Algorithmic techniques in algebraic cryptanalysis and their applications

Makarim, R.H.

Part II
Technical Preliminaries

2 Multivariate Quadratic (MQ) Problems in Cryptology

Contents

Algebraic cryptanalysis is a generic technique that allows assessment of numerous cryptographic schemes. Its main principle is to express a cryptanalytic problem (e.g., key-recovery, plaintext-recovery, etc) into a system of multivariate polynomial equations. The public parameters, such as description of encryption schemes or public-keys, are used to express the system of equations. The construction of the equations are done in a way that allows a correspondence between the solutions of the system with the secret component of the cryptographic primitive. The polynomials are, in general, quadratic and the coefficients are taken from a finite field.

The security level of an encryption scheme against algebraic cryptanalysis is linked with the difficulty of solving system of polynomial equations over a finite field. It is often the case that a cryptographic primitive can be represented by different system of equations. This cryptanalytic technique has been successfully applied to break the security of multivariate public-key/signature schemes [FJ03, FLP08] as well as certain families of stream ciphers [CP02, CM03, Cou03].

This chapter is meant to provide a bird's-eye view on various aspects of algebraic cryptanalysis. We begin by describing a computationally intractable problem related to solving a system of quadratic polynomial equations over a field called the *Multivariate Quadratic* (MQ) problem in Section 2.1. This hard computational problem is also used as a foundation to construct post-quantum cryptographic primitives. The description of public-key cryptography based on the difficulty of solving polynomial equations is given in Section 2.2. This will be followed by Section 2.3 which provides a survey on how algebraic cryptanalysis plays a role in the cryptanalysis of symmetric-key primitives. We also give a brief overview of the algebraic cryptanalysis against post-quantum

cryptographic schemes in Section 2.4. We will then discuss several methods that have been applied to find a solution of multivariate polynomial equations from different cryptographic primitives. Some of the methods mentioned are algebraic in nature while others require reductions of solving multivariate polynomial equations to other intractable problems. They will be explained in Section 2.5.

2.1 The Intractability of the MQ Problem

The central subject of this thesis is techniques to solve quadratic systems of polynomial equations over a finite field. The term "solve" or "solving" can be interpreted in multiple ways. One may view it as an effort to recover all solutions of the system of equations, including the one in the algebraic closure of the coefficient field. However, in the context of cryptanalysis the term "solve" and "solving" are restricted to finding a solution that lies in the coefficient field. Definition 2.1 formally defines this perspective.

Definition 2.1 (Polynomial System Solving)**.** *Let R be a commutative ring with a multiplicative identity $1 \neq 0$ and let $f_1, \ldots, f_m \in R[x_1, \ldots, x_n]$ be m polynomials over n-variables $x_1, \ldots, x_n$ with coefficients in R. The polynomial system solving problem (PoSSo) for $f_1, \ldots, f_m$ over R is the problem of finding an element $(a_1, \ldots, a_n) \in R^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i \in \{1, \ldots, m\}$. We call the set $\{f_1, \ldots, f_m\}$ an instance of the PoSSo problem over R.*

Definition 2.2 (MQ problem)**.** *An instance $F = \{f_1, \ldots, f_m\} \subset R[x_1, \ldots, x_n]$ of the PoSSo problem over R is called a multivariate quadratic (MQ) problem over R if the degree of f_i, $1 \leq i \leq m$, is (at most) quadratic.* The decisional MQ problem over R asks whether there exists an assignment $(a_1, \ldots, a_n) \in R^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i \in \{1, \ldots, m\}$.*

We will show that the decisional MQ problem over $\mathbb{F}_2$ is NP-complete. The proof is based on the reduction to the following Boolean satisfiability problem.

Definition 2.3 (3-CNF SAT Problem)**.** *Let $X = \{X_1, \ldots, X_n\}$ be a set of Boolean variables, let $L = \{X_1, \neg X_1, \ldots, X_n, \neg X_n\}$ be the set of its corresponding literals (where $\neg X_i$ denotes the negation of X_i), and let $\wedge, \vee$ denote Boolean* `AND` *and* `OR` *respectively. A Boolean formula $C = \{c_1, \ldots, c_m\}$ is in 3-CNF (Conjunctive Normal Form) if it is of the form*

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m$$

where $c_i \in (L \cup L^2 \cup L^3)$ is a Boolean `OR` *of at most three literals (we shall call c_i a clause). The corresponding 3-CNF SAT problem is the problem to determine if there is a valid assignment $A \in \{\texttt{True}, \texttt{False}\}^n$ for X such that all $c_i \in C$ are true and, hence, C itself is satisfied.*

The Cook-Levin theorem [Coo71, Lev73] states that the Boolean satisfiability problem is NP-complete. Based on this result, the following proposition shows that the decisional MQ problem over $\mathbb{F}_2$ is also NP-complete.

Proposition 2.4. *The decisional MQ problem over $\mathbb{F}_2$ is NP-complete.*

* See Definition 3.12 for the definition of the degree of a polynomial.

Proof. The proof here is adapted from subsection 2.5.1 of [Wol05]. The outline of this proof consists of two parts. We first show that solving an instance of MQ problem over $\mathbb{F}_2$ is in NP, i.e., there exists a non-deterministic polynomial-time algorithm that finds a solution if one exists. This will be followed by the second part that shows there exists a polynomial-time reduction algorithm that transforms an instance of 3-CNF SAT problem to an instance of MQ problem over $\mathbb{F}_2$.

Let $(f_1, \ldots, f_m) \in \mathbb{F}_2[x_1, \ldots, x_n]$ be an instance of MQ problem over $\mathbb{F}_2$. There exists a non-deterministic algorithm that finds, in polynomial time, an element $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i = 1, 2, \ldots, m$ if one exists. The algorithm is described as follows

1. Select an element $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$.
2. If $f_i(a_1, \ldots, a_n) = 0$ for all $i = 1, 2, \ldots, m$, then return `True`. Otherwise, go to step 1.

Note that the complexity of step (2) requires $m \sum_{i=0}^{2} \binom{n}{i} = m(1+n+n(n-1)/2)$ field operations, thus step (2) has polynomial-time complexity. The algorithm is terminated if step (2) outputs `True`. Otherwise, it goes to an infinite loop. Thus, the MQ problem over $\mathbb{F}_2$ is in NP.

In order to prove the NP-completeness, we reduce an instance of 3-CNF SAT problem to an instance of MQ problem over $\mathbb{F}_2$. Let C be a set of clauses of n Boolean variables in 3-CNF and $|C| = m$. Here we associate the Boolean value `True` to $1 \in \mathbb{F}_2$ and `False` to $0 \in \mathbb{F}_2$. Similarly, for each $i = 1, \ldots, n$ we associate each Boolean variable X_i to its corresponding indeterminate x_i in $\mathbb{F}_2[x_1, \ldots, x_n]$. The following Algorithm 2.1 transforms C to a set of multivariate polynomial equations over $\mathbb{F}_2$.

Input: A set of clauses $C = \{c_1, \ldots, c_m\}$ in CNF with variables $X_1, \ldots, X_n$
Output: A subset $F = \{f_1, \ldots, f_m\} \subset \mathbb{F}_2[x_1, \ldots, x_n]$

```
1  F ← {}
2  for i = 1 to m do
3      f_i ← 1 ∈ F_2[x_1, ..., x_n]
4      forall ℓ ∈ c_i do
5          if ℓ = ¬X_j, j ∈ {1, ..., n} then
6              f_i ← f_i · x_j
7          else if ℓ = X_j, j ∈ {1, ..., n} then
8              f_i ← f_i · (x_j − 1)
9      F ← F ∪ {f_i}
10 return F
```

Algorithm 2.1: Convert a 3-CNF problem into a MQ-problem over $\mathbb{F}_2$.

Note that an assignment $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ to the variables of a polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is considered "valid" if $f(a_1, \ldots, a_n) = 0$.* Each clause $c_i \in C$

* By associating `True` $\leftrightarrow$ 1 and `False` $\leftrightarrow$ 0, the "satisfiability" of a polynomial is the opposite of a clause. However, this does not affect the correctness of the result since the satisfiability is not related with any assignment to the variables.

is transformed to its corresponding polynomial equation $f_i \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree ≤ 3.

In order to reduce the degree of polynomials in F to quadratic, we introduce $n(n-1)/2$ additional variables $y_{i,j}$ that substitute the quadratic monomials $x_i x_j$, $1 \leq i < j \leq n$. At the same time $n(n-1)/2$ equations of the form $x_i x_j + y_{i,j}$ are also added to F. This yields a system of $m + n(n-1)/2$ quadratic polynomials in $n(n+1)/2$ variables. If there exists a solution for this system of equations, then we also have a solution to the 3-CNF SAT.* Since all the steps require polynomial time and space, we have reduced an instance of 3-CNF SAT in polynomial time to an instance of the MQ problem over $\mathbb{F}_2$. Therefore, the decisional MQ problem over $\mathbb{F}_2$ is NP-complete. ■

Proposition 2.5. *Let D be a finite integral domain. Then, the MQ problem over D is NP-hard. In particular, if the addition and multiplication in D are polynomial time operations, then the decisional MQ problem over D is NP-complete.*

Proof. The following proof is a summary of subsection 2.5.2 in [Wol05]. We first prove that MQ over a finite integral domain D is NP-hard. The proof is based on reduction of an instance of MQ over $\mathbb{F}_2$ to an instance of MQ over D.

Consider the following quadratic system of m equations in n variables over $\mathbb{F}_2$

$$\begin{aligned} \sum_{1 \leq j < k \leq n} a_{j,k}^{(1)} x_j x_k + \sum_{j=1}^{n} b_j^{(1)} x_j + c^{(1)} &= 0 \\ &\vdots \\ \sum_{1 \leq j < k \leq n} a_{j,k}^{(m)} x_j x_k + \sum_{j=1}^{n} b_j^{(m)} x_j + c^{(m)} &= 0 \end{aligned}$$

where $a_{j,k}^{(i)}, b_j^{(i)}, c^{(i)} \in \mathbb{F}_2$ are constants, for $1 \leq i \leq m$. Each polynomial above is transformed into the following system of polynomial equations over $\mathbb{F}_2$

$$\begin{aligned} a_{1,2}^{(i)} x_1 x_2 &= y_{1,2}^{(i)} \\ a_{1,3}^{(i)} x_1 x_3 &= y_{1,2}^{(i)} + y_{1,3}^{(i)} \\ &\vdots \\ a_{n-1,n}^{(i)} x_{n-1} x_n &= y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)} \\ b_1^{(i)} x_1 &= y_{n-1,n}^{(i)} + z_1^{(i)} \\ b_2^{(i)} x_2 &= z_1^{(i)} + z_2^{(i)} \\ &\vdots \\ b_{n-1}^{(i)} x_{n-1} &= z_{n-2}^{(i)} + z_{n-1}^{(i)} \\ b_n^{(i)} x_n + c^{(i)} &= z_{n-1}^{(i)} \end{aligned} \tag{2.1}$$

* For instance, a clause $(X_1 \vee X_2 \vee \neg X_3)$ is converted into a polynomial $f = (x_1 + 1)(x_2 + 1)x_3$ by Algorithm 2.1. The set of solutions to f is $\mathbb{F}_2^3 \setminus \{(0,0,1)\}$ and one can also verify that the set of valid assignments to X_1, X_2, X_3 is $\{\texttt{True}, \texttt{False}\}^3 \setminus \{(\texttt{False}, \texttt{False}, \texttt{True})\}$.

where $y_{j,k}^{(i)}$ and $z_j^{(i)}$ are new variables. Each polynomial is transformed into a set of $n(n+1)/2$ equations with $n+\binom{n}{2}+(n-1) = (n(n+3)-2)/2$ variables. Thus, the original system of equations is transformed into a system of $mn(n+1)/2$ equations in $n + m\binom{n}{2} + m(n-1) = (2n + m(n-1)(n+2))/2$ variables. This step requires polynomial-time in the size of the input.

The next step is to transfer the newly constructed system of equations over $\mathbb{F}_2$ into a system of equations over D. This is done by considering the assignment on each variables $x_j, y_{j,k}^{(i)}, z_j^{(i)}$ using the elements of D and each coefficient $a_{j,k}^{(i)}, b_j^{(i)}, c^{(i)} \in \mathbb{F}_2$ is replaced by either $0 \in D$ or $1 \in D$. The multiplication in $\mathbb{F}_2$ is replaced by the multiplication in D. However, the addition operation in $\mathbb{F}_2$ needs to be replaced by $(x+y)((1+1)-(x+y))$ in D. Thus, from (2.1) we obtain the following system of equations over D

$$\begin{aligned}
a_{1,2}^{(i)} x_1 x_2 &= y_{1,2}^{(i)} \\
a_{1,3}^{(i)} x_1 x_3 &= (y_{1,2}^{(i)} + y_{1,3}^{(i)})((1+1) - (y_{1,2}^{(i)} + y_{1,3}^{(i)})) \\
&\vdots \\
a_{n-1,n}^{(i)} x_{n-1} x_n &= (y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)})((1+1) - (y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)})) \\
b_1^{(i)} x_1 &= (y_{n-1,n}^{(i)} + z_1^{(i)})((1+1) - (y_{n-1,n}^{(i)} + z_1^{(i)})) \\
b_2^{(i)} x_2 &= (z_1^{(i)} + z_2^{(i)})((1+1) - (z_1^{(i)} + z_2^{(i)})) \\
&\vdots \\
b_{n-1}^{(i)} x_{n-1} &= (z_{n-2}^{(i)} + z_{n-1}^{(i)})((1+1) - (z_{n-2}^{(i)} + z_{n-1}^{(i)})) \\
z_{n-1}^{(i)} &= (b_n^{(i)} + c^{(i)})((1+1) - (b_n^{(i)} + c^{(i)}))
\end{aligned}$$

for all $i = 1, \ldots, m$. In order to restrict the solution in the set $\{0,1\} \subset D$ we add new equations of the form $x(1-x)$ for each variable. The number of such equation is equal to $(2n + m(n-1)(n+2))/2$. The system of equations over D consists of $n + m(n^2+n-1)$ equations in $(2n + m(n-1)(n+2))/2$ variables. Thus, an instance of MQ problem over $\mathbb{F}_2$ is transformed into an instance of MQ problem over D in polynomial-time complexity. The existence of a solution for MQ problem over D implies the existence of a solution for MQ problem over $\mathbb{F}_2$.

The last part of this proof is to show that if addition and multiplication in D can be done in polynomial-time, then the MQ problem over D is NP-complete. Consider the following nondeterministic algorithm to find a solution of an MQ problem over D

1. Select an element $(d_1, \ldots, d_n) \in D^n$ for variables $x_1, \ldots, x_n$.
2. If all m equations over D are satisfied by $(d_1, \ldots, d_n)$ then return `True`. Otherwise, go to step 1.

Note that if addition and multiplication in D can be computed in polynomial-time, then step 2 in the above procedure is also polynomial-time. Therefore, the decisional MQ problem over D, where operations in D are computable in polynomial-time, is NP-complete. Otherwise, the MQ problem over D is NP-hard. ■

2.2 Multivariate Public-Key Cryptography

In Chapter 1 we have discussed the importance of public-key cryptography in solving the problem of key-distribution for symmetric-key primitives using insecure channels. Two of the most successful proposals today are based on the difficulty of integer factorization [RSA78] and computation of discrete logarithms in a cyclic group [Gam84]. Although both problems are regarded as hard problems for classical computers, in 1997 Peter Shor published a polynomial-time quantum algorithm to solve integer factorization and discrete logarithm [Sho97].

This situation has prompted researchers to revisit public-key proposals in the past that were based on different computational hard problems. In 1985, Matsumoto and Imai introduced the first construction of multivariate public-key cryptography in which the security is based on the difficulty of solving an MQ-problem over $\mathbb{F}_2$ [IM85]. Since then, various other public-key cryptosystems based on the hardness of solving systems of polynomial equations over (small) finite field have been proposed [Pat96, PBD14, KPG99, PCDY17, DS05, Din04, TDTD13, CBD+09]. This also has led to several submitted proposals to the NIST post-quantum cryptography standardization process that are based on the hardness of the MQ problem over finite fields: DME [Lue17], RAINBOW [DS05], LUOV [BP17], DUALMODEMS [FPR17a], GEMSS [FPR+17b], GUI [DCP+17], HIMQ-3 [SPK17] and MQDSS [CHR+17]. Despite the devastating attack on RAINBOW [Beu22], which is the multivariate signature scheme that advances the farthest from the initial NIST call-for-proposals for the post-quantum cryptography standardization, the interest in developing other multivariate-based digital signature schemes remains high due to its two primary advantages: small signature size and fast verification. This is reflected in the 10 out of 40 submissions to the NIST call for additional post-quantum digital signature schemes standardization that are classified under the category of multivariate signatures: 3WISE [Rod23a], DME SIGN [Ln23], HPPC [Rod23b], MAYO [BCC+23], PROV [GCF+23], QR-UOV [FIH+23], SNOVA [WCD+23], TUOV [DGG+23], UOV [BCD+23], and VOX [PCF+23]. Moreover, the security of two other submissions that are classified under MPC-in-the-Head signatures, namely BISCUIT [BKPV23] and MQOM [FR23], are also based on the hardness of solving multivariate polynomial equations over finite field.

The public-key of a multivariate-based encryption/signature scheme consists of quadratic polynomials in $F = (f_1, \ldots, f_m) \subset \mathbb{F}_q[x_1, \ldots, x_n]$ (though there is also a construction that uses cubic polynomials [DPW14]). Note that F has three different natures: as an ordered subset of the ring $\mathbb{F}_q[x_1, \ldots, x_n]$, as a system of equations $f_1 = 0, \ldots, f_m = 0$, and as a function $F : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$.

The construction of F as a function is a composition of the following three functions

$$F = B \circ \bar{F} \circ A$$

where $A : \mathbb{F}_q^n \mapsto \mathbb{F}_q^n$ and $B : \mathbb{F}_q^m \mapsto \mathbb{F}_q^m$ are invertible affine transformations and $\bar{F} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$ is a quadratic map. The affine functions A and B are randomly selected and their purpose is to hide the algebraic structure of $\bar{F}$ given the knowledge of F. The construction of $\bar{F}$ must satisfy two main requirements. The first one is, given any y in the image of $\bar{F}$, it should be computationally feasible to find $x \in \mathbb{F}_q^n$ such that $\bar{F}(x) = y$. The second requirement is that the complexity of solving $F = B \circ \bar{F} \circ A$ as a system of equations must be as close

as possible to the complexity of solving random quadratic system of equations over $\mathbb{F}_q$. The public-key consists of m polynomial components of F and the private-key consists of A and B. The function $\bar{F}$ may or may not be a part of the private-key and that depends on its exact nature.

Public-key encryption In order to use MQ problems as an encryption function, the function $\bar{F}$ must be injective, i.e., the preimage of every element in the image of $\bar{F}$ must be unambiguously determined. This implies that F is an overdefined system of equations where $m \geq n$. The encryption of a plaintext $P \in \mathbb{F}_q^n$ is performed by computing $C = F(P) \in \mathbb{F}_q^m$. (When $\bar{F}$ is non-injective, one can concatenate the plaintext and its hash value before encryption. The hash value helps to determine the correct preimage when inverting $\bar{F}$.)

The receiver computes $A^{-1}(\bar{F}^{-1}(B^{-1}(C)))$ to recover the original plaintext P. Given F and C, in order to recover the plaintext P without the knowledge of $A, B, \bar{F}$, an attacker must solve the system of equations $F(x_1, \ldots, x_n) = C$.

Public-key signature In the case of an MQ-based digital signature, the function $\bar{F}$ should be close to surjective. This implies that F is an underdefined system of equations where $m \leq n$. A signature S for a plaintext $P \in \mathbb{F}_q^m$ is computed as $S = A^{-1}(\bar{F}^{-1}(B^{-1}(P)))$.

The verifier computes $F(S)$ and checks if $P = F(S)$. In order to forge a signature, an attacker has to solve the polynomial system $F(x_1, \ldots, x_n) = P$.

2.3 Algebraic Cryptanalysis of Symmetric-Key Cryptography

In this section we explain how the problem of solving a system of multivariate polynomial equations over a finite field, especially over $\mathbb{F}_2$, occurs in the cryptanalysis of symmetric-key primitives. We focus particularly on the cryptanalysis of block ciphers and stream ciphers.

2.3.1 Algebraic Cryptanalysis of Block Ciphers

Let $\mathbb{F}_2^n$ be the plaintext/ciphertext space and $\mathbb{F}_2^\kappa$ be the key space. The function $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \mapsto \mathbb{F}_2^n$ is an n-bit block cipher with κ-bit secret key if for every $K \in \mathbb{F}_2^\kappa$ the function

$$\begin{aligned} E_K : \mathbb{F}_2^n &\mapsto \mathbb{F}_2^n \\ P &\mapsto E(P, K) \end{aligned} \tag{2.2}$$

is invertible. A block cipher E is called an *r-round iterated block cipher* if

$$E_K(P) = R_{K_r}(R_{K_{r-1}}(\ldots(R_{K_1}(P)))).$$

where $R : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \mapsto \mathbb{F}_2^n$ is a *round function* of E and R_{K_i} is defined similarly as in (2.2) with respect to R for $i = 1, \ldots, r$ and $K_i \in \mathbb{F}_2^\kappa$. The keys K_i are

called the *round keys* and they are derived from the original key K using a *key-schedule algorithm*. In this subsection, the term block cipher refers to iterated block cipher.

Recall that conventional cryptanalytic approaches to block ciphers are dominated by two techniques: linear cryptanalysis [Mat93] and differential cryptanalysis [BS90]. In the case of linear cryptanalysis an attacker studies linear relations, i.e., the parity of chosen bits of the plaintext, ciphertext, and the key, with high probability. On the other hand, differential cryptanalysis studies the difference between two plaintexts and how this difference evolves in various operations of the block cipher. Many other cryptanalytic techniques for block ciphers are derived from the two techniques mentioned previously [Knu94, KR96, BBS99, LH94, HCN19]. These techniques are all statistical in nature and they require large number of plaintexts and ciphertexts in order to use them to obtain partial or full information about the key with high success probability.

The approach to algebraically mount a key-recovery attack on a block cipher E starts by viewing E as a system of multivariate polynomial equations over $\mathbb{F}_2$. Given a plaintext and its corresponding ciphertext, the indeterminates in this system of equations represent the internal state of the block cipher and the unknown key. Solving this system means recovering full information about the key as well as the internal state. In contrast to other conventional cryptanalysis techniques for block ciphers, this approach is deterministic and it requires only a handful of plaintexts-ciphertexts to obtain full information about the key.

In practice, modelling the encryption function of a block cipher as a system of equations considers each layer of a round function (linear, nonlinear, key addition) separately. For a block cipher with an ℓ layer of operations, the model defines the internal state variables $\mathcal{X}_i = \{x_{i,1}, \ldots, x_{i,n}\}, i = 0, 1, \ldots, \ell$. The set $\mathcal{X}_i$ has dual roles: it represents the output variables of the i-th layer and the input variables for the $(i+1)$-th layer. The set $\mathcal{X}_0$ and $\mathcal{X}_\ell$ are the set of plaintext and ciphertext variables, respectively. Let $\mathcal{K} = \{k_1, \ldots, k_\kappa\}$ denote the set of key variables. If the round keys are generated by selecting a subset of bits from the master key K, then $\mathcal{K}$ is the set of all key variables in the system of equations. Otherwise, for each $i = 1, \ldots, r$ and for some $j \in \{1, \ldots, n\}$, the model defines the set $\mathcal{K}_i = \{k_{i,1}, \ldots, k_{i,j}\}$ as the variables for the i-th round key. The system of equations for a block cipher can then be divided into three disjoint sets: (1) a set of linear equations that represent the linear layers and key addition, (2) a set of non-linear equations describing the non-linear operations of the cipher, and (3) a set of equations that describes the key-schedule algorithm.

Generating a system of equations for linear layers is rather obvious. For the non-linear layers, block ciphers in general employ a nonlinear map $S : \mathbb{F}_2^s \mapsto \mathbb{F}_2^t$ with relatively small s, t commonly referred to as *substitution boxes* (S-Box). We describe two approaches to obtain a system of polynomial equations that represent S. We denote by x_i, y_j $(1 \leq i \leq s, 1 \leq j \leq t)$ the input and output variable of S respectively. An S-Box S can be viewed as a parallel applications of t s-variable Boolean functions i.e, $S(x) = (h_1(x), \ldots, h_t(x))$ where $h_i : \mathbb{F}_2^s \mapsto \mathbb{F}_2$ for $i = 1, \ldots, t$. The first approach to obtain a set of polynomials that represent S is to compute the *algebraic normal form* of h_i for each $i = 1, \ldots, t$ (see [Car10, pg. 9] for the definition and the algorithm to compute the algebraic normal form of a Boolean function). An example of this approach is given in Figure 5. The second approach is described in [BC03] and the goal is to obtain as many linearly

independent polynomial equations as possible. Firstly, it selects $d \in \mathbb{Z}$ such that $\sum_{i=0}^{d} \binom{s+t}{i} > 2^s$. In order to find all linearly independent equations involving monomials of degree $\leq d$, we construct a $\sum_{i=0}^{d} \binom{s+t}{i} \times (2^s+1)$ matrix containing a separate row for each monomial, where the last column of the matrix consists of these monomials. The entries from column 1 to the column 2^s corresponding to the different values of the particular monomial for all possible input and output values. The final step is to compute the (reduced)-row echelon form of the matrix and all row operations required by this step are applied to the corresponding monomials as well. In this way, the zero rows appear in the (reduced)-row echelon form of the matrix and all the polynomials corresponding to these zero rows are a set of linearly independent equations representing the S-Box. We provide an example for both approaches in Figure 6.

$$
\begin{aligned}
&y_1 + x_1x_2 + x_1x_3 + x_1 + x_2x_3 + x_2 + 1,\\
&y_2 + x_1x_3 + x_2 + 1,\\
&y_3 + x_1 + x_2x_3 + x_2 + x_3 + 1
\end{aligned}
$$

Figure 5: *An example of a system of equations representing the S-Box defined by the lookup table* $(7, 6, 0, 4, 2, 5, 1, 3)$ *by computing the algebraic normal form of each coordinate function.*

$$
\left[\begin{array}{cccccccc|l}
1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&x_1\\
0&0&1&1&0&0&1&1&x_2\\
0&0&0&0&1&1&1&1&x_3\\
1&0&0&0&0&1&1&1&y_1\\
1&1&0&0&1&0&0&1&y_2\\
1&1&0&1&0&1&0&0&y_3\\
0&0&0&1&0&0&0&1&x_1x_2\\
0&0&0&0&0&1&0&1&x_1x_3\\
0&0&0&0&0&1&0&1&x_1y_1\\
0&1&0&0&0&0&0&1&x_1y_2\\
0&1&0&1&0&1&0&0&x_1y_3\\
0&0&0&0&0&0&1&1&x_2x_3\\
0&0&0&0&0&0&1&1&x_2y_1\\
0&0&0&0&0&0&0&1&x_2y_2\\
0&0&0&1&0&0&0&0&x_2y_3\\
0&0&0&0&0&1&1&1&x_3y_1\\
0&0&0&0&1&0&0&1&x_3y_2\\
0&0&0&0&0&1&0&0&x_3y_3\\
1&0&0&0&0&0&0&1&y_1y_2\\
1&0&0&0&0&1&0&0&y_1y_3\\
1&1&0&0&0&0&0&0&y_2y_3
\end{array}\right]
\rightarrow
\left[\begin{array}{cccccccc|l}
1&0&0&0&0&0&0&0&x_1y_3+y_3\\
0&1&0&0&0&0&0&0&x_1y_3+x_1+x_3+y_1+y_2\\
0&0&1&0&0&0&0&0&x_1y_3+x_1+x_2+x_3+y_2+y_3\\
0&0&0&1&0&0&0&0&x_1y_3+x_2+x_3+y_1+1\\
0&0&0&0&1&0&0&0&x_1y_3+x_3+y_1+y_3\\
0&0&0&0&0&1&0&0&x_1y_3+x_1+x_2+y_2+1\\
0&0&0&0&0&0&1&0&x_1y_3+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&1&x_1y_3+x_1\\
\hline
0&0&0&0&0&0&0&0&x_1x_3+x_2+y_2+1\\
0&0&0&0&0&0&0&0&x_1y_1+x_2+y_2+1\\
0&0&0&0&0&0&0&0&x_1y_2+x_3+y_1+y_2\\
0&0&0&0&0&0&0&0&x_1x_2+x_1+x_2+x_3+y_1+1\\
0&0&0&0&0&0&0&0&x_2x_3+x_1+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&0&x_2y_1+x_1+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&0&x_1y_3+x_2y_2+x_1\\
0&0&0&0&0&0&0&0&x_1y_3+x_2y_3+x_2+x_3+y_1+1\\
0&0&0&0&0&0&0&0&x_1y_3+x_3y_1+y_1+y_3\\
0&0&0&0&0&0&0&0&x_3y_2+x_1+x_3+y_1+y_3\\
0&0&0&0&0&0&0&0&x_1y_3+x_3y_3+x_1+x_2+y_2+1\\
0&0&0&0&0&0&0&0&y_1y_2+x_1+y_3\\
0&0&0&0&0&0&0&0&y_1y_3+x_1+x_2+y_2+y_3+1\\
0&0&0&0&0&0&0&0&y_2y_3+x_1+x_3+y_1+y_2+y_3
\end{array}\right]
$$

Figure 6: *An example of a system of equations representing the S-Box defined by the lookup table* $(7, 6, 0, 4, 2, 5, 1, 3)$ *using the method described in [BC03]. The zero-rows correspond to the desired polynomial equations for the S-Box.*

In order to restrict the set of solutions to the base field $\mathbb{F}_2$, we add polynomials of the form $x^2 + x$ for all variables x in the system of equations. This is equivalent to working in the quotient ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$. Magma [BCP97] and PolyBoRi/BRiAl [BD09b] offer dedicated implementation for polynomial computations in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$.

The systems of equations arising from block ciphers usually have large num-

ber of variables and equations that are sparse and highly structured. Table 3 describes the size of the equation system from several block ciphers such as MISTY1 [Mat97], KHAZAD [BR00], KASUMI [Pro99], and CAMELLIA-128 [AIK+00] and SERPENT-128 [BAK98].

	# variables	# equations
KHAZAD	6464	7664
MISTY1	3856	3856
KASUMI	4264	4264
CAMELLIA-128	3584	6224
SERPENT-128	16640	17680

Table 3: *The size of equation system from several block ciphers [BC03].*

Note that due to the large number of variables and equations, the direct approach to solve a system of polynomial equations from a block cipher often turns out to be infeasible. A more promising approach is to combine generic methods to solve polynomial equations together with techniques that exploit the structural properties of block ciphers.

One direction is to employ a *divide-and-conquer* approach. For instance, the iterative nature of a block cipher allows the equations to be splitted into several subsystems. Let r be the number of rounds and assume that r is even. We split the equations into two subsystems represent the first and the last $r/2$ rounds. The input variables of the second subsystem correspond to the output variables of the first subsystem. The goal is to eliminate variables that do not appear in the round $r/2$ and $r/2+1$ by computing the Gröbner bases of the first and the second subsystem independently (see Chapter 3 for the definition of Gröbner bases). This *meet-in-the-middle* approach was proposed and tested by Cid et al. against small-scale variants of Advanced Encryption Standard (AES) [CMR05, CMR06]. Another divide-and-conquer approach, proposed by Albrecht [Alb07], is based on an incremental method. The idea is to compute the Gröbner basis of the whole system round-by-round, starting from the first round. A Gröbner basis corresponding to the ideal generated by polynomials at the i-th round is added to the set of polynomials for the $(i+1)$-th round. This is iteratively applied until the system of equations for the last round.

Another approach in the algebraic cryptanalysis on block ciphers is to combine it with conventional cryptanalysis techniques such as linear or differential cryptanalysis. Albrecht and Cid [AC09] proposed an attack that combines an algebraic attack with differential cryptanalysis. If an r'-rounds differential characteristic was found, to perform a key-recovery attack typically an attacker tries to guess several key bits in the last $r_d = r - r'$ rounds for an r-rounds block cipher. The authors used algebraic techniques to extend r_d from 2 to 4 rounds for the block cipher PRESENT [BKL+07]. The proposed method is to construct polynomial equations for many plaintext pairs and connect pairs using a set of linear equations that represent a difference in the plaintext. A Gröbner basis algorithm is used to check if the equations satisfy the differential characteristic, instead of directly performing a key-recovery attack.

2.3.2 Algebraic Cryptanalysis of Stream Ciphers

Even though algebraic cryptanalysis did not receive much success in attacking block ciphers, it is considered to be an effective approach to attack several classes of stream ciphers [CM03, Cou03]. The goal of algebraic cryptanalysis of stream ciphers is to recover its initial state from some subset of the keystream. The attack model assumes that the attacker has access to some keystream bits and they need not even be consecutive.

There are two classical stream cipher constructions based on LFSR. The first construction uses multiple LFSRs in parallel and combines their outputs with a Boolean function. We refer to this construction as a *combination generator*. Another construction employs one LFSR together with a Boolean function with the LFSR state as its input to generate an output sequence. This construction is known as a *filter generator*. We refer to the Boolean function used to filter the output of multiple LFSRs in a combination generator or the internal state of the stream cipher in a filter generator as a *non-linear filter*. A standard cryptographic criterion for a non-linear filter is that its algebraic degree should be sufficiently high. The high degree of the non-linear filter provides resistance against algebraic cryptanalysis. In [CM03], Courtois and Meier introduced a simple yet a powerful strategy to significantly reduce the degree of the polynomials from a combination generator or a filter generator. The main idea is to multiply each polynomial by a well-chosen nonzero polynomial such that the resulting product is of low degree.

In this section, we outline the main idea behind the attack of Courtois and Meier. We restrict ourselves to stream ciphers in which the state and the keystream are elements of $\mathbb{F}_2$ and they generate one output bit at a time. The Boolean function that computes the output bit from the internal state of the stream cipher is denoted by f. The function L that computes the next state of a stream cipher is a linear transformation over $\mathbb{F}_2$ and we assume L is public. Note that the computation on polynomials representing stream ciphers is done on the ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.

Let $s_1, \ldots, s_n$ be variables that represent the initial state of a stream cipher and let z_t denote the variables that represent the output bit generated at the time t. The polynomial equation of the stream cipher that represent the computation of the output bit at time t is given by

$$f(L^t(s_1, \ldots, s_n)) = z_t \qquad t = 0, 1, 2, \ldots.$$

The main idea of Courtois-Meier attack is to find a well-chosen multivariate polynomial g such that multiplying g by $f(L^t(s_1, \ldots, s_n)) + z_t$ yields a substantially lower degree polynomial. For instance, when $z_t = 0$ then we can get an equation $f(L^t(s_1, \ldots, s_n))g(L^t(s_1, \ldots, s_n)) = 0$ of degree lower than the degree of $f(L^t(s_1, \ldots, s_n))$. With sufficiently many keystream bits, an attacker can obtain an overdefined system of equations that is efficiently solvable. Note that in case $z_t = 1$, then g is also required to be a low degree polynomial. We may also use different g for the same f in order to generate more polynomial equations. The polynomial g is called the *annihilator* of f. The minimum degree of g such that g is an annihilator of f is called the *algebraic immunity* of f. Courtois and Meier proved that there exists such g of degree less than or equal to $\lceil n/2 \rceil$ such that the degree of fg is at most $\lceil (n+1)/2 \rceil$ (see Theorem 6.0.1 in [CM03]). We refer to the Section 9 of [Car10] for further discussion about algebraic im-

munity, its relation to other cryptographic criteria of Boolean functions, and several known functions that have high algebraic immunity.

The next question arises is how to find such g. One strategy is to first consider terms of high degree in f and see if they share a common non-trivial low degree divisor g'. The product fg with $g = g' - 1$ yields cancellation of terms of f divisible by g', which shows that the polynomial fg is of low degree.

Courtois and Meier demonstrated their idea to attack the stream cipher Toyocrypt(see [DCG+01, Section 2] for a full description of ToyoCrypt). It is a filter generator stream cipher with one 128-bit LFSR and a non-linear Boolean function of the form

$$f(x_1, \ldots, x_{128}) = x_{128} + \sum_{i=1}^{63} x_i x_{\alpha_i} + x_{11}x_{24}x_{33}x_{43} + x_2x_3x_{10}x_{13}x_{19}x_{21}x_{24}x_{26}x_{27}x_{29}x_{34}x_{39}x_{42}x_{43}x_{52}x_{54}x_{60} + \prod_{i=1}^{63} x_i$$

with $\{\alpha_1, \ldots, \alpha_{63}\}$ being some permutation of the set $\{64, \ldots, 126\}$. Observe that that the terms of degree 4, 17, and 63 have a common factor $x_{24}x_{43}$. Let $g = x_{24} - 1$. Multiplying f by g yields a cubic polynomial since all the terms of f divisible by x_{24} are cancelled out. Similarly, the polynomial $f \cdot (x_{43} - 1)$ is also of degree 3 by the same reasoning. Thus, for each keystream bit, we can generate two cubic polynomials in variables $s_1, \ldots, s_{128}$.

Let m be the number of known keystream bits, which corresponds to the number of polynomials. Once an attacker obtains $m \geq \sum_{i=0}^{d} \binom{n}{i}$ keystream bits, the initial state of the stream cipher is recoverable in a polynomial-time using standard techniques from linear algebra where each monomial in the system of equations is treated as an independent variable.

In case of Toyocrypt, there are $T = \sum_{i=0}^{3} \binom{128}{i}$ monomials of degree less than or equal to 3. Recall that for each keystream bit we obtain two linearly independent cubic polynomials. By obtaining $T/2$ keystream bits, the initial state can be recovered in T^ω bit operations where $2 \leq \omega \leq 3$ is the linear algebra constant.*

A similar attack is also applied against Lili-128, an irregularly clocked stream cipher consisting two LFSRs of size 39-bit and 89-bit [SDGM00]. The first LFSR of size 39-bit is used to clock the second LFSR, in which the output is filtered by a non-linear filter f. The non-linear filter in Lili-128 is a 10-variable Boolean function of degree 6. The algebraic normal form of the non-linear filter is the following

$$\begin{aligned} f(x_1, \ldots, x_{10}) = {} & x_1x_8x_9x_{10} + x_1x_8 + x_1x_9 + x_2x_7x_8x_9 + x_2x_7x_9x_{10} + x_2x_8 + \\ & x_2x_9x_{10} + x_2 + x_3x_7x_8x_9x_{10} + x_3x_7x_8x_{10} + x_3x_7x_9 + x_3x_8x_9x_{10} + x_3x_8x_9 + \\ & x_3x_8x_{10} + x_3x_9x_{10} + x_3x_9 + x_3 + x_4x_6x_7x_8x_9x_{10} + x_4x_6x_7x_8x_9 + \\ & x_4x_6x_7x_9x_{10} + x_4x_6x_7x_9 + x_4x_7x_8x_9x_{10} + x_4x_7x_8x_9 + x_4x_7x_9x_{10} + x_4x_7x_9 + \\ & x_4x_7x_{10} + x_4x_8x_9x_{10} + x_4x_8x_{10} + x_4x_9x_{10} + x_4x_{10} + x_4 + x_5x_6x_7x_8x_9x_{10} + \\ & x_5x_6x_7x_8x_9 + x_5x_6x_7x_9x_{10} + x_5x_6x_7x_9 + x_5x_7x_8x_{10} + x_5x_7x_{10} + x_5x_9x_{10} + \\ & x_5 + x_6x_7x_9x_{10} + x_6x_7x_9 + x_6x_7x_{10} + x_6x_7 + x_6x_8x_9x_{10} + x_6x_8x_9 + x_6x_{10}. \end{aligned}$$

* For instance, $\omega = 3$ when uses Gaussian elimination.

The terms of degree 5 and 6 of f have a common divisor x_7x_9. Thus, the polynomials $f \cdot (x_7 - 1)$ and $f \cdot (x_9 - 1)$ have degree 5. Moreover, the terms of degree 4 and 5 in $f \cdot (x_9 - 1)$ have a common factor x_{10}. Thus, the degree of the polynomial $f \cdot (x_9 - 1) \cdot (x_{10} - 1)$ is equal to 4. In [CM03] Courtois and Meier found 14 linearly independent polynomials g of degree 4 such that the polynomial fg also has degree 4.

The attack for Lili-128 proceeds as follows. Since the first 39-bit LFSR is used to clock the second LFSR, we need to guess the internal state of the first LFSR which has 2^{39} possible states. The degree of the polynomials are reduced from degree 6 to degree 4, hence there are $T = \sum_{i=0}^{4} \binom{89}{i}$ possible monomials of degree ≤ 4. For each keystream bit, we obtain 14 linearly independent equations. It is then sufficient to obtain $T/14$ keystream bits to recover the initial state. The time complexity of the attack is then equal to $2^{39} \cdot T^{\omega}$ where ω is the linear algebra constant.

The same attack can also be applied against E0, a stream cipher which is part of the Bluetooth scheme used for wireless communications [AK03, Blu01]. The stream cipher E0 is a non-linear combiner with a 4-bit memory and 4 LFSRs with a total length of 128-bits. Armknecht and Krause showed that given 4 consecutive keystream bits, there exists a polynomial equation of degree 4 [AK03]. The number of polynomials of degree ≤ 4 is equal to $T = \sum_{i=0}^{4} \binom{128}{i}$. In order to minimize the amount of data needed, the attacker must have access to $T + 3$ consecutive keystream bits. Otherwise, the algebraic cryptanalysis on E0 requires access to T 4-bit consecutive keystream bits, which in total requires $4T$ keystream bits. Note that this is different from the previous attack on Toyocrypt and Lili-128 that does not require access to consecutive keystream bits.

In [Cou03] Courtois described a new approach to reduce the complexity of algebraic attacks on stream ciphers called *fast algebraic attack*. The attack model assumes that the attacker is able to obtain consecutive keystream bits. Once obtained, the system of equations involved will have a recursive structure. With such structure, Courtois proposed solving the system of equations using Berlekamp-Massey algorithm, which has a linear complexity instead of Gaussian elimination that has cubic complexity.

2.4 Algebraic Cryptanalysis of Post-Quantum Cryptography

When applied to post-quantum cryptographic (PQC) schemes, i.e., schemes that are designed to withstand quantum adversaries, algebraic cryptanalysis also serves as a valuable analytical tool. PQC schemes includes not only MPKCs, but also code-based, lattice-based, hash-based, and isogeny-based constructions. For an overview of code-based, lattice-based and hash-based schemes, we refer the reader to [BBD09], while the overview on isogeny-based schemes can be found in [JDF11, Feo17].

The primary targets of algebraic cryptanalysis within PQC schemes are naturally the MPKCs. Nonetheless, its applicability extends beyond MPKCs, as it has also proven relevant in the context of code-based and lattice-based schemes. However, its impact tends to be more limited for isogeny-based and hash-based schemes.

Based on the attack objective, the algebraic cryptanalysis against MPKCs can be categorized into two types. The first type, direct attacks, aims at recov-

ering plaintexts for encryption schemes or forging signatures for digital signature schemes. These attacks directly attempt to find a solution to the system of polynomial equations from the public-key of the schemes [FJ03]. The second type, key-recovery attacks, aim to recover the private key of the schemes. This involves solving instances of the MinRank problem (see [BBD09, pg. 224] for a detailed description). In many approaches, a critical step towards solving MinRank instances requires solving systems of polynomial equations, as demonstrated in [FGP$^+$15, AST24, Beu22, Beu24].

In code-based schemes, algebraic cryptanalysis generally models the underlying decoding problem, or closely related structural problems, as systems of polynomial equations. For instance, the algebraic attack in [BBB$^+$20] presents improvements over previous security assumptions. Similarly, the work in [FOPT10] demonstrates that certain attempts to reduce the key size lead to a practical algebraic attack that recovers the private key. In contrast, lattice-based PQC schemes exhibit strong resistance against algebraic attacks. Nonetheless, several works have explored the use of algebraic cryptanalysis to solve the computational problem underpinning lattice-based PQC schemes [AG11, ACF$^+$15, Ste24].

2.5 Solving Systems of Polynomials in Algebraic Cryptanalysis

2.5.1 Gröbner Bases

One of the most widely used methods to solve systems of polynomial equations is using a Gröbner basis algorithm. The theory of Gröbner bases treats a system of polynomial equations as generators for an ideal in the polynomial ring, instead of merely as a set of polynomials. The rationale behind this is that a solution for the system of equations is also a solution for any polynomial in the ideal. Additionally using an appropriate ordering on the monomials, the set of solutions for the system of equations can be parametrized by some polynomials in the ideal.

Gröbner basis algorithms compute a so-called Gröbner basis of the ideal generated by the polynomials in the equation system, with respect to a chosen ordering on the set of monomials. The theory of Gröbner bases was developed by Bruno Buchberger in his PhD thesis [Buc65, Buc06]. One of its immediate application is that Gröbner bases allows us to determine whether a polynomial in the ring is also an element of the ideal. In this regard, Gröbner bases is seen as the multivariate generalization of the greatest common divisor of univariate polynomials.

On the other hand, computing a Gröbner basis for the ideal equipped with the appropriate monomial ordering allows us to recover a set of solutions for the system of polynomial equations that generates the ideal. In this respect, Gröbner bases algorithm is seen as a nonlinear generalization of Gaussian elimination on a set of linear equations. The theory of Gröbner bases together with its applications are discussed further in Chapter 3.

Given an arbitrary basis of a polynomial ideal, there exist algorithms that compute a Gröbner basis of the ideal. Buchberger proposed the first algorithm to compute a Gröbner basis of an ideal in his PhD thesis [Buc65, Buc06]. A major improvement in this field was the F_4 algorithm proposed by Jean-Charles

Faugère [Fau99]. Faugère devised a method to perform reduction on multiple polynomials at once using fast linear algebra. In Chapter 4 we will discuss both algorithms in more detail.

2.5.2 Linearization Based Algorithms

Another technique to solve a system of quadratic polynomial equations that received a lot of attention from cryptography community is linearization-based algorithms. The main idea of linearization-based algorithms is to first interpret each monomial occurring in the system of equations as a new variable. Once linearized and solved, the solutions of the linearized system of equations is then validated against the original system of polynomial equations.

Since the problem of solving an instance of MQ problem is now reduced to the problem of solving a system of linear equations, the efficiency of linearization-based algorithms relies on the number of linearly independent polynomials in the system. For example, in an n-variables polynomial ring over $\mathbb{F}_2$ there are $\binom{n}{2} + n$ monomials of degree less than or equal to 2 (considering $x^2 = x$ and ignoring constant monomial). Thus, if an instance of MQ problem over $\mathbb{F}_2$ in n variables has $\binom{n}{2} + n$ linearly independent equations, then the linearization method works best for this particular case. There are many algorithms that are based on linearization technique [CKPS00, CP02, CP03, Cou04, MP08, CB07, MDBW09, MMDB08, MCD$^+$09]. Most of the variants focus on generating sufficiently many linearly independent equations.

One of the prominent example is *eXtended Linearization* (XL) algorithm, which was introduced in [CKPS00]. XL algorithm takes a system of polynomial equations and a positive integer D as inputs. It generates more polynomials by multiplying a polynomial f in the initial system of equations by all monomials $\mathfrak{m}$ such that the degree of $\mathfrak{m}f$ is less than or equal to D. Once all polynomials of the form $\mathfrak{m}f$ are added to the initial system of equations, the algorithm proceeds by linearizing the system and tries to find solutions to the linearized system of equations. Later XL turned out to be closely related to Gröbner bases algorithms, in particular the F_4 algorithm as shown in [AFI$^+$04]. A more detailed discussion on the XL algorithm and its F_4-like description are given in Section 4.3.

The simplicity of XL eventually stimulates the development of other variants. One of them is *eXtended Sparse Linearization* (XSL) [CP02]. Instead of multiplying a polynomial f in the initial system by all possible monomials of degree less than or equal to $D - 2$, XSL algorithm multiplies f by "carefully selected monomials" [CP02]. The aim is to avoid introducing unnecessary monomials when generating new equations by taking advantage of the sparsity and the structure of the system of equations. However, the authors did not explicitly state how the monomials should be chosen, leaving some rooms for interpretation.

In [MP08], Murphy and Paterson provide a generalization of the XL algorithm. In the paper the authors proved that the XL algorithm is a specialization of an algorithm that finds the intersection of hyperplanes called *GeometricXL*. The main idea of GeometricXL is based on the fact that the formulation of a MQ problem and its solution are invariant under a linear coordinate transformation. This is particularly relevant especially in the case of multivariate public-key cryptosystems where linear changes of coordinates are required in

order to hide the structure of the central map. However, the algorithm requires the order of the underlying finite field to be larger than the maximal degree D.

Another variant of XL called *MutantXL* was introduced in [MMDB08, MCD+09, MDBW09]. The authors proposed a concept of *mutants*. They are the polynomials that have degree strictly less than D after the elimination step. However, it turns out that the concept of mutant is closely related with *normal selection strategy* used in the F_4 algorithm. Thus the MutantXL algorithm can be fundamentally understood as a redundant variant of the F_4 algorithm [ACFP12].

Another linearization based approach is the Crossbred algorithm, proposed by Joux and Vitse [JV17]. It is a hybrid method for solving systems of multivariate polynomial equations over finite fields, particularly $\mathbb{F}_2$, that combines variable partitioning and partial enumeration with XL-like linearization. Given positive integers D, d, k, the algorithm first generates r new polynomials of degree D and degree d in the first k variables, which are then appended to the original system. Then it performs an exhaustive search over all possible values $\mathbb{F}_2^{n-k}$ for the last $n-k$ variables. For each assignment to the last $n-k$ variables, the resulting system in the first k variables is solved. If it has no valid solution then the exhaustive search continues until a valid solution is found [VDI24].

2.5.3 SAT Solving

Recall that the proof of NP-completeness of MQ over $\mathbb{F}_2$ in Proposition 2.4 is based on the reduction of a 3-CNF SAT instance into an instance of multivariate quadratic polynomials over $\mathbb{F}_2$. One may then ask the following question: can we solve an instance of the MQ problem over $\mathbb{F}_2$ by reducing it into an instance of the SAT problem in Conjunctive Normal Form ?

In this subsection we discuss how a problem of solving an instance of MQ problem over $\mathbb{F}_2$ is viewed as a SAT problem. We mention two existing strategies in the literature to convert a system of polynomial equations over $\mathbb{F}_2$ into a SAT problem in CNF.

The first strategy is based on linearization of polynomials and conversion of the resulting linear polynomial into CNF. Since this strategy is proven to be effective for dense polynomial, we refer to this as *dense conversion*. The second conversion strategy is based on the truth table of the polynomials. The method relies on the number of variables in the polynomial being relatively small, i.e., the polynomials must be sparse. Thus we refer to the second strategy as *sparse conversion*. We will now discuss them both in depth.

Dense conversion This idea of converting an instance of MQ over $\mathbb{F}_2$ into a SAT problem in CNF was first proposed by Courtois, Bard, and Jefferson in [BCJ07]. It is also mentioned in Gregory Bard's PhD thesis [Bar07] and in Chapter 13 of [Bar09]. The dense conversion strategy is based on the observation that the derivation of the CNF formula for a linear polynomial over $\mathbb{F}_2$ is straightforward. For instance, the polynomial $x_1 + x_2 + x_3 + x_4$ is represented by the following Boolean formula in CNF

$$\begin{aligned}
&(X_1 \vee X_2 \vee X_3 \vee X_4) \wedge (X_1 \vee X_2 \vee \neg X_3 \vee \neg X_4) \wedge \\
&(X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4) \wedge \\
&(\neg X_1 \vee X_2 \vee X_3 \vee \neg X_4) \wedge (\neg X_1 \vee X_2 \vee \neg X_3 \vee X_4) \wedge \\
&(\neg X_1 \vee \neg X_2 \vee X_3 \vee X_4) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4).
\end{aligned} \tag{2.3}$$

The dense conversion converts a quadratic polynomial over $\mathbb{F}_2$ into a Boolean formula in CNF in three steps.

The first step is to linearize the quadratic polynomial by replacing the quadratic monomials of the form $x_i x_j$ using additional variables $x_{i,j}$.* An equation of the form $x_i x_j + x_{i,j}$ is then added to the system. Once a linear polynomial is constructed, one may be tempted to directly convert it into CNF. This yields a Boolean formula where the number of literals in each clause is equal to the number of terms, say ℓ, in the linear polynomial. At the same time, the number of clauses generated is equal to $2^{\ell-1}$. Thus, this is not a desirable approach since the large number of clauses and literals in each clause increases the complexity of the SAT solving algorithm.

The second step in dense conversion reduces the number of literals in a clause by cutting each sum in the linear equation into subsums of length, say c. We refer to c as the cutting number. The cutting number represents the number of terms in each subsum, which corresponds to the number of literal in a clause once these subsums are converted to CNF. For instance, let $c = 4$ and $\ell \equiv 2 \pmod{c}$. The linear equation $x_1 + \cdots + x_\ell = 0$ is converted into the following system of linear equations

$$\begin{aligned} x_1 + x_2 + x_3 + y_1 &= 0 \\ y_1 + x_4 + x_5 + y_2 &= 0 \\ &\vdots \\ y_i + x_{4i+2} + x_{4i+3} + y_{i+1} &= 0 \\ &\vdots \\ y_{\lceil \ell/c \rceil - 2} + x_{\ell-2} + x_{\ell-1} + x_\ell &= 0. \end{aligned}$$

where $y_1, \ldots, y_{\lceil \ell/c \rceil - 2}$ are additional auxiliary variables. If $\ell \not\equiv 2 \pmod{c}$ then the number of terms in the last equation is less than c. This implies that the number of literals in the clause that corresponds to the last linear equation is also less than c.

The third step, which is the last step, is to convert each cutted linear equation into a Boolean formula in CNF, similar to Equation 2.3. For a linear equations with ℓ number of terms, there are $\lceil \ell/c \rceil - 1$ subsums and each of it corresponds to a set of 2^{c-1} clauses with c literals. For quadratic polynomials of the form $x_i x_j + x_{i,j}$ that represent the substitution of quadratic monomimals $x_i x_j$ by a dummy variable $x_{i,j}$, we can examine its truth table to derive its corresponding Boolean formula in CNF. For instance, the CNF formula for $x_i x_j + x_{i,j}$ is equal to

$$(X_i \vee \neg X_{i,j}) \wedge (X_j \vee \neg X_{i,j}) \wedge (\neg X_i \vee \neg X_j \vee X_{i,j})$$

Sparse conversion The second approach to convert an instance of the MQ problem over $\mathbb{F}_2$ to a Boolean formula in CNF is due to Brickenstein and

* Note that since Boolean formulas in CNF do not have any constant, an additional Boolean variable, say T, must also be introduced to represent the constant term of the polynomial together with a clause (T) (or $(T \vee T \vee \cdots T)$). In this description, we assume that the quadratic polynomial does not have a constant term.

it is described in his PhD thesis [Bri10]. The conversion is essentially truth-table based, thus requiring the number of variables in the polynomial to be small enough such that it is feasible to obtain all solutions of the polynomial. We demonstrate the sparse conversion strategy in the following example.

Example. *Let* $f = x_1x_3 + x_1 + x_2 + x_3 + 1 \in \mathbb{F}_2[x_1, x_2, x_3]$. *We define the following set*

$$S_f = \{(v_1, v_2, v_3) \in \mathbb{F}_2^3 : f(v_1, v_2, v_3) = 1\}.$$

For each vector $(v_1, v_2, v_3) \in S_f$ *we construct a clause that eliminates the corresponding assignment to be a solution for the Boolean formula in CNF. In this example we have*

$$S_f = \{(0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 1, 1)\}.$$

Therefore, the corresponding Boolean formula for f *in CNF is*

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3).$$

Note that the CNF representation of f *above with sparse conversion is not unique. For instance, we can further simplify the CNF formula of* f *as follows*

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2).$$

The two approaches above can be seen as a companion to one another. Therefore, a better algorithm to convert an instance of the MQ problem over $\mathbb{F}_2$ into a SAT problem in CNF combines both techniques. If it encountered a sparse polynomial in the system, the sparse conversion technique is employed to obtain a more compact CNF. Otherwise the polynomial is considered dense and the algorithm then uses dense conversion.

Once constructed, one can feed the CNF into an off-the-shelf SAT solver to obtain a solution. Recall that the SAT problem is a well-known NP-complete problem. This means that the worst-case time complexity is expected to be exponential. Nevertheless, there are many good implementations of SAT solver available such as MINISAT [ES03], CRYPTOMINISAT [SNC09], PLINGELING [Bie17]. In particular, CRYPTOMINISAT is designed to efficiently work on cryptographic related problems. More importantly, it provides an extension to the input language by supporting XOR clauses.

Note that the algorithms to solve instances of the SAT problem in CNF are randomized, implying that a careful approach has to be taken to estimate their average case runtime to solve a particular instance.

2.5.4 Raddum-Semaev Algorithm

In [RS06], Raddum and Semaev introduced another approach to solve a system of equations over $\mathbb{F}_2$. The approach works for sparse systems of equations such that the number of variables that appear in each polynomial is relatively small. We refer to this approach as the Raddum-Semaev algorithm.

Let $f_1, \ldots, f_m$ be polynomials with coefficients in $\mathbb{F}_2$ in variables $X = (x_1, \ldots, x_n)$. Let $\mathsf{Var}(f_i) \subseteq X$ be an ordered set of the variables that appear in

f_i. Assume that for all $i = 1, \ldots, m$, we have $|\mathsf{Var}(f_i)| \leq k$ for some positive integer k such that 2^k is relatively small. Raddum-Semaev algorithm does not treat each polynomial as an element of the ring $\mathbb{F}_2[x_1, \ldots, x_n]$, but simply as an equation restricted to variables that appear in each polynomial.

Definition 2.6. *A configuration of a polynomial f_i is an assignment of values to the variables in $\mathsf{Var}(f_i)$ such that $f_i = 0$.*

Note that the elements in $\mathsf{Var}(f_i)$ are ordered and a configuration of f_i is represented by a bit string of length $|\mathsf{Var}(f_i)|$. The j-th bit of a configuration then corresponds to the value for the j-th variable in $\mathsf{Var}(f_i)$. This allows each polynomial f_i to be identified with $\mathsf{Var}(f_i)$ and a set of configurations of f_i.

Definition 2.7. *A symbol $S_i = (\mathsf{Var}(f_i), L_i)$ of the polynomial f_i consists of $\mathsf{Var}(f_i)$ and a set L_i of configurations of f_i.*

Since the cardinality of $\mathsf{Var}(f_i)$ is small, it is possible to construct L_i by exhaustively running through all bit strings of length $|\mathsf{Var}(f_i)|$ and checking whether it satisfies as a configuration of f_i.

Every solution $s = (s_1, \ldots, s_n) \in \mathbb{F}_2^n$ of the system of equations $f_1, \ldots, f_m$ also satisfies each polynomial f_i for all $i = 1, \ldots, m$. This implies that s restricted to the variables in $\mathsf{Var}(f_i)$ is a configuration of f_i, i.e., it is an element of L_i.

The idea of the Raddum-Semaev algorithm is to repeatedly remove configurations that contradict a solution for the system of equations. Once all pathological configurations are removed, the values of the remaining configurations constitute as candidate solutions of the system.

Definition 2.8. *Let $\mathsf{Var}(f_i) \subseteq \mathsf{Var}(f_j)$ for $i, j \in \{1, \ldots, m\}$. A configuration of f_j* covers *a configuration of f_i if the two are equal for all variables in $\mathsf{Var}(f_i)$.*

Definition 2.9. *Let S_i, S_j be the two symbols of the polynomials f_i, f_j, respectively, where $f_i \neq f_j$. Let $f_{i,j}$ be a polynomial with variables*

$$\mathsf{Var}(f_{i,j}) = \mathsf{Var}(f_i) \cap \mathsf{Var}(f_j)$$

and the symbol $S_{i,j}$ of $f_{i,j}$ is defined as

$$S_{i,j} = \left(\mathsf{Var}(f_{i,j}), \mathbb{F}_2^{|\mathsf{Var}(f_{i,j})|}\right).$$

Let $L_{i,j}, L_{j,i} \subseteq \mathbb{F}_2^{|\mathsf{Var}(f_{i,j})|}$ be two sets of all configurations of $f_{i,j}$ that are covered by at least one configuration of f_i, f_j, respectively. The two symbols S_i and S_j agree if and only if $L_{i,j} = L_{j,i}$.

When two distinct symbols S_i and S_j do not agree, all configurations in both L_i and L_j that do not cover any configuration of $L_{i,j} \cap L_{j,i}$ are eliminated to obtain an updated set of configurations $L_i' \subseteq L_i, L_j' \subseteq L_j$. The configurations removed from L_i and L_j are indeed the wrong ones because they can not satisfy both f_i and f_j. The set of configurations in $S_i = (\mathsf{Var}(f_i), L_i'), S_j = (\mathsf{Var}(f_j), L_j')$ are updated by L_i', L_j' respectively and both symbol now agree. The step above is referred to as the *agreeing procedure*.

The procedure above is used inside the *Agreeing algorithm*. The algorithm iteratively searches for two distinct polynomials f_i, f_j such that their respective

symbols S_i, S_j disagree and apply the *agreeing procedure* on S_i and S_j. Let f_k be another polynomial such that $\mathsf{Var}(f_j) \cap \mathsf{Var}(f_k)$ is nonempty. Suppose that S_j and S_k disagree when S_i and S_j agree. Applying *agreeing procedure* on S_j, S_k may cause S_i and S_j to disagree. This means that running *agreeing procedure* on one pair can cause other pairs to disagree.

Once all pairs of symbols agree, it is possible that the correct solution is not yet recoverable from each set of configurations. In this case, there are two methods proposed in [RS06] so that its possible to re-start the *Agreeing algorithm*. The two methods are *Splitting* and *Gluing*.

Splitting After all possible pairs of symbols are already in agreeing state without leading to an obvious solution for the system of equations, the splitting strategy chooses a particular set of configurations L_i of a symbol S_i and splits L_i into two parts, $L_i^{(1)}$ and $L_i^{(2)}$. Assuming there is an unique solution, then this is either contained in configuration $L_i^{(i)}$ or in configuration $L_i^{(2)}$. Replacing L_i by $L_i^{(1)}$ or $L_i^{(2)}$ and restarting the *Agreeing algorithm*, the splitting strategy essentially tries to guess which subset matches the unique solution. If a particular choice of subset does not contain the correct configuration, then we proceed by running the *Agreeing algorithm* with the other subset.

It is often the case that performing splitting once is still insufficient to recover a solution for the system of equations, i.e., all distinct pairs of symbols end up in agreeing state again without an obvious solution. What Raddum-Semaev algorithm does to handle this situation is to perform another guess followed by the *Agreeing algorithm* again.

When a wrong subset is chosen from L_i, the *Agreeing algorithm* eventually removes the correct configuration from a set of configurations of a symbol. This means that the *Agreeing algorithm* eliminates all configurations in some symbol S_j. When this event happens, the algorithm backtracks to the last guess made and tries the other subset of configurations.

The splitting strategy can thus be viewed as the traversal of a binary search tree. The leave nodes represent a situation when a solution is found or a particular set of configurations L_i is empty. Note that the depth of the leave nodes varies and the complexity of this approach is exponential in the average of their depth.

Gluing This method can be viewed as the opposite of splitting. It merges two sets of configurations into one, essentially combining two symbols. Let $X_{i,j} = \mathsf{Var}(f_i) \cap \mathsf{Var}(f_j)$ and $Y = \mathsf{Var}(f_i) \cup \mathsf{Var}(f_j)$ where f_i and f_j are two distinct polynomials. We define a set L of configurations for Y that consists of all configurations (p, o, q) such that o is a configuration for $X_{i,j}$, $(p, o) \in L_i$, and $(o, q) \in L_j$. Equivalently, every configuration in L simultaneously covers one configuration of f_i and one configuration of f_j.

Let $S = (Y, L)$ be the symbol constructed after gluing symbols S_i and S_j. The symbols S_i and S_j can then be removed since their respective set of configurations are already contained in L. We can then start the *Agreeing algorithm* again.

The cost of gluing two symbols is that the cardinality of the set of merged configurations is larger. Assume that S_i and S_j are already in agreeing

state. The new set of configurations L of S can have cardinality as small as $\max\{|L_i|, |L_j|\}$ and it can be as large as $|L_i| \cdot |L_j|$.

This approach was generalized later by the same authors in [RS08]. The generalization does not consider multivariate polynomials over $\mathbb{F}_2$ and their configurations but it treats each polynomial as multiple right hand side linear (MRHS) equations.

2.5.5 Mixed Integer-Linear Programming

Another approach to solve a system of polynomial equations over $\mathbb{F}_2$ is by expressing it as a mixed integer-linear programming (MILP) problem. This technique was proposed in [BKS09] to solve polynomial equations of Bivium [Rad06], a small-scale variant of the stream cipher Trivium [Can06].

Definition 2.10 (Mixed-Integer Linear Programming Problem)**.** *Let $k, \ell \in \mathbb{Z}_{\geq 0}$ and set $n = k + \ell$. A Mixed-Integer Linear Programming (MILP) problem is defined as determining*

$$\min_{\mathbf{x}=(\mathbf{x}_1,\ldots,\mathbf{x}_n)} \{c \cdot \mathbf{x} : A\mathbf{x} \leq b, \mathbf{x} \in \mathbb{Z}^k \times \mathbb{R}^\ell\}$$

given a vector $c = (c_1, \ldots, c_n)$, an $m \times n$ matrix A, and a vector $b = (b_1, \ldots, b_m)$. Equivalently, it is a problem of minimizing the linear equation $\sum_{i=1}^{n} c_i \mathbf{x}_i$ subject to the linear inequality constraints defined by A and b. An element $\mathbf{x} \in \mathbb{Z}^k \times \mathbb{R}^\ell$ that satisfies $A\mathbf{x} \leq b$ is called a feasible point. The set of all feasible points is called the feasible set.

In [BKS09], the authors mentioned three (3) techniques to convert a system of multivariate polynomial equations over $\mathbb{F}_2$ to a set of (non-)linear (in-)equality constraints.

Standard Conversion Method. This method is a straightforward approach to express a polynomial over $\mathbb{F}_2$ into equalities in $\mathbb{R}$. It converts binary operations in $\mathbb{F}_2$ in the following way

$$x \cdot y \mapsto \mathbf{x} \cdot \mathbf{y}$$
$$x + y \mapsto \mathbf{x} + \mathbf{y} - 2\mathbf{xy}$$

where x, y and $\mathbf{x}, \mathbf{y}$ are variables over $\mathbb{F}_2$ and $\mathbb{R}$ respectively.

Adapted Standard Conversion (ASC) Method. This second approach takes a polynomial over $\mathbb{F}_2$ and directly converts it into a set of (in)equalities, instead of considering each binary operation in the polynomial. The addition and multiplication in $\mathbb{F}_2$ are converted into addition and multiplication over $\mathbb{R}$. The polynomial over $\mathbb{R}$ is evaluated for all values that holds for the polynomial over $\mathbb{F}_2$. Each of these evaluations yields a new equation over $\mathbb{R}$ by substracting the evaluation result from the left-hand side of the polynomial. It implies that only one of these new polynomials over $\mathbb{R}$ can hold and this is expressed by multiplying each equation with an exclusion variable, followed by adding a new constraint that the sum of all exclusion variables is equal to 1. The following example demonstrates the above steps.

Example 2.11. *Consider the following equation over* $\mathbb{F}_2$ *in* 6 *variables*

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1 + x_2 + x_3x_4 + x_5 + x_6. \tag{2.4}$$

Let $\mathbf{f}$ *be the corresponding polynomial of* f *over* $\mathbb{R}$*. By evaluating* $\mathbf{f}$ *at all solutions of* f*, we get* $0, 2, 4$ *as results. This implies that one of the following linear equations over* $\mathbb{R}$ *is true*

$$\begin{aligned}
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 &= 0,\\
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2 &= 0,\\
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 4 &= 0.
\end{aligned}$$

Three exclusion variables $\mathbf{x}_{e_1}, \mathbf{x}_{e_2}, \mathbf{x}_{e_3}$ *are then introduced for each equation above.*

$$\begin{aligned}
\mathbf{x}_{e_1}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6) &= 0,\\
\mathbf{x}_{e_2}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2) &= 0,\\
\mathbf{x}_{e_3}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 4) &= 0,\\
\mathbf{x}_{e_1} + \mathbf{x}_{e_2} + \mathbf{x}_{e_3} &= 1.
\end{aligned}$$

Integer Adapted Standard Conversion (IASC) Method. This conversion technique is an improvement over the ASC method. It follows similar steps as in ASC, except that there is no new equation generated over $\mathbb{R}$. A critical observation used for this method is that when a polynomial over $\mathbb{R}$ is evaluated for all solutions of its corresponding Boolean polynomial, the results are multiples of 2. Instead of generating new equations and adding exclusion variables, this method introduces only one new integer-valued variable. The following example illustrates the benefit of IASC method over ASC.

Example 2.12. *We use the same polynomial as in* (2.4)*. Recall that the results of evaluating* $\mathbf{f}$ *at all solutions of* f *is* $0, 2, 4$*. Thus, a solution of* (2.4) *is a solution to the following equation*

$$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2y = 0.$$

where $y \in \{0, 1, 2\}$*. The advantage of IASC over ASC is obvious: the degree of the converted polynomial is equal to the original polynomial and the conversion introduces only one new monomial.*

Note that IASC conversion is only applicable if the newly introduced variable can be restricted to an integer value.

The above techniques to generate equalities from an instance of MQ problem over $\mathbb{F}_2$ do not automatically produce an instance of MILP problem, since monomials of degree ≥ 2 still appear in the set of equalities. The final step of the conversion resolves this situation by linearizing the equalities. It substitutes monomials of the form $\mathbf{x}_i\mathbf{x}_j$ by auxiliary variables $\mathbf{y}_{i,j}$. The value $\mathbf{y}_{i,j}$ is equal to zero whenever $\mathbf{x}_i = 0$ or $\mathbf{x}_j = 0$ and this is expressed by the following inequalities

$$\mathbf{y}_{i,j} \leq \mathbf{x}_i, \qquad \mathbf{y}_{i,j} \leq \mathbf{x}_j.$$

The other case where $\mathbf{y}_{i,j} = 1$ if and only if $\mathbf{x}_i = 1$ and $\mathbf{x}_j = 1$ is ensured by the following inequality

$$\mathbf{x}_i + \mathbf{x}_j - 1 \leq \mathbf{y}_{i,j}.$$

In practice, the reduction of a MQ problem over $\mathbb{F}_2$ to a MILP problem combines all three conversion techniques [BKS09]. Once constructed, an off-the-shelf MILP solver such as Gurobi [Gur16], GLPK [Mak18] or SCIP [GEG+17] is used to obtain a feasible solution for the set of linear constraints.

2.5.6 Fast Exhaustive Search

If an instance of an MQ problem in n variables and m equations is defined over a small finite field, a possible approach to find a solution is to perform exhaustive search. When evaluating the system of equations on a possible solution, one can employ an early abort technique that terminates the evaluation after an equation does not evaluate to zero [Nin17].

An improved exhaustive search technique leveraging partial derivatives and Gray code enumeration was proposed in [BCC+10]. The algorithm iterates through all possible solutions in $\mathbb{F}_2^n$ with $\mathcal{O}(\log_2 n \cdot 2^{n+2})$ elementary bit operations. The main idea is that the value of a function f evaluated at $a \in \mathbb{F}_2^n$ can be computed from the value of f and its derivative $D_{e_i} f(x) = f(x + e_i) + f(x)$ evaluated at another point $a' = a + e_i$ where e_i is the i-th canonical basis of $\mathbb{F}_2^n$, that is

$$f(a) = f(a') + D_{e_i} f(a').$$

This can then be applied recursively on the first order derivative. When applying it to a quadratic equation, the first order derivative becomes linear, and the second order derivative becomes constant. The latter serves as the base case for the recursion.

Gray code enumeration ensures that two consecutive elements in $\mathbb{F}_2^n$ differ by only one bit, allowing efficient incremental evaluation. A key optimization in fast-exhaustive search is that not all equations need to be evaluated simultaneously. Instead, only a subset of equations are evaluated with Gray code enumeration and the rest are evaluated using the naive approach when a solution candidate is found. Consequently, the complexity of the algorithm is independent of the number of equations. The implementation in [BCC+10] evaluates a subset of 32 equations with Gray code enumeration in order to utilize the 32-bit registers provided by the hardware.

2.5.7 Other Algorithms

In addition to the methods discussed earlier, several other algorithms exist for solving systems of polynomial equations over finite fields. Among these are probabilistic algorithms for systems over $\mathbb{F}_2$, such as those proposed by Lokshtanov et al. [LPT+17] and Björklund et al [BKW19]. A common feature of both algorithms is that they asymptotically outperform exhaustive search algorithms in the worst case and their complexity does not depend on any assumption about the original system of equations. Other approaches combine partial exhaustive search with algebraic solving procedure such as BooleanSolve [BFSS13], FXL [CKPS00], and Hybrid F_4/F_5 [BFP09]. Moreover, several algorithms have been specifically designed to solve underdefined system of equations [KPG99, MHT13, CGMT02], where the number of variables exceeds the

number of equations. For a comprehensive overview of existing techniques for solving systems of equations over finite fields, we refer the reader to the survey by Bellini et al. [BMSV22] which provides a detailed discussion of their computational complexity.