Determining Whether a Given Cryptographic Function is a Permutation of Another Given Cryptographic Function—a Problem in Intellectual Property

Gregory V. Bard*

October 20, 2018

Abstract

Imagine that, in order to avoid patent fees, licensing agreements, or export restrictions, someone permutes the plaintext bits, ciphertext bits, or key bits of a block cipher. All security properties of the block cipher would be preserved. There are many possible such permutations (e.g. $2^{3116.32}$ for the Advanced Encryption Standard, AES-256). It might seem infeasible to detect this fraud, and even harder to determine the permutation matrices used. Instead of a block cipher, it could be the compression function of a cryptographic hash, or any other cryptographic function.

This paper presents an algorithm whereby this fraud could be easily detected, by means of a SAT-Solver—a standard off-the-shelf software package that solves small-to-medium sized instances of the logical satisfiability problem. This paper also presents how this problem can be modeled in a system of polynomial equations (e.g. in the context of algebraic cryptanalysis). Moreover, this problem is related to the "isomorphism of polynomials" problem and that connection is explored at length.

Keywords: Algebraic Cryptanalysis, Block ciphers, Circuit Equivalence, Intellectual Property Fraud Detection, Isomorphism of Polynomials, Logical Satisfiability Solvers (SAT-Solvers), Polynomials mod 2.

1 Definitions

Let $\{0,1\}^\ell$ represent the set of all binary strings of length ℓ. We will now endeavor to define block ciphers as a mathematical object and make explicit some of their more relevant properties. Formally, a block cipher is a function

$$E : K \times P \to C$$

with several important properties. The key k is chosen from K, the keyspace; the plaintext message is chosen by the sender as some $p \in P$. The ciphertext, computed by the cipher, is some $c \in C$.

For practical implementation purposes, nearly all ciphers in use during the last 40 years have had all three sets K, P, and C be the set of binary strings of some fixed length. (This enables digital circuitry to be used for the cipher.) For example, the small and outdated cipher called the Data Encryption Standard (DES) uses $K = \{0,1\}^{56}$ but $P = C = \{0,1\}^{64}$ [34, Ch. 4]. The Advanced Encryption Standard (AES) uses $P = C = \{0,1\}^{128}$ but the user can choose $K = \{0,1\}^{128}$, $K = \{0,1\}^{192}$, or $K = \{0,1\}^{256}$ [34, Ch. 5]. Beyond these trivial details, there are five crucial security requirements for any block cipher.

First, it must be fast to encrypt—i.e. one can compute $E(k,p) = c$ rapidly when $k \in K$ and $p \in P$ are known. Second, it must be fast to decrypt—i.e. one can find p such that $E(k,p) = c$ when $k \in K$ and $c \in C$ are known. Third, it must be computationally infeasible to cryptanalyze—i.e. it should be infeasible to compute k such that $E(k,p) = c$ when $p \in P$ and $c \in C$ are known. (This is called a "chosen-plaintext

*Dept of Math., Stats., and Comp. Sci., The University of Wisconsin—Stout, Menomonie, Wi, 54751.

attack.") There is a consensus in cryptography that "computationally infeasible" means that the fastest known algorithm should be no faster than checking all possible values of K. The set K is chosen to be extremely large—as a necessary but insufficient condition for security.

For the fourth and fifth properties, consider that for any fixed k, we can rewrite $E(k, p) = c$ as $E_k(p) = c$, and then E_k becomes a function $P \to C$ for any fixed $k \in K$. A property called unique decodability requires that E_k be injective for all $k \in K$. Otherwise, if there existed $k_s \in K$, $p_1 \in P$, and $p_2 \in P$ with $p_1 \neq p_2$ but $E(k_s, p_1) = E(k_s, p_2) = c_s$ then a receiver who receives the ciphertext c_s, and who is using key k_s, will have no way of determining if the sender had intended to transmit the message p_1 or the message p_2. It is almost always the case that $C = P$ as sets in practice, therefore textbooks usually say that E_k must be bijective—however, that is not strictly necessary. If $|C| > |P|$ then E_k can be injective without being surjective or bijective.[1]

The fifth and most important criterion is that for some $k \in K$ selected uniformly at random, it should be the case that the E_k obtained is computationally indistinguishable from a random function chosen uniformly from the set of all possible injective functions that map $P \to C$. Formal cryptography textbooks model block ciphers by a mathematical object called a pseudorandom permutation, for this reason. For more information and additional formal definitions, see [19, Ch. 5].

1.1 A Note about Polynomial Time

Readers familiar with theoretical cryptography might be surprised to see that the phrase "in polynomial time" is missing above. When we say that Gaussian Elimination runs in polynomial time, we mean that if one were to take a sequence of problems, each an $n \times n$ matrix, and measure the running time (e.g. by counting the number of arithmetic operations), then one can upper-bound this running time as a function of n, by some polynomial of n. In this case, the polynomial can be cubic.

However, when working with block ciphers, this notion of polynomial time is unavailable. Any particular hardware cipher has a fixed K, a fixed P, and a fixed C. There is no limit as n goes to infinity, because n is not going to infinity—all sizes are constant.

1.2 Permutations of a Block Cipher

For maximum generality, let $K = \{0,1\}^{\ell_k}$, $P = \{0,1\}^{\ell_p}$, and $C = \{0,1\}^{\ell_c}$. As mentioned earlier, in practice it is almost always the case that $\ell_p = \ell_c$.

With some thought, one can see that none of the requirements of a block cipher will change after applying three permutations, represented by the permutation matrices M_1, M_2, and M_3, in the following sense:

$$\hat{E}(K, P) = M_3 E(M_1 K,\ M_2 P)$$

where M_1 is an $\ell_k \times \ell_k$ permutation matrix, M_2 is an $\ell_p \times \ell_p$ permutation matrix, and M_3 is an $\ell_c \times \ell_c$ permutation matrix.

In other words, if E meets all the criteria of a block cipher, then a software pirate can fix three particular permutation matrices, and construct $\hat{E}$, which will share all of the security properties of E. However, this will not be readily apparent externally (especially if implemented in hardware, but even if implemented with software in the case where code-obfuscation tools have been used). Therefore, the software pirate can evade patent fees, licensing agreements, or cipher-export restrictions.

It is worth noting that the number of permutations is rather large. For the outdated DES cipher, there would be 56! choices for M_1, 64! choices for M_2, and 64! choices for M_3. This comes to

$$(56!)(64!)(64!) \approx 2^{840.643} \approx 10^{253.059}$$

[1] Though not relevant to this paper, one could consider the following example of $|C| > |P|$. The plaintexts are 512-bits long, and are encrypted using the old Data Encryption Standard (DES) in Cipher Block Chaining mode (CBC mode), with random initialization vectors. The plaintext will be divided into 8 blocks of 64 bits each, and there will be 9 blocks of 64-bit ciphertext—the first being the initialization vector and the others being the outputs of DES. One could consider this application as a block cipher with $K = \{0,1\}^{56}$, $P = \{0,1\}^{512}$, and $C = \{0,1\}^{576}$.

possibilities, and the number would be considerably larger for modern ciphers like the AES. The largest key setting permitted for the AES is 256 bits, thus $K = \{0,1\}^{256}$, so there would be

$$(256!)(128!)(128!) \approx 2^{3116.32} \approx 10^{938.106}$$

One could sympathize with anyone who would assume that this ruse would be undetectable. The problem of detecting this type of fraud was posed to the author of this paper in 2009 during a coffee break at the conference CHES (Cryptographic Hardware and Embedded Systems). Regrettably, the author does not remember the name of the US Department of Defense employee who proposed this interesting problem—and the proposer did wish to remain anonymous.

This paper shows a quick and efficient method of determining, when presented with two block ciphers F and G, if one is a permutation of the other, by use of a SAT-Solver. Another form of the solution is presented in terms of a polynomial system of equations, similar to algebraic cryptanalysis [5]. Moreover, it explicitly computes M_1, M_2, and M_3, though very indirectly. The solution is given in Section 3.

2 Background

The following background material will be useful in the later sections. Some readers will already be familiar with some subsections, and can therefore skip accordingly.

2.1 Background: Digital Circuits and Polynomials mod 2

The relationship between digital circuits and polynomials over $\{0,1\}$, the finite field with two elements, arises from the fact that a combinatorial digital circuit is a collection of logic gates, and each logic gate is a simple polynomial mod 2.

There, because the sum of two polynomials is a polynomial, the product of two polynomials is a polynomial, and the composition of two polynomials is a polynomial, the entire combinatorial circuit (with n inputs and 1 output) is a polynomial in n variables. Here are some classic examples:

Digital Gate	Logical Operation	Polynomial
x AND y	Conjunction	xy
x OR y	Disjunction	$x + y + xy$
NOT x	Negation	$1 + x$
x XOR y	Distinction	$x + y$
x NAND y	Incompatibility	$1 + xy$
x XNOR y	Equivalence	$1 + x + y$
IF x THEN y ELSE z	Switching	$z + xy + xz$
MAJ(x,y,z)	Majority Vote	$xz + xy + yz$

A circuit with n inputs and m outputs becomes a polynomial system of m equations in n variables, with one polynomial for each output. Therefore, converting a digital circuit to its polynomial system is often straightforward, and results in one equation for each output bit, and one variable for each input bit. For more details, see the classic textbook [7].

For example, the block cipher Keeloq is converted into polynomials mod 2 in [5, Ch 2]. The key fact for this paper is that, for any cryptographic digital circuit, the equations for polynomials mod 2, as well as the clauses for a SAT-solver, will be fairly easily obtained.

2.2 Background: The Isomorphism of Polynomials Problem

The problem posed in this paper is related to the isomorphism of polynomials problem. Imagine a set of m polynomial functions, $f_i(x_1, x_2, x_3, \ldots, x_n)$. The x_is as well as the coefficients are from some finite field F. Each f_i can be thought of, individually, as a function $F^n \to F$. Using vector notation, one can aggregate the inputs and write instead $f_i(\vec{x})$. The system can also be thought of as one function, $F^n \to F^m$ in the sense

that for any $\vec{x} \in F^n$ (the domain) one can define a $\vec{y} \in F^m$ such that $y_i = f_i(\vec{x})$. More simply, a copy of $\vec{x}$ is given to each f_i, which returns a single field element from F, and that becomes y_i. The y_is are assembled into a vector $\vec{y} \in F^m$.

Consider two such systems, $f_1, f_2, \ldots, f_m$ and $g_1, g_2, \ldots, g_m$. We can define the systems as isomorphic if and only if there exist an invertible $m \times m$ matrix M_1 and an invertible $n \times n$ matrix M_2 such that

$$f(\vec{x}) = M_1 g(M_2 \vec{x})$$

The "isomorphism of polynomials" problem, as posed by Patarin in [25], is the task of finding M_1 and M_2 given the two polynomial systems, f and g. If one further drops the requirement that M_1 and M_2 are invertible then it is the "morphism of polynomials problem" which is proven NP-hard in [20] and [26]. (Sometimes the morphism of polynomials problem is called "PolyProj" [8].) In contrast, the "isomorphism of polynomials" problem has been solved in many cases. With this in mind, it is useful to compare this paper's problem to the isomorphism of polynomials problem.

Despite the structure being largely analogous, some differences are apparent. First, the problem in this paper has three matrices, and the isomorphism of polynomials problem has only two. Second, the matrices in this paper have to be permutation matrices, whereas in the isomorphism of polynomials problem, the matrices merely must be invertible. Third, as it turns out, most of the applications of the isomorphisms of polynomials work over finite fields F of positive odd characteristic, whereas this paper works in the finite field $\{0, 1\}$. Fourth, the solution here is extremely rapid and somewhat straightforward, whereas the isomorphism of polynomials, where it has been solved, has in some cases been solved in time quartic to the number of variables, neglecting logarithmic factors. Fifth, most research focuses on polynomials of 2nd or 3rd degree for the isomorphism of polynomials, but this paper has no such restriction. (In their recent preprint [29], Plût, Fouque, and Macario-Rat mention that they are not aware of any progress on solving the cubic and higher-degree cases.)

The special case where the inputs are permuted, but the outputs are not permuted (M_1 is the identity matrix) is sometimes denoted IP1S, for "isomorphism of polynomials with one secret." That case was addressed in [27]. Contrastingly, if M_1 is not the identity matrix, this can be denoted by IP2S, for "isomorphism of polynomials with two secrets."

We are not the first to consider M_1 and M_2 to be permutation matrices, as was done by Patarin, Goubin, and Courtois in [26]. In that paper, they prove that the permutation-matrix case is reducible to graph isomorphism, and conjecture that it is much easier than the non-singular-matrix case. In fact, the cryptosystems proposed in [25] were proposed as a response to the graph-isomorphism based examples in [18].

It was shown by Agrawal and Saxena [4] [30] that graph isomorphism is reducible to the isomorphism of polynomials problem, producing polynomials of cubic degree. Of course, this means that the isomorphism of polynomials problem is as hard as graph isomorphism. Moreover, any algorithms or heuristics that solve the isomorphism of polynomials problem can also solve graph isomorphism problems. The cases used to solve graph isomorphism have M_1 equal to the identity matrix. The $m = 1$ case has been thoroughly studied by Kayal in [20].

Several cryptographic schemes have been built on the hardness of the isomorphism of polynomials problem:

- A zero-knowledge identification scheme of Patarin [25] (M_1 is the identity matrix)
- A proxy-signature scheme of Tang and Xu [35] [36] (M_1 is the identity matrix)
- A group-signature scheme of Yang, Tang, and Yang [37] (M_1 is the identity matrix)
- Cryptosystems based on the Hidden Field Equations (HFE) property [25]
- Multivariate public-key encryption/signature schemes [24] [25] [38]

Unfortunately, these schemes were broken in many cases, for example [16], [17], and [26]. The paper [8] lists many other applications of the isomorphism of polynomials with extensive citations, including deciding VP vs VNP, geometric complexity theory, the equivalence of the determinant of a matrix versus the permanent of a matrix (both taken as polynomials), the minimum complexity of matrix multiplication, and computing the symmetric rank of a symmetric tensor.

2.3 Background: SAT-Solvers

The logical satisfiability problem is very simple. Given a logical sentence in many variables, can one find an assignment of 0 or 1 to each variable so that the entire sentence comes out to be true. This problem is the central problem of NP-completeness, in the following sense. To prove that some problem in NP is NP-complete, one must show a reduction to an already known NP-complete problem. Thus every NP-complete problem is reducible to another, reducible to another, until arriving at the logical satisfiability problem.

In practice, however, these reductions can be used to actually solve problems. SAT-solvers are tools which solve the logical satisfiability problem in small to medium sized problems, in a reasonable amount of time. There is an annual SAT-solver[2] competition, with prizes. For this reason, SAT-solvers have become very good over the last 25 years. MiniSAT [2] is a small and freely available SAT-solver that has won several competitions.

In summary, a SAT-Solver is a software tool which, given a logical sentence written in the predicate calculus, tries to find an assignment of true and false to each variable to make the entire sentence come out true. By "predicate calculus," we mean that the logical sentence can use the operators `and`, `or`, `not`, `xor`, `equals`, and `implies` but not quantifiers such as $\forall$ and $\exists$. However, in practice, SAT-solvers take their input in a format called conjunctive normal form (to be defined momentarily).

While the logical satisfiability problem is theoretically NP-complete, SAT-solvers tend to be very fast on problems arising from applications. It should be noted that NP-Completeness is about problems where n grows to infinity. If n is fixed and "sufficiently" small, many NP-Complete problems are easily solved in practice by heuristics, approximation algorithms, or specialized tools. Sadly, "sufficiently small" can be very difficult to define for many categories of problems.

When presented with a polynomial system of equations mod 2, one can convert to circuit satisfiability and use a SAT-Solver [9], such as MiniSAT [2]. This was a major part of the author's PhD dissertation [6, Ch. 3–4] and monograph [5, Ch. 13–15]. Often, researchers will think in the world of equations mod 2, and convert to logical satisfiability at the last possible moment, before feeding the problem into a computer. This is an alternative to using a computer algebra system to solve the polynomial system of equations, such as Sage [3] or MAGMA [1], which in turn use Faugere's F4 [15] algorithm and its many descendants.

The format of a logical satisfiability problem is called Conjunctive Normal Form, or CNF. There are a large number of clauses, and each clause must be true for the entire problem to be considered satisfied. Each clause is a disjunction (logical OR) of several variables, each which might or might not be negated. For example, a clause might look like

$$(x_{25} \vee \overline{y_{31}} \vee \overline{z_{17}} \vee x_{42})$$

where the overline represents negation. See [5, Ch 13] for further details, if desired.

3 The Solution

We will now solve the intellectual property problem. Namely, presented with two ciphers F and G, we wish to determine if G is a copy of F merely with the plaintext bits, ciphertext bits, and key bits permuted. In the case of such copying, we also wish to determine the exact permutations used. For clarity, it is useful to first consider a special case: a block cipher with 8-bits of plaintext, 8-bits of ciphertext, and an 8-bit key. Of course, this is far too small for realistic usage, but it is merely a pedagogical stepping stone toward the final solution.

3.1 A Useful Polynomial

Let us consider the following polynomial μ, in the finite field of two elements. There are eleven inputs, three of which (i_2, i_1, i_0) can be thought of as an integer $0 \leq x < 8$. Then the polynomial will output s_x, choosing the xth of the remaining eight inputs $s_0, s_1, s_2, \ldots, s_7$. In the notation of polynomials, we shall write

[2]Held once during most, but not all, years. See `http://www.satcompetition.org/` for details.

$$\begin{aligned}\mu(i_2,i_1,i_0,s_0,s_1,s_2,\ldots,s_7) &= (1+i_2)(1+i_1)(1+i_0)s_0 + (1+i_2)(1+i_1)(i_0)s_1 + \\ &\quad (1+i_2)(i_1)(1+i_0)s_2 + (1+i_2)(i_1)(i_0)s_3 + \\ &\quad (i_2)(1+i_1)(1+i_0)s_4 + (i_2)(1+i_1)(i_0)s_5 + \\ &\quad (i_2)(i_1)(1+i_0)s_6 + (i_2)(i_1)(i_0)s_7\end{aligned}$$

We can summarize this useful property by

$$\mu(i_2,i_1,i_0,s_0,s_1,s_2,\ldots,s_7) = s_x \text{ where } x = 4i_2 + 2i_1 + i_0$$

and that, in turn, follows from the fact that this function will always have seven of its eight terms "zeroed-out" by i_2, i_1, i_0 regardless of the values that those three variables take. The remaining term is the correct s_i. For example, if $i_2 = 1$, $i_1 = 1$, and $i_0 = 0$, then μ will output s_6, because all the other seven terms have a coefficient of 0 for the s_is, and there is a 1 for the coefficient of s_6.

This boolean polynomial comes from computer engineering. The digital circuit given by this boolean polynomial is called "an 8-way multiplexer" by computer engineers.

In the CNF notation of SAT-Solvers, we would write this with sixteen clauses:

Clause 0a $(i_2 \vee i_1 \vee i_0 \vee \mu \vee \overline{s_0})$
Clause 0b $(i_2 \vee i_1 \vee i_0 \vee \overline{\mu} \vee s_0)$
Clause 1a $(i_2 \vee i_1 \vee \overline{i_0} \vee \mu \vee \overline{s_1})$
Clause 1b $(i_2 \vee i_1 \vee \overline{i_0} \vee \overline{\mu} \vee s_1)$
Clause 2a $(i_2 \vee \overline{i_1} \vee i_0 \vee \mu \vee \overline{s_2})$
Clause 2b $(i_2 \vee \overline{i_1} \vee i_0 \vee \overline{\mu} \vee s_2)$
Clause 3a $(i_2 \vee \overline{i_1} \vee \overline{i_0} \vee \mu \vee \overline{s_3})$
Clause 3b $(i_2 \vee \overline{i_1} \vee \overline{i_0} \vee \overline{\mu} \vee s_3)$
Clause 4a $(\overline{i_2} \vee i_1 \vee i_0 \vee \mu \vee \overline{s_4})$
Clause 4b $(\overline{i_2} \vee i_1 \vee i_0 \vee \overline{\mu} \vee s_4)$
Clause 5a $(\overline{i_2} \vee i_1 \vee \overline{i_0} \vee \mu \vee \overline{s_5})$
Clause 5b $(\overline{i_2} \vee i_1 \vee \overline{i_0} \vee \overline{\mu} \vee s_5)$
Clause 6a $(\overline{i_2} \vee \overline{i_1} \vee i_0 \vee \mu \vee \overline{s_6})$
Clause 6b $(\overline{i_2} \vee \overline{i_1} \vee i_0 \vee \overline{\mu} \vee s_6)$
Clause 7a $(\overline{i_2} \vee \overline{i_1} \vee \overline{i_0} \vee \mu \vee \overline{s_7})$
Clause 7b $(\overline{i_2} \vee \overline{i_1} \vee \overline{i_0} \vee \overline{\mu} \vee s_7)$

Those 16 clauses can be explained as follows. For any choice of i_2, i_1, and i_0, the first three terms in the clause will, for 14 clauses, have at least one T, and the clause is therefore satisfied and irrelevant. The two that remain are those numbered by whatever binary number (between 0 and 7) happens to be formed by i_2, i_1, and i_0. For example, if $i_2 = 1$, $i_1 = 1$, and $i_0 = 0$ then all clauses except 6a and 6b are satisfied. However, the first three terms of clauses 6a and 6b are now all Fs. The remaining two terms cause the following. Clauses 6a and 6b will both be satisfied if and only if $\mu = s_6$.

In summary, any satisfying solution will have $\mu = s_x$ where x is the binary number between 0 and 7 formed by $x = 4i_2 + 2i_1 + i_0$.

3.2 A Special Case: $8 \times 8 \times 8$

First, let us consider for simplicity that $K = P = C = \{0,1\}^8$. We have $p_0, p_1, p_2, \ldots, p_7 = \vec{p} \in P$, $k_0, k_1, k_2, \ldots, k_7 = \vec{k} \in K$, and $c_0, c_1, c_2, \ldots, c_7 = \vec{c} \in C$, for the cipher F. We have $p'_0, p'_1, p'_2, \ldots, p'_7 = \vec{p'} \in P$, $k'_0, k'_1, k'_2, \ldots, k'_7 = \vec{k'} \in K$, and $c'_0, c'_1, c'_2, \ldots, c'_7 = \vec{c'} \in C$, for the cipher G.

The goal is to discover the relationship. In other words, each p_i matches up with some p'_j, in the sense that $p_i = p'_j$, but we do not yet know which p'_j is associated with each p_i. Likewise we need to discover the mapping of the cs and the c's, as well as the ks and the k's.

We start with the descriptions of the ciphers F and G. This can be a set of polynomial equations mod 2, or it can be a set of CNF clauses for a SAT-Solver. As we saw in Section 2.1, this is not too difficult to create. Indeed, it is a step in every case of algebraic cryptanalysis [5] and at times it is also done for other reasons, such as circuit minimization. To those, we shall add 72 variables, and several equations/clauses, as follows.

We will add $x_{2,j}$, $x_{1,j}$, and $x_{0,j}$ for $j \in \{0, 1, 2, \ldots, 7\}$, and these 24 variables will reveal to us the relationship between the ps and the p's. Using either the polynomial μ from the previous subsection, or the 16 clauses from there when using CNF notation, we write $s_i = p_i$, $\mu = p'_j$, $i_2 = x_{2,j}$, $i_1 = x_{1,j}$, and $i_0 = x_{0,j}$. This is repeated 8 times, once for each of the 8 values of $j \in \{0, 1, 2, \ldots, 7\}$.

We will add $y_{2,j}$, $y_{1,j}$, and $y_{0,j}$ for $j \in \{0, 1, 2, \ldots, 7\}$, and these 24 variables will reveal to us the relationship between the ks and the k's. Another 8 times, we write $s_i = k_i$, $\mu = k'_j$, $i_2 = y_{2,j}$, $i_1 = y_{1,j}$, and $i_0 = y_{0,j}$. This is also repeated 8 times, once for each of the 8 values of $j \in \{0, 1, 2, \ldots, 7\}$.

We will add $z_{2,j}$, $z_{1,j}$, and $z_{0,j}$ for $j \in \{0, 1, 2, \ldots, 7\}$, and these 24 variables will reveal to us the relationship between the cs and the c's. Yet another 8 times, we write $s_i = c_i$, $\mu = c'_j$, $i_2 = z_{2,j}$, $i_1 = z_{1,j}$, and $i_0 = z_{0,j}$. This is also repeated 8 times, once for each of the 8 values of $j \in \{0, 1, 2, \ldots, 7\}$.

Now suppose that the polynomials are put into a polynomial-system solver (such as MAGMA or PolyBori) and a solution is obtained, or alternatively the CNF clauses are put into a SAT-Solver which finds a satisfying solution. Either way, further suppose that in the satisfying solution, we have

$$x_{2,4} = 1,\ x_{1,4} = 1,\ x_{0,4} = 0$$

then that means $p'_4 = p_{4(1)+2(1)+1(0)} = p_6$. Similarly, suppose we have

$$y_{2,7} = 0,\ y_{1,7} = 1,\ y_{0,7} = 1$$

then that means $k'_7 = k_{4(0)+2(1)+1(1)} = k_3$. Continuing the pattern, if we have

$$z_{2,1} = 0,\ z_{1,1} = 1,\ z_{0,1} = 0$$

then that means $c'_1 = c_{4(0)+2(1)+1(0)} = c_2$.

In more generality, if we have

$$x_{2,j} = \alpha,\ x_{1,j} = \beta,\ x_{0,j} = \gamma$$

then that means $p'_j = p_{4(\alpha)+2(\beta)+1(\gamma)}$. Similarly, if

$$y_{2,j} = \alpha,\ y_{1,j} = \beta,\ y_{0,j} = \gamma$$

then that means $k'_j = k_{4(\alpha)+2(\beta)+1(\gamma)}$. As the reader is no doubt anticipating, if

$$z_{2,j} = \alpha,\ z_{1,j} = \beta,\ z_{0,j} = \gamma$$

then that means $c'_j = c_{4(\alpha)+2(\beta)+1(\gamma)}$.

Therefore, if a satisfying solution exists, these 24 variables expose the mapping of the ps to the p's, the cs to the c's, and the ks to the k's. Conversely, if a satisfying solution does not exist, then no such mapping is possible, and F is not equivalent to G.

3.3 A Useful Family of Polynomials

Before we can generalize our solution, we must first generalize μ. We shall define a family of polynomials, indexed by the positive integers. The "polynomial μ of order w" shall have $w + 2^w$ inputs, and one output. The first w inputs $i_{w-1}, i_{w-2}, i_{w-3}, \ldots, i_0$ can be thought of as an integer $0 \leq x < 2^w$. Then the polynomial will output s_x, choosing the xth of the remaining 2^w inputs $s_0, s_1, s_2, \ldots, s_{2^w-1}$. As the reader has no doubt noticed, the μ from Subsection 3.1 is actually the polynomial μ of order 3 in this notation. To write μ as a polynomial,

$$\mu(i_{w-1}, i_{w-2}, i_{w-3}, \ldots, i_0, s_0, s_1, s_2, \ldots, s_{2^w-1}) = \sum_{j=0}^{j=2^w-1} s_j \prod_{\ell=0}^{\ell=w-1} (1 + bit_\ell(j) + i_\ell)$$

where $bit_\ell(j)$ is the ℓth bit of the positive integer j when j is written in binary. To be precise, $bit_0(j)$ is the least significant bit of j, and $bit_{w-1}(j)$ is the most significant bit of j.

This construction allows for the useful property:

$$\mu(i_{w-1}, i_{w-2}, i_{w-3}, \ldots, i_0, s_0, s_1, s_2, \ldots, s_{2^w-1}) = s_x \text{ where } x = 2^{w-1}i_{w-1} + 2^{w-2}i_{w-2} + 2^{w-3}i_{w-3} + \cdots + i_0$$

This property follows from the fact that this function will always have $2^w - 1$ of its 2^w terms "zeroed-out" by the is regardless of the values that those w variables take. The remaining term is the correct s_i. For example, if $w = 5$ and $i_4 = 1$, $i_3 = 1$, $i_2 = 0$, $i_1 = 0$, and $i_0 = 1$, then μ will output s_{25}, because out of 32 terms, there will be 31 terms with a coefficient of 0 for the s_is, and there is a 1 for the coefficient of s_{25}. Of course, 11001 in binary is 25 in decimal.

As before, this boolean polynomial also comes from computer engineering. The digital circuit given by this boolean polynomial is called "a 2^w-way multiplexer" by computer engineers.

In the CNF notation of SAT-Solvers, we would write this with 2^{w+1} clauses, numbered 0a, 1a, 2a, ..., $(2^w - 1)$a, 0a, 1b, 2b, ..., $(2^w - 1)$b. All these 2^{w+1} clauses will have $w + 2$ terms. The first w terms are $i_{w-1}, i_{w-2}, i_{w-3}, \ldots, i_0$. However, the negation or non-negation of those terms is very important, and is computed as follows:

For $0 \le j < 2^w$, and $0 \le \ell < w$, the ℓth term in the jth clause (regardless if "a" or "b") shall be negated if $bit_{w-1-\ell}(j) = 1$, and not negated if $bit_{w-1-\ell}(j) = 0$. Thus if $bit_{w-1-\ell}(j) = 1$ then it shall be $\overline{i_{w-1-\ell}}$ or else if $bit_{w-1-\ell}(j) = 0$ then it shall be $i_{w-1-\ell}$.

We have specified the first w terms of the clauses, and must now specify the $(w + 1)$th and $(w + 2)$th terms. For an "a" clause, the $(w + 1)$th term shall be μ, and for a "b" clause the $(w + 1)$th term shall be $\overline{\mu}$. Finally, for $0 \le j < 2^w$, the $(w + 2)$th term of the clause shall be s_j for a "b" clause, and $\overline{s_j}$ for an "a" clause.

Those 2^{w+1} clauses can be explained as follows. For any choice of i_{w-1}, i_{w-2}, i_{w-3}, ... i_0, the first w terms in the clause will, in $2^{w+1} - 2$ cases, have at least one T, and the clause is therefore satisfied and irrelevant. The two that remain are those numbered by whatever binary number between 0 and 2^w happens to be formed by i_{w-1}, i_{w-2}, i_{w-3}, ... i_0. For example, if $w = 5$ and $i_4 = 1$, $i_3 = 1$, $i_2 = 0$, $i_1 = 0$, and $i_0 = 1$ then all clauses except 25a and 25b are satisfied. However, the first w terms of clauses 25a and 25b are now all Fs. The remaining two terms cause the following. Clauses 25a and 25b will both be satisfied if and only if $\mu = s_{25}$.

In summary, any satisfying solution will have $\mu = s_x$ where x is the binary number between 0 and $2^w - 1$ formed by

$$x = 2^{w-1}i_{w-1} + 2^{w-2}i_{w-2} + 2^{w-3}i_{w-3} + \cdots + i_0$$

3.4 The Final Algorithm

Usually, the length of the key, the ciphertext, and the plaintext are (possibly distinct) powers of two. Sometimes this is not the case, but we will address that in Section 3.7, and it causes no difficulty. Let us consider $\ell_p = 2^{L_p}$, $\ell_k = 2^{L_k}$, and $\ell_c = 2^{L_c}$. This means that $K = \{0,1\}^{2^{L_k}}$, $P = \{0,1\}^{2^{L_p}}$, and $C = \{0,1\}^{2^{L_c}}$. For the cipher F, we have 2^{L_p} plaintext bits, $p_0, p_1, p_2, \ldots, p_{(2^{L_p}-1)}$, we have 2^{L_k} key bits, $k_0, k_1, k_2, \ldots, k_{(2^{L_k}-1)}$, and we have 2^{L_c} ciphertext bits, $c_0, c_1, c_2, \ldots, c_{(2^{L_c}-1)}$. Similarly define p'_i, k'_i, and c'_i for the cipher G.

As before, the goal is to discover the relationship. In other words, each p matches up with some p', in the sense that $p_i = p'_j$, but we do not yet know which p'_j is associated with each p_i. Likewise we need to discover the mapping of the cs and the c's, as well as the ks and the k's.

We start with the descriptions of the ciphers F and G. As before, this can be a set of polynomial equations mod 2, or it can be a set of CNF clauses for a SAT-Solver. As noted earlier, and as we saw in Section 2.1,

this is not too difficult to create. Indeed, it is a step in every case of algebraic cryptanalysis [5] and at times it is also done for other reasons, such as circuit minimization.

1. To that description will shall add $L_k 2^{L_k} + L_p 2^{L_p} + L_c 2^{L_c}$ variables, and several equations/clauses. More precisely, we shall require 2^{L_p} copies of the polynomial μ of order L_p, and 2^{L_k} copies of the polynomial μ of order L_k, as well as 2^{L_c} copies of the polynomial μ of order L_c.

2. We will add $x_{0,j}$, $x_{1,j}$, $x_{2,j}$, $\ldots$, $x_{L_p-1,j}$ for $j \in \left\{0, 1, 2, \ldots, 2^{L_p-1}\right\}$, and these $L_p 2^{L_p}$ variables will reveal to us the relationship between the ps and the p's. Using either the polynomial μ of order L_p from Section 3.3, or the 2^{L_p+1} clauses from that section when using CNF notation, we write $s_i = p_i$, $\mu = p'_j$, $i_0 = x_{0,j}$, $i_1 = x_{1,j}$, $i_2 = x_{2,j}$, $\ldots$, $i_{L_p-1} = x_{L_p-1,j}$. This is repeated 2^{L_p} times, once for each of the 2^{L_p} values of $j \in \left\{0, 1, 2, \ldots, 2^{L_p} - 1\right\}$.

3. We will add $y_{0,j}$, $y_{1,j}$, $y_{2,j}$, $\ldots$, $y_{L_p-1,j}$ for $j \in \left\{0, 1, 2, \ldots, 2^{L_k-1}\right\}$, and these $L_k 2^{L_k}$ variables will reveal to us the relationship between the ks and the k's. Using either the polynomial μ of order L_k from the previous subsection, or the 2^{L_k+1} clauses from there when using CNF notation, we write $s_i = k_i$, $\mu = k'_j$, $i_0 = y_{0,j}$, $i_1 = y_{1,j}$, $i_2 = y_{2,j}$, $\ldots$, $i_{L_k-1} = y_{L_k-1,j}$. This is repeated 2^{L_k} times, once for each of the 2^{L_k} values of $j \in \left\{0, 1, 2, \ldots, 2^{L_k} - 1\right\}$.

4. We will add $z_{0,j}$, $z_{1,j}$, $z_{2,j}$, $\ldots$, $z_{L_c-1,j}$ for $j \in \left\{0, 1, 2, \ldots, 2^{L_c-1}\right\}$, and these $L_c 2^{L_c}$ variables will reveal to us the relationship between the cs and the c's. Using either the polynomial μ of order L_c from the previous subsection, or the 2^{L_c+1} clauses from there when using CNF notation, we write $s_i = c_i$, $\mu = c'_j$, $i_0 = z_{0,j}$, $i_1 = z_{1,j}$, $i_2 = z_{2,j}$, $\ldots$, $i_{L_c-1} = z_{L_c-1,j}$. This is repeated 2^{L_c} times, once for each of the 2^{L_c} values of $j \in \left\{0, 1, 2, \ldots, 2^{L_c} - 1\right\}$.

5. Now suppose that the CNF clauses are put into a SAT-Solver, and it finds a satisfying solution. Alternatively, suppose that the polynomials are put into a polynomial-system solver (such as MAGMA or PolyBori) and a solution is obtained. Further suppose that in the solution, we have

$$x_{0,j} = \alpha_0,\ x_{1,j} = \alpha_1,\ x_{2,j} = \alpha_2,\ \ldots,\ x_{L_p-1} = \alpha_{L_p-1}$$

   then that means $p'_j = p_x$ where

$$x = \alpha_0 + 2\alpha_1 + 4\alpha_2 + \cdots + 2^{L_p-1}\alpha_{L_p-1}$$

6. Similarly, suppose we have

$$y_{0,j} = \alpha_0,\ y_{1,j} = \alpha_1,\ y_{2,j} = \alpha_2,\ \ldots,\ y_{L_k-1} = \alpha_{L_k-1}$$

   then that means $k'_j = k_y$ where

$$y = \alpha_0 + 2\alpha_1 + 4\alpha_2 + \cdots + 2^{L_k-1}\alpha_{L_k-1}$$

7. Finally, suppose we have

$$z_{0,j} = \alpha_0,\ z_{1,j} = \alpha_1,\ z_{2,j} = \alpha_2,\ \ldots,\ z_{L_c-1} = \alpha_{L_c-1}$$

   then that means $c'_j = c_z$ where

$$z = \alpha_0 + 2\alpha_1 + 4\alpha_2 + \cdots + 2^{L_c-1}\alpha_{L_c-1}$$

Therefore, if a satisfying solution exists, these $L_k 2^{L_k} + L_p 2^{L_p} + L_c 2^{L_c}$ variables expose the mapping of the ps to the p's, the cs to the c's, and the ks to the k's. Conversely, if a satisfying solution does not exist, then no such mapping is possible, and F is not equivalent to G.

3.5 An Estimation of Difficulty as Applied to AES-256

To further clarify the previous description, and to estimate the difficulty of actually solving the problem at hand, we shall now consider the case of the cipher AES-256.

- For the AES-128, $\ell_p = \ell_c = 128$ while $\ell_k = 256$. This means that $L_p = L_c = 7$ and $L_k = 8$.
- We shall require $\ell_p = 128$ copies of the polynomial μ of order $L_p = 7$. Each requires 7 variables, for a total of $7(128) = 896$ variables.
- We shall require $\ell_c = 128$ copies of the polynomial μ of order $L_c = 7$. Again, each requires 7 variables, for a total of $7(128) = 896$ variables.
- We shall require $\ell_k = 256$ copies of the polynomial μ of order $L_k = 8$. This time each requires 8 variables, for a total of $8(128) = 1024$ variables.
- This is a total of $896 + 896 + 1024 = 2816$ additional variables. However, we must also count the number of new equations mod 2, or the number of new CNF clauses for a SAT-Solver.
    - If working with equations mod 2, each polynomial μ (of any order) is a single equation. Therefore, we require $128 + 128 + 256 = 512$ equations.
    - If working with CNF clauses for a SAT-Solver, each copy of the polynomial μ of order 7 requires 256 clauses, and each copy of the polynomial μ of order 8 requires 512 clauses. Therefore, we would require $(256)(128 + 128) + (512)(256) = 196,608$ additional clauses.

Clearly, this computation is too large to run on a laptop in a short amount of time—at least at the time of the publication of this paper. Nonetheless, it is clear that this computation is much easier than trying all $2^{3116.32}$ possible permutations. Moreover, the author conjectures that with a month of time on a cloud computing platform (e.g. SageMathCloud [3]), the problem could be solved.

3.6 Consideration of Non-Injective Maps

Suppose that the xs map two different p's to the same p. (For example, p'_{17} and p'_{19} are both assigned to p_8.) Similarly, the ys might map two different k's to the same k, or the zs might map two different c's to the same c.

At first, that might seem like a major flaw in the approach of this paper. However, this type of invalid mapping will not actually occur. Take the example of p'_{17} and p'_{19} both being assigned to p_8. This means that the map from the p's to the ps is not injective. Since the domain and range of that map are both the same finite set $\{0,1\}^{2^{L_p}}$, non-injectivity implies non-surjectivity. Therefore, the map is not surjective, and some p must have no p' assigned to it. (Suppose it is p_4.) The only way that this can result in some satisfying solution being computed is if p_4 is not actually used in the computation of F as a function. We say such a block cipher would never actually occur, because the plaintext bit p_4 would be ignored. Surely, it would produce the same ciphertext when p_4 (as a bit) is flipped, and this violates unique decodability.

Similar arguments apply to the ks and to the cs. For the ks, if one of the key bits is ignored, then the key length is actually not $\ell_k = 2^{L_k}$, but is smaller. Such a cipher, ignoring one of its key bits, would be cryptographically absurd—the cipher designer is aiding the attacker by cutting the key space in half for each ignored bit. An ignored bit of ciphertext would be totally meaningless. We cannot consider a bit of $\vec{c}$ to be an output of the block cipher if it is never actually computed. If a ciphertext is computed it is not ignored, and if it is ignored, it is not computed.

3.7 What if some L is not a Power of Two?

It is frequent that the length of the plaintext, the ciphertext, and the key are powers of two. However, this is not always the case, as the DES has a key of length 56 and the AES allows a key of length 192.

There is no harm in using the next highest power of two. For example, $32 < 56 < 64$, so for the DES, we would write $L_k = 6$ since $2^6 = 64$. No clauses should be written for the key bits that do not exist (i.e. k_{56}, k_{57}, k_{58}, ..., k_{63}). Using the same argument as the previous subsection, there is no need to worry about the ys assigning some k' to a k that does not exist.

For a non-cryptographic application, one can easily write clauses to prevent such assignments. For example, if $\ell_k = 56$, one could freely match up k_{56} with k'_{56}, k_{57} with k'_{57}, and k_{63} with k'_{63}, since none of them actually exist—only k_0, k_1, ..., k_{55} actually exist.

3.8 Forcing Only the Consideration of Injective Maps

In the event that some future reader has an application of this paper to a problem distant from cryptography, it should be noted that the injectivity can be forced. The two clauses

$$(x_{i,j} \vee x_{i,k} \vee e_{i,j,k})(\overline{x_{i,j}} \vee \overline{x_{i,k}} \vee e_{i,j,k})$$

will force $e_{i,j,k} = 1$ if $x_{i,j} = x_{i,k}$. (Note, this is not an if-and-only-if relationship.)

Those two clauses must be repeated for all L_p values of i satisfying $0 \leq i < L_p$. Next, the clause

$$(\overline{e_{0,j,k}} \vee \overline{e_{1,j,k}} \vee \overline{e_{2,j,k}} \vee \cdots \vee \overline{e_{L_p-1,j,k}})$$

will reject any solution that has $e_{0,j,k}, e_{1,j,k}, e_{2,j,k}, \ldots, e_{L_p-1,j,k}$ all equal to 1. Therefore, in any satisfying assignment at least one $e_{i,j,k}$ equals zero. Therefore, one of the following equalities must be false.

$$x_{0,j} = x_{0,k},\ x_{1,j} = x_{1,k},\ x_{2,j} = x_{2,k},\ \ldots,\ x_{L_p-1,j} = x_{L_p-1,k}$$

Finally, that means that p'_j and p'_k cannot be assigned the same ps. Those $2L_p + 1$ clauses must be repeated for all $(2^{L_p})(2^{L_p} - 1)/2$ possible pairs of p'_j and p'_k. That is all j and k such that $0 \leq j < k < 2^{L_p}$.

This is not as bad as it sounds, as even for the AES where $L_p = 7$, there would be 8128 pairs, each of 15 clauses, for a total of 121,920 additional clauses. Yet, the author would like to emphasize that this would not be required in a realistic cryptographic problem.

3.9 Application to Compression Functions inside Hash Functions

Cryptographic hash functions are also valuable elements of cryptosystems, and their design is not at all simple. It is possible that someone might attempt to pirate a hash function. For example, the now obsolete MD5 has a compression function with 512 bits of input and 128 bits of output. The methods here would work fine to detect permutations by a software pirate. However, it should be noted that these functions are not themselves permutations. For example, MD5 has 2^{512} possible inputs, and 2^{128} possible outputs. That will not adversely affect the methods of this paper. Instead, a 512×512 permutation matrix applied to the inputs, and a 128×128 permutation matrix applied to the outputs, would be searched for.

4 Connections to a Related Problem in Coding Theory

Much research has been done on the code-equivalence problem [10] [12] [28] [32]. At first glance, and even at a second glance, these topics might seem closely related, or perhaps nearly identical. However, an extremely fundamental distinction must be made. Before defining the code-equivalence problem, it is necessary to define the concept of an error-correcting linear code (or error-detecting linear code).

4.1 The Code-Equivalence Problem

Suppose there is an error-correcting linear code (or error-detecting linear code) where any message $\vec{m}$ is encoded with codeword $\vec{c} = A\vec{m}$. For example, an $[a, b, d]_Q$-linear code has a matrix A over the finite field $GF(Q)$, and the dimensions of A are $a \times b$. That means the messages $\vec{m}$ are from $\{0,1\}^b$ and the codewords $\vec{c}$ are from $\{0,1\}^a$, with $a > b$. Since the code must be uniquely decodable to be of any use, it is necessary that the map be injective, and this means that the rank of A should be equal to b, or more plainly, that A is a full-rank matrix. We did not mention yet the parameter d, which is the minimum distance of the code. The hamming distance between any two distinct codewords $\vec{c}_1$ and $\vec{c}_2$ is guaranteed to be at least d.

Given two distinct codes that are both $[a, b, d]_Q$-linear codes, with matrices A_1 and A_2, there might exist some permutation matrix P_1 that is an $a \times a$ matrix, and some permutation matrix P_2 that is a $b \times b$ matrix, such that $P_1 A_1 P_2 = A_2$. If such matrices P_1 and P_2 exist, then we say that the codes are equivalent. The code-equivalence problem is the problem of determining if two distinct codes are actually equivalent, or not.

In summary, we wish to detect when one code can be thought of as applying a permutation to the input message, encoding with another code, and then applying another permutation to the output codeword. Clearly, this question is highly related to the problem of this paper, and the author thanks the anonymous referee who pointed out this connection.

4.2 The Code-Equivalence Problem and the Support Splitting Algorithm

The Support Splitting Algorithm by Sendrier [31] [33], can efficiently detect if one error-correcting (or error-detecting) linear code is a permutation of another. Moreover, it also produces the exact permutations involved. For each coordinate, it computes a set of invariant properties that Sendrier calls "a signature." If the signature is different for all coordinates, then the permutations can be recovered. For random problem instances, the algorithm runs in polynomial time. However, that research does not attempt to address non-linear codes at all, only linear codes. See [31] for details.

4.3 The Fundamental Difference with the Code-Equivalence Problem

The most fundamental difference between the two problems is that block ciphers are non-linear. There is also the question that block ciphers require three matrices, whereas the code-equivalence problem requires only two. In practice, it is extremely important that block ciphers are very different from linear functions because of linear cryptanalysis. Block ciphers that are linear, or nearly linear, would be easily broken via linear cryptanalysis, which we will now explore.

4.4 The Connection with Linear Cryptanalysis

One can consider equations of the form

$$c_{a_1} + c_{a_2} + \cdots + c_{a_n} + p_{b_1} + p_{b_2} + \cdots + p_{b_m} + k_{c_1} + k_{c_2} + \cdots + k_{c_r} = 0$$

or alternatively $= 1$. As before, c_{a_i} refers to the a_ith bit of the ciphertext, p_{b_i} refers to the b_ith bit of the plaintext, and k_{c_i} refers to the c_ith bit of the key. For a block cipher that indeed performs like a pseudorandom permutation, any equation of this form should be true with probability very close to $1/2$. By probability, we mean the case when the key and the plaintext are bit strings that uniformly randomly generated, with each bit being an independent and identically distributed fair coin.

For a particular block cipher, careful analysis might produce some equations that actually hold with probability significantly above $1/2$. This concept is at the core of linear cryptanalysis, as discovered by Mitsuru Matsui in the 1990s [21] [22] [23], and as generalized by Eli Biham [11], and later Nicolas Courtois [13]. Similarly, if the probability of some equation holding is far below $1/2$, then flipping the constant on the right-hand-side produces an equation that holds with high probability. The subject of linear cryptanalysis is very large, with too many papers to cite here.

Modern block ciphers are expected to show resistance to linear cryptanalysis prior to adoption. That means that no such equations should be found that hold with probability significantly different from 1/2. (See also [39].) In fact, the entire subject of bent functions is devoted to making functions that are distant from all available linear functions of the same domain and range [14]. (Distance here has a formal mathematical definition, which we omit to save space.) Such bent functions guarantee that any such equations that can be written will be true with probability optimally close to 1/2.

4.5 Summary of Comparison with the Code-Equivalence Problem

In any case, because modern block ciphers are expected to show resistance to linear cryptanalysis prior to adoption, such as by using bent functions or by probabilistic analysis of "S-Boxes" and similar non-linear functions inside the block cipher, it is simply not possible to find a linear function which is a good approximation of any modern block cipher.

That means the Support Splitting Algorithm and related methods cannot be used to solve this problem of detecting when one block cipher is a permutation of another block cipher.

5 Conclusions

At times, software pirates might hope to evade patent fees, licensing agreements, or cipher-export restrictions by taking an existing block cipher but permuting its plaintext, ciphertext, or keys. At first glance, it seems as though this would preserve all security properties while being hard to detect.

The problem considered here is somewhat related to the rather difficult "isomorphism of polynomials" problem, but it is only slightly harder than the task of determining if two (non-permuted) circuits are identical. That is because a proper digital description of the two ciphers, along with a moderate number of additional equations/clauses is all that is needed to solve the problem. This paper describes all that is needed whether working with CNF clauses and a SAT-solver, or solving a system of polynomial equations mod 2. Furthermore, one obtains the actual permutations used with no additional effort.

Therefore, the criminal activity can be easily detected.

References

[1] Magma. Software Package. Available at `http://magma.maths.usyd.edu.au/magma/`

[2] MiniSAT. Software Package. Available at `http://www.cs.chalmers.se/cs/Research/FormalMethods/MiniSat/` or `http://minisat.se/Papers.html`

[3] Sage. Software Package. Available at `http://www.sagemath.org/`

[4] Agrawal, M, and Saxena, N.: "Equivalence of F-algebras and Cubic Forms." *Proc: Symposium on Theoretical Aspects of Computer Science (STACS'06), Lecture Notes in Computer Science*, **Vol. 3884**, Pp 115–125. Springer. (2006). Available at `http://repository.ias.ac.in/92017/1/3-p.pdf`

[5] Bard, G.: *Algebraic Cryptanalysis*. Springer-Verlag. (2009). Available at `http://www.springer.com/us/book/9780387887562`

[6] Bard, G.: *Algorithms for the Solution of Linear and Polynomial Systems of Equations over Finite Fields, with Applications to Cryptanalysis*. PhD Dissertation. Department of Applied Mathematics and Scientific Computation, University of Maryland at College Park. Defended April 30, 2007. Available at `http://www.gregorybard.com/papers/bard_thesis.pdf`

[7] Bartee, T.C.: *Digital Computer Fundamentals*, 6th Edition. McGraw Hill (1985).

[8] Berthomieu, J., Faugère, J.-C., Perret L.: "Polynomial-time algorithms for quadratic isomorphism of polynomials: The regular case." *Journal of Complexity*. **Vol. 31**. Pp. 590–616. (2015). Available at `https://arxiv.org/abs/1307.4974`

[9] Bard, G., Courtois, N., Jefferson, C.: "Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-Solvers." Preprint. Cryptology ePrint Archive, Report 2007/024 (2006). Available at `http://eprint.iacr.org/2007/024.pdf`

[10] Babai, L., Codenotti, P., Grochow, J., and Qiao, Y.: "Code Equivalence and Group Isomorphism." *Proc: 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*. Pp. 1395–1408. SIAM. (2011). Available at `http://people.cs.uchicago.edu/~laci/papers/soda11.pdf`

[11] Biham, E.: "On Matsui's Linear Cryptanalysis." *Advances in Cryptology (EUROCRYPT'94), Lecture Notes in Computer Science*, **Vol. 950**, pp. 341–355. Springer-Verlag (1994). Available at `https://link.springer.com/chapter/10.1007/BFb0053449`

[12] Bouyukliev, I.: "About the Code Equivalence." *Advances in Coding Theory and Cryptography*. World Press Scientific. Pp. 126–151. (2007). Available at `https://www.worldscientific.com/doi/abs/10.1142/9789812772022_0009`

[13] Courtois, N.: "Feistel Schemes and Bi-linear Cryptanalysis," *Advances in Cryptology (CRYPTO'04), Lecture Notes in Computer Science*, **Vol. 3152**, Pp. 23–40, Springer-Verlag, (2005).

[14] Cusick, T., and Stanica, P.: *Cryptographic Boolean Functions and Applications*. 2nd Edition. Academic Press. (2017).

[15] Faugère, J.-C.: "A new efficient algorithm for computing Gröbner Bases (F_4)." *Journal of Pure and Applied Algebra*. **Vol. 139**. Pp. 61–88. (1999). Available at `http://www-polsys.lip6.fr/~jcf/Papers/F99a.pdf`

[16] Faugère, J.-C., and Perret, L.: "Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects." *Advances in Cryptology (EUROCRYPT'06), Lecture Notes in Computer Science*, **Vol. 4004**, pp. 30–47. Springer-Verlag (2006). Available at `http://www-spaces.lip6.fr/papers/FP06b.pdf`

[17] Geiselmann, W., Meier, W., and Steinwandt, R.: "An attack on the isomorphisms of polynomials problem with one secret". *International Journal of Information Security*. **Vol. 2**, No. 1, (2003). Available at `http://eprint.iacr.org/2002/143.pdf`

[18] Goldreich, O., Micali, S., and Wigderson, A.: "Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems." *Journal of the Association of Computing Machinery*, **Vol. 38**, No. 3, Pp. 691–729, (1991). Available at `https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Zero%20Knowledge/Proofs_That_Yield_Nothing_But_Their_Validity_or_All_Languages_in_NP_Have_Zero-Knowledge_Proof_Systems.pdf`

[19] Goldreich, O.: *Foundations of Cryptography, Vol 2: Basic Applications*. Cambridge University Press. 2004.

[20] N. Kayal.: "Efficient Algorithms for Some Special Cases of the Polynomial Equivalence Problem." In: Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1409–1421. SIAM. (2011). Available at `https://www.microsoft.com/en-us/research/publication/efficient-algorithms-for-some-special-cases-of-the-polynomial-equivalence-problem/`

[21] M. Matsui: "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology (EUROCRYPT'93), Lecture Notes in Computer Science*, **Vol. 765**, Pp. 386–397, Springer-Verlag, (1993). Available at `https://link.springer.com/content/pdf/10.1007/3-540-48285-7_33.pdf`

[22] M. Matsui: "The First Experimental Cryptanalysis of the Data Encryption Standard," *Advances in Cryptology (CRYPTO'94), Lecture Notes in Computer Science*, **Vol. 839**, Pp. 1–11, Springer-Verlag, (1994). Available at `https://link.springer.com/chapter/10.1007/3-540-48658-5_1`

[23] M. Matsui: "On correlation between the order of S-boxes and the strength of DES," *Advances in Cryptology (EUROCRYPT'94), Lecture Notes in Computer Science*, **Vol. 950**, Pp. 366–375, Springer-Verlag, (1995). Available at `https://link.springer.com/chapter/10.1007/BFb0053451`

[24] Matsumoto, T., and Imai, H.: "Public quadratic polynomial-tuples for efficient signature-verification and message-encryption," *Advances in Cryptology (EUROCRYPT'88), Lecture Notes in Computer Science*, **Vol. 330**, Pp. 419–453, Springer-Verlag, (1988). Available at `http://link.springer.com/chapter/10.1007/3-540-45961-8_39`

[25] Patarin, J.: "Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms." *Advances in Cryptology (EUROCRYPT'96), Lecture Notes in Computer Science*, **Vol. 1070**, Pp. 33–48, Springer-Verlag, (1996). Available at `http://citeseer.ist.psu.edu/article/patarin96hidden.html`

[26] Patarin, J., Goubin, L., and Courtois, N.: "Improved Algorithms for Isomorphisms of Polynomials." *Advances in Cryptology (EUROCRYPT'98), Lecture Notes in Computer Science*, **Vol. 1403**, pp. 184–200. Springer-Verlag (1998). Available at `http://citeseer.ist.psu.edu/article/patarin98improved.html`

[27] Perret, L.: "A fast cryptanalysis of the isomorphism of polynomials with one secret problem." *Advances in Cryptology (EUROCRYPT'05), Lecture Notes in Computer Science*, **Vol. 3495**, pp. 354–370. Springer-Verlag (2005). Available at `https://link.springer.com/chapter/10.1007%2F11426639_21`

[28] Petrank, E., and Roth, R.: "Is Code Equivalence Easy to Decide?" *IEEE Transactions on Information Theory*, **Vol. 43**, No. 5, Pp. 1602–1604, (1997). Available at `https://ieeexplore.ieee.org/document/623157/`

[29] Plût, J., Fouque, P.-A., and Macario-Rat, G.: "Solving the 'Isomorphism of Polynomials with Two Secrets' Problem for All Pairs of Quadratic Forms." Preprint. (2017). Available at `http://arxiv.org/pdf/1406.3163v2.pdf`

[30] Saxena, N.: "Morphisms of Rings and Applications to Complexity (PhD Thesis)." Indian Institute of Technology, Kanpur, 2006. Available at `https://www.cse.iitk.ac.in/users/nitin/papers/thesis.pdf`

[31] Sendrier, N.: "Finding the permutation between equivalent linear codes: The support splitting algorithm." *IEEE Transactions on Information Theory*, **Vol. 46**, No. 4, Pp. 1193–1203, (2000). Available at `https://ieeexplore.ieee.org/abstract/document/850662/`

[32] Sendrier, N., and Simos, D.: "The Hardness of Code Equivalence over F_q and its Application to Code-based Cryptography." *Proc: Post-Quantum Cryptography (PQCRYPTO'13), Lecture Notes in Computer Science* **Vol. 7932**, Pp 203–216. Springer. (2013). Available at `https://link.springer.com/chapter/10.1007/978-3-642-38616-9_14`

[33] Simos, D.: "The Support Splitting Algorithm and its Application to Code-based Cryptography." 3rd Code-Based Cryptography Workshop. Technical University of Denmark, Lyngby, Denmark. (2012). Available at `https://pdfs.semanticscholar.org/presentation/2293/cf394a050bdc8da894b66e21b1ecf86650b9.pdf`

[34] Trappe, W., and Washington, L.: *Introduction to Cryptography with Coding Theory*. 2nd Edition. Pearson, Prentice-Hall. 2006.

[35] Tang, S., and Xu, L.: "Proxy signature scheme based on isomorphisms of polynomials," In: *Proc. Network and System Security (NSS'12), Lecture Notes in Computer Science*, **vol. 7645**, Springer, Pp. 113–125. (2012). Available at http://link.springer.com/chapter/10.1007/978-3-642-34601-9_9

[36] Tang, S., and Xu, L.: "Towards provably secure proxy signature scheme based on isomorphisms of polynomials." *Future Generation Computer Systems* **Vol. 30**, Elsevier, Pp 91–97. (2014). Available at http://www.sciencedirect.com/science/article/pii/S0167739X13001179

[37] Yang, G., Tang, S., and Yang, L.: "A Novel Group Signature Scheme based on MPKC." *Information Security Practice and Experience (ISPEC'11), Lecture Notes in Computer Science*, **Vol. 6672**, Springer, Pp 181–195. (2011). Available at http://link.springer.com/chapter/10.1007/978-3-642-21031-0_14

[38] Wolf, C., and Preneel, B.: "Equivalent keys in multivariate quadratic public key systems," *Journal of Mathematical Cryptology*, **Vol. 4**, No. 4, Pp. 375–415, (2011). Available at https://eprint.iacr.org/2005/464.pdf

[39] Vaudenay, S.: "On Measuring Resistance to Linear Cryptanalysis." Mikulášská Kryptobesídka. Prague, Czech Republic. (2003). Available at https://pdfs.semanticscholar.org/582e/1b42d4544e059cd5134a5cbed67eb23b937d.pdf