the monomial set $M_L = \{Z, AB, Ba_0, Ba_1, z_0, z_1, a_0b_0, a_0b_1\},\ i = 4, mon = Ba_1$.*
5) *Continuing in this fashion ...*
6) *Iteration 7: $L = \{f, f_3, Bf_1, a_0f_2, a_1f_2, f_5, f_6, f_4\},\ M_L = \{Z, AB, Ba_0, Ba_1, z_0, z_1, r_0, a_0b_0, a_0b_1, a_1b_0, a_1b_1\}, i = 8, mon = a_0b_0$.*
7) *Iteration 8: Since $mon = a_0b_0$ contains only the primary inputs, no polynomial in F has a leading term that can cancel mon, so the loop terminates. The matrix M can be constructed using L as rows and M_L as columns.*

Fig. 3a shows the matrix M, and its subsequent Gaussian elimination is shown in Fig. 3b. The last row of the reduced matrix corresponds to the reduction $f \xrightarrow{f_1,\ldots,f_s}_+ r$, where $r = (\alpha^2 + \alpha + 1) \cdot (a_1b_1)$. Note that since α is the root of the irreducible polynomial $P(x) = x^2 + x + 1$, we have that $\alpha^2 + \alpha + 1 = 0$ in $\mathbb{F}_{2^2}$, and hence $r = 0$. In conclusion, $f \xrightarrow{f_1,\ldots,f_s}_+ 0$, and the circuit correctly implements f.

Other implementation issues: Recall that in our problem, we also have to account for bit-level vanishing polynomials of primary input variables: $F_0^{PI} = \{x_i^2 - x_i : x_i \in X_{PI}\}$. In our algorithm, if we encounter a bit-level variable x_i with degree $n \geq 2$, we replace $x_i^n = x_i$. Consequently, no specific entries for vanishing polynomials in the matrix need to be created. Moreover, it is also possible to encounter coefficients α^n, where $n \geq k$ in $\mathbb{F}_{2^k}$. We precompute $\alpha^k, \ldots, \alpha^{2k-2}$ modulo the primitive polynomial $P(x)$, store them in a table, and use these reduced values as coefficients in the matrix for reduction.

As discussed by Faugére in [11], the sparse linear algebra approach of $F4$ *simulates* multivariate polynomial division on a matrix, as it requires the generation of all monomials and polynomials utilized in the division process. Therefore, the complexity of generating the matrix using Algorithm 3, in the worst case, is the same as that of multivariate polynomial division. Moreover, the arithmetic complexity of Gaussian elimination over an $n \times n$ matrix is known to be $O(n^3)$ [48].

The above approach is implemented as a standalone, custom verification tool, which inputs a Boolean gate-level circuit netlist C and a specification polynomial f. The tool performs a reverse topological traversal of the circuit and derives the term order to represent polynomials. Then, Algorithm 3 is invoked to construct the matrix M; Gaussian elimination is finally performed on M to obtain the verification result via reduction: $f \xrightarrow{F, F_0^{PI}}_+ r$. Our tool is written in C++.

VIII. Experimental Results

Using our setup, we perform formal verification of several large, custom-designed, Galois field arithmetic circuits in $\mathbb{F}_{2^k}$. These circuits include Mastrovito [15], Montgomery [2] and Barrett multipliers [3], along with ECC point addition and doubling circuits [5]. Verification experiments are conducted with both bug-free and with buggy circuits. To compare our approach against contemporary verification techniques, we also experiment with SAT, SMT, ABC [23] solvers, and the SINGULAR tool [v. 3-1-3] [10]. The circuit designs (C) are given as gate-level netlists: these are translated to different formats: CNF, SMTLIB, BLIF/EQN, and polynomials, that are used by SAT, SMT, BDD/ABC and SINGULAR, respectively. Our experiments are conducted on a desktop with 2.40GHz Intel Core $^{\text{TM}}2$ Quad CPU with 8GB memory running 64-bit Linux. *A few of these circuit benchmarks are made available to the community through the website* [49].

A. Evaluation of SAT/SMT/BDD/ABC based methods

In our problems, while the implementation is given as a circuit, the specification is given as a *word-level polynomial*. Bit-level solvers cannot readily verify a circuit C against a *polynomial*. Therefore, we convert the polynomial specification into a gate-level, *golden* circuit. Then, using conventional equivalence checking approaches, we create a *miter* with the specification and the implementation, and use SAT/SMT/BDD/ABC methods to check if it is unsatisfiable.

For SAT, BDDs and ABC, we use a, pre-designed Mastrovito-style bit-level circuit as the golden model, and verify it against the given circuit. For SMT experiments, the designs are modeled at bit-vector level using QF-BV theories, maintaining a BV-level abstraction whenever possible (the circuits are described at partial-product level using bit-vectors). Table I shows the results of verification of the Mastrovito multiplier (golden model) against a Montgomery multiplier implementation. None of BDDs, SAT, SMT and ABC solvers can verify the correctness of circuits beyond 16-bits.

TABLE I: Runtime for verification of correct multiplier circuits over $\mathbb{F}_{2^k}$ for BDDs, SAT, ABC, SMT-solver based methods. TO = timeout of 10hrs.

	Word size of the operands k-bits		
Solver	8	12	16
MiniSAT	22.55	TO	TO
CryptoMiniSAT	7.17	16082.40	TO
PrecoSAT	7.94	TO	TO
PicoSAT	14.85	TO	TO
Yices	10.48	TO	TO
Beaver	6.31	TO	TO
CVC	TO	TO	TO
Z3	85.46	TO	TO
Boolector	5.03	TO	TO
Sonolar	46.73	TO	TO
SimplifyingSTP	14.66	TO	TO
ABC	242.78	TO	TO
BDD	0.10	14.14	1899.69

The approach employed by ABC [23] for combinational equivalence checking uses an integration of AIG rewriting via structural hashing, simulation, mitering and SAT, and it is considered to be state-of-the-art. In [21], the authors have shown that using such an approach, verification between a given design and its synthesized counterpart can be efficiently performed for large circuits, even for industrial designs. The success of these techniques is attributed to being able to find a large number of internal node equivalences — which reduces the overall verification complexity. However, when not many internal node equivalences can be identified, such techniques prove to be infeasible.

The multiplier architectures that we experiment with, are structurally very dissimilar. Indeed, the approach of [23] [21] is unable to identify and merge internal node equivalencies in the miter — mostly because they do not exist. We describe the following experiments to demonstrate this effect. The specification (golden model) and the implementation circuits are given as input to the ABC tool, and the total number of nodes N_1 are counted in the miter. Then, we invoke the equivalence checking engine of ABC. Just before the "fraigswept and reduced" miter is given to the SAT solver for equivalence proof, we count the total number of nodes N_2 in the reduced miter. Then $\frac{N_1-N_2}{N_1}$ reflects the internal node equivalencies identified and merged to create the reduced miter.

In Table II, the internal node equivalencies identified between Mastrovito and Montgomery multipliers are reported for various bit-widths. For verification between Montgomery and Barrett architectures, the data is reported in Table III. It can be seen from the results that ABC is unable to find many internal node equivalences. For this reason, such techniques are inefficient for verification of our applications.

TABLE II: Node equivalences: Mastrovito versus Montgomery multipliers. N_1, N_2 are the number of nodes counted before and after structural hashing, respectively.

Size k	8	16	32	64	96	128	163
N_1	218	832	2226	7412	15576	26422	42273
N_2	198	756	2160	7232	15384	26098	41947
Similarity	9.17%	9.13%	2.96%	2.42%	1.23%	1.23%	0.77%

TABLE III: Node equivalences: Barrett versus Montgomery multipliers. N_1, N_2 are the number of nodes counted before and after structural hashing, respectively.

Size k	8	16	32	64	96	128	163
N_1	208	774	2108	7221	15293	26108	41835
N_2	196	713	2074	7105	15197	25907	41672
Similarity	4.29%	7.89%	1.62%	1.61%	0.63%	0.77%	0.39%

B. Computing a Gröbner basis using Singular

Conceptually, our approach requires the computation of a Gröbner basis, and then to conduct a polynomial reduction (ideal membership testing). Without deducing the result of Theorem 6.1 and Corollary 6.1, if we use SINGULAR to *compute* a Gröbner basis using the term order of Proposition 6.1, we can only verify the correctness up to 48-bit multipliers. Beyond that, the Gröbner basis engine encounters a memory explosion. This result is shown in Table IV.

TABLE IV: Verification of Mastrovito multipliers by computing Gröbner bases using SINGULAR. MO=out of 8G memory.

Operand Size k	16	32	48	64	96
#variables	323	1155	2499	4355	9603
#polynomials	291	1091	2403	4227	9411
#terms	1793	7169	16129	28673	64513
Time (sec)	0.94	93.80	1174.27	MO	MO

C. Evaluation of Our Approach

Our approach only requires a polynomial reduction (division) for the verification test: $f \xrightarrow{F, F_0^{PI}}_{+} r$ and to check if $r = 0$? Results for verification of Mastrovito multipliers using only this polynomial reduction are shown in Table V. We experiment with : i) SINGULAR to perform the reduction using the REDUCE command, denoted (Singular) in the table; and ii) our own $F4$-style reduction approach. We also experimented with bug-catching in incorrect designs; the bugs are introduced by arbitrarily inter-changing the wires (variables) x_i with x_j, for some $i \neq j$. In such cases, we obtained a non-zero r. We used a SAT-solver to find a SAT assignment to $r \neq 0$, and the counter-example was generated in negligible amount of time. As shown in Table V, both SINGULAR and our $F4$ approach can verify the correctness of up to 163-bit Mastrovito multipliers – corresponding to the practical NIST-specified Galois field $\mathbb{F}_{2^{163}}$. However, our $F4$-style approach is almost $2.5X$ faster.

TABLE V: Runtime for verifying bug-free and buggy Mastrovito multipliers using our approach. TO = timeout of 10hrs. Time is given in seconds.

Operand size k:	32	64	96	128	160	163
#variables	1155	4355	9603	16899	26243	27224
#polynomials	1091	4227	9411	16643	25923	26989
#terms	7169	28673	64513	114689	179201	185984
Bug-free (Singular)	1.41	112.13	758.82	3054	9361	16170
Bug-free (F_4)	0.83	39.23	243.16	1138	3496	6537
Bugs (Singular)	1.43	114.86	788.65	3061	9384	16368
Bugs (F_4)	0.84	40.01	249.84	1152	3530	6592

TABLE VI: Runtime for verifying bug-free and buggy Montgomery multipliers. TO = timeout of 10hrs. Time is given in seconds. $*$ denotes SINGULAR's capacity exceeded.

Operand size k	32	48	64	96	128	163
#variables	1194	2280	4395	6562	14122	91246
#polynomials	1130	2184	4267	6370	13866	89917
#terms	10741	18199	40021	55512	134887	484738
Bug-free (Singular)	1.50	11.03	27.70	1802.75	10919	$*$
Bug-free (F_4)	0.86	4.47	10.11	700.59	4539	18374
Bugs (Singular)	1.52	11.10	28.18	1812.15	11047	$*$
Bugs (F_4)	0.88	4.49	10.12	709.03	4564	17803

The results for the verification of Montgomery multipliers are shown in Table VI. Montgomery multipliers are significantly larger than Mastrovito multipliers. If we represent a polynomial for every gate in the design, then we create too many variables (d) in the system, exceeding SINGULAR'S capacity ($d \leq 32767$). For this reason, we have to cluster/partition the circuit according to a signal's fanin cone, and construct a polynomials for each cluster. Moreover, we ensure that our term ordering constraint is not violated. With such efforts, we are able to verify Montgomery multipliers up to 128-bits, beyond which we still exceed SINGULAR'S capacity. However, our $F4$-style approach has no such limitation, and it is also $> 2X$ faster than the REDUCE operation of SINGULAR. Similarly, the results for verification of Barrett multipliers are shown in Table VII.

TABLE VII: Runtime for verifying bug-free and buggy Barrett multipliers. TO = timeout of 10hrs. Time is given in seconds.

Operand size k	32	48	64	96	128	163
#variables	1103	2389	4146	9216	16072	26847
#polynomials	1041	2263	4004	8986	15008	25746
#terms	6757	15228	26452	60824	107454	174571
Bug-free (Singular)	1.31	22.12	103.30	724.14	2865	14048
Bug-free (F_4)	0.76	7.95	37.45	239.64	1098	6428
Bugs (Singular)	1.32	23.06	106.02	734.63	2947	14836
Bugs (F_4)	0.76	7.97	37.91	241.39	1135	6501

Tables VIII and IX depict the results for verification of ECC point addition and point doubling circuits. As described in Section II, these circuit designs are based on the L*ó*pez-Dahab coordinate system [5]. In the circuits, polynomial multiplication is implemented using Barrett reduction. Our approach can verify 163-bit ECC operations — previously unachievable by other verification techniques.

TABLE VIII: Verification of correct ECC point addition circuits. Run-time given in seconds.

Operand size k	48	64	96	128	160	163
#variables	3623	6854	13986	28468	30237	31384
#polynomials	3489	6612	12548	26835	28319	30024
#terms	86482	123544	288720	509660	604740	646129
Runtime(Singular)	118	557	3598	15346	47290	81016
Runtime($F4$)	42	268	1427	6471	19832	35240

TABLE IX: Verification of correct ECC point doubling circuits. Run-time given is seconds.

Operand size k	48	64	96	128	160	163
#variables	3321	6409	12230	26493	29015	30442
#polynomials	3204	6257	10981	24867	26918	28359
#terms	42324	61274	142733	243452	297465	313145
Runtime(Singular)	54	263	1532	8012	21493	36439
Runtime($F4$)	26	98	683	3128	7648	15235

IX. CONCLUSIONS

A formal approach to model and verify arithmetic circuits over Galois fields $\mathbb{F}_{2^k}$ using a computer-algebra based approach is presented in this paper. Given a specification polynomial f over $\mathbb{F}_{2^k}$, and a gate-level combinational circuit C, we formally prove that C correctly implements f; or disprove the equivalence. The verification problem is formulated as membership testing of the specification polynomial f in a

(radical) ideal $J + J_0 = \langle f_1, \ldots, f_s, x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$; where $J = \langle f_1, \ldots, f_s \rangle$ corresponds to the ideal generated by polynomials extracted from the circuit, and $J_0 = \langle x_i^{2^k} - x_i \rangle$ corresponds to the ideal of vanishing polynomials of the field. The formulation is derived from the application of Strong Nullstellensatz over $\mathbb{F}_{2^k}$. Subsequently, a Gröbner basis G of the ideal $(J + J_0)$ can be computed and the ideal membership test can be decided via Gröbner basis reduction.

The Gröbner basis (Buchberger's) algorithm, however, exhibits high computational complexity, which is very susceptible to the term orderings used to represent and manipulate the polynomials. We show that a specific term ordering can be derived by performing a topological analysis of the circuit. This term ordering renders the set of polynomials itself a (minimal) Gröbner basis – thus obviating the need to apply Buchberger's algorithm. As a consequence of our theoretical deductions, the verification test reduces to a much simpler case of polynomial reduction: $f \xrightarrow{G}_{+} r$, which is performed using multivariate polynomial division. To perform this reduction efficiently, we engineer an $F4$-style approach — where the reduction is performed via Gaussian elimination on a matrix representation of the problem.

Our approach is implemented as a custom verification tool, which is used to conduct experiments for verification of a variety of custom-designed Galois field arithmetic circuits. Using our approach, we can verify up to 163-bit Galois field circuits, whereas contemporary verification approaches are impractical. As compared to the use of a general-purpose computer algebra tool (SINGULAR), our $F4$-style reduction approach gives approximately 2.5X speed-up.

As future work, we would like to develop computer algebra techniques for verification of sequential circuits that perform Galois field arithmetic computations. For verification of such circuits, it is required to analyze the state-space of the sequential circuit. This will require the study of quantifier elimination techniques using Gröbner bases over Galois fields, such as those presented in [40]. We will investigate whether and how the formulations of [40] can be made scalable using the techniques presented in this paper.

REFERENCES

[1] E. Mastrovito, "VLSI architectures for computation in Galois fields," Ph.D. dissertation, Linköping University, Sweden, 1991.

[2] C. Koc and T. Acar, "Montgomery Multiplication in GF(2^k)," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, Apr. 1998.

[3] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF(2^n) Without Pre-Computational Phase," in *Proceedings of the International Workshop on Arithmetic of Finite Fields*, 2008, pp. 77–87.

[4] L. Gao and K. K. Parhi, "Custom vlsi design of efficient low latency and low power finite field multiplier for reed-solomon codec," in *ISCAS (4)*, 2001, pp. 574–577.

[5] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)," in *Proceedings of the Selected Areas in Cryptography*. London, UK, UK: Springer-Verlag, 1999, pp. 201–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=646554.694442

[6] E. Biham, Y. Carmeli, and A. Shamir, "Bug Attacks," in *Proceedings on Advances in Cryptology*, 2008, pp. 221–240.

[7] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[8] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*. American Mathematical Society, 1994.

[9] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal," Ph.D. dissertation, University of Innsbruck, 1965.

[10] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-3 — A computer algebra system for polynomial computations," 2011, http://www.singular.uni-kl.de.

[11] J. C. $Faugère$, "A New Efficient Algorithm for Computing Gröbner Bases (F_4)," *Journal of Pure and Applied Algebra*, vol. 139, pp. 61–88, June 1999.

[12] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.

[13] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," *IEEE Transactions On Computers*, vol. 51, no. 5, May 2002.

[14] Y. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1514–1527, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1109/TC.2008.148

[15] E. Mastrovito, "VLSI Designs for Multiplication Over Finite Fields GF(2^m)," *Lecture Notes in Computer Science*, vol. 357, pp. 297–309, 1989.

[16] P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," in *Proceedings of Advances In Cryptology*. London, UK, UK: Springer-Verlag, 1987, pp. 311–323. [Online]. Available: http://dl.acm.org/citation.cfm?id=36664.36688

[17] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.

[18] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications," in *IEEE Design, Automation and Test in Europe*, 1999, pp. 52–58.

[19] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *Proc. Des. Auto. Conf. (DAC)*, 1996, pp. 629–634.

[20] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, Nov. 2006.

[21] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to Combinational Equivalence Checking," in *Proc. Intl. Conf. on CAD (ICCAD)*, 2006, pp. 836–843.

[22] F. Lu, L. Wang, K. Cheng, and R. Huang, "A Circuit SAT Solver With Signal Correlation Guided Learning," in *IEEE Design, Automation and Test in Europe*, 2003, pp. 892–897.

[23] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*, vol. 6174. Springer, 2010, pp. 24–40.

[24] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann, "Scalable automated verification via expert-system guided transformations," in *Prof. Formal Methods in Computer Aided Design*, 2004, pp. 159–173.

[25] "Cadence encounter conformal equivalence checker," Available at: http://www.cadence.com/products/ld/equivalence_checker/pages/default.aspx.

[26] "Synopsys formality equivalence checker," Available at: http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/Formality.aspx.

[27] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Ordered Kronecker Functional Decision Diagrams," in *Design Automation Conference*, 1994, pp. 415–419.

[28] R. E. Bryant and Y.-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," in *Proceedings of Design Automation Conference*, 1995, pp. 535–541.

[29] T. L. Rajaprabhu, A. K. Singh, A. M. Jabir, and D. K. Pradhan, "MODD for CF: A Compact Representation for Multiple Output Function," in *IEEE International High Level Design Validation and Test Workshop*, 2004.

[30] B. Alizadeh and M. Fujita, "Modular Datapath Optimization and Verification based on Modular-HED," *IEEE Transactions CAD*, pp. 1422–1435, Sept. 2010.

[31] N. Shekhar and *et al.*, "Equivalence Verification of Polynomial Datapaths with Fixed-Size Bit-Vectors using Finite Ring Algebra," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2005.

[32] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing," *IEEE Transactions on CAD*, pp. 1320–1330, July 2007.

[33] S. Morioka, Y. Katayama, and T. Yamane, "Towards Efficient Verification of Arithmetic Algorithms Over Galois Fields $GF(2^m)$," *Computer Aided Verification Conference*, vol. 2102, pp. 465–477, 2001.
[34] D. Mukhopadhyaya, G. Sengar, and D. Chowdhury, "Hierarchical Verification of Galois Field Circuits," *IEEE Transactions on CAD*, 2007.
[35] L. Erkök, M. Carlsson, and A. Wick, "Hardware/Software Coverification of Cryptographic Algorithms using Cryptol," in *Proc. Formal Methods in CAD (FMCAD)*, 2009, pp. 188–191.
[36] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Application of Symbolic Computer Algebra to Arithmetic Circuit Verification," in *IEEE International Conference on Computer Design*, October 2007, pp. 25–32.
[37] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths," in *Computer Aided Verification Conference*, 2008, pp. 473–486.
[38] S. Gao, "Counting Zeros over Finite Fields with Gröbner Bases," Master's thesis, Carnegie Mellon University, 2009.
[39] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G.-M. Greuel, "STABLE: A New QBF-BV SMT Solver for Hard Verification Problems Combining Boolean Reasoning with Computer Algebra," in *IEEE Design, Automation and Test in Europe Conference*, 2011, pp. 155–160.
[40] S. Gao, A. Platzer, and E. Clarke, "Quantifier Elimination over Finite Fields with Gröbner Bases," in *Intl. Conf. Algebraic Informatics*, 2011.
[41] L. Lastras-Montaño, P. Meany, E. Stephens, B. Trager, J. O'Conner, and L. Alves, "A new class of array codes for memory storage," in *Proc. Information Theory and Applications Workshop*, 2011, pp. 1–10.
[42] L. Lastras, A. Lvov, B. Trager, S. Winograd, V. Paruthi, A. El-Zhein, R. Shadowen, and G. Janssen, "New Formal Verification Techniques for Algorithms over Finite Fields," 2012, presented at Intl. Workshop on Internation Theory and Applications. Abstract of the paper available at: http://ita.ucsd.edu/workshop/12/talks.
[43] A. Lvov, L. Lastras-Montaño, V. Paruthi, R. Shadowen, and A. El-Zein, "Formal Verification of Error Correcting Circuits using Computational Algebraic Geometry," in *Proc. Formal Methods in Computer-Aided Design (FMCAD)*, 2012, pp. 141–148.
[44] J. Lv, P. Kalla, and F. Enescu, "Verification of Composite Galois Field Multipliers over GF($(2^m)^n$) using Computer Algebra Techniques," in *IEEE High-Level Design Validation and Test Workshop*, 2011, pp. 136–143.
[45] ——, "Formal Verification of Galois Field Multipliers using Computer Algebra," in *25th IEEE International Conference on VLSI Design*, 2012.
[46] ——, "Efficient Groebner Basis Reductions for Formal Verification of Galois Field Multipliers," in *IEEE Design, Automation and Test in Europe*, 2012.
[47] B. Buchberger, "A criterion for detecting unnecessary reductions in the construction of a groebner bases," in *EUROSAM*, 1979.
[48] D. Andren, L. Hellstrom, and K. Markstrom, "On the complexity of matrix reduction over finite fields," *Advances in applied mathematics*, vol. 39, no. 4, 2007.
[49] J. Lv, "Galois field circuit benchmarks," http://ece.utah.edu/l̃v/uugb.html.