Optimization and Guess-then-Solve Attacks in Cryptanalysis

Guangyan Song

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

August 29, 2018

I, Guangyan Song, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

In this thesis we study two major topics in cryptanalysis and optimization: software algebraic cryptanalysis and elliptic curve optimizations in cryptanalysis. The idea of algebraic cryptanalysis is to model a cipher by a Multivariate Quadratic (MQ) equation system. Solving MQ is an NP-hard problem. However, NP-hard problems have a point of phase transition where the problems become easy to solve. This thesis explores different optimizations to make solving algebraic cryptanalysis problems easier.

We first worked on guessing a well-chosen number of key bits, a specific optimization problem leading to guess-then-solve attacks on GOST cipher. In addition to attacks, we propose two new security metrics of contradiction immunity and SAT immunity applicable to any cipher. These optimizations play a pivotal role in recent highly competitive results on full GOST. This and another cipher Simon, which we cryptanalyzed were submitted to ISO to become a global encryption standard which is the reason why we study the security of these ciphers in a lot of detail.

Another optimization direction is to use well-selected data in conjunction with Plaintext/Ciphertext pairs following a truncated differential property. These allow to supplement an algebraic attack with extra equations and reduce solving time. This was a key innovation in our algebraic cryptanalysis work on NSA block cipher Simon and we could break up to 10 rounds of Simon64/128. The second major direction in our work is to inspect, analyse and predict the behaviour of ElimLin attack the complexity of which is very poorly understood, at a level of detail never seen before. Our aim is to extrapolate and discover the limits of such attacks, and go beyond with several types of concrete improvement.

Finally, we have studied some optimization problems in elliptic curves which also deal with polynomial arithmetic over finite fields. We have studied existing implementations of the secp256k1 elliptic curve which is used in many popular cryptocurrency systems such as Bitcoin and we introduce an optimized attack on Bitcoin brain wallets and improved the state of art attack by 2.5 times.

Keywords: algebraic cryptanalysis, SAT solver, ElimLin, symmetric encryption, GOST, Simon, Bitcoin brain wallets, Elliptic curves

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Nicolas Courtois for his guidance and advice throughout my research. He is an excellent example of codebreaker and a great mentor. His patience, motivation, enthusiasm and immense knowledge have been invaluable throughout my academic and personal development.

I would like to express my great appreciation to Dr. Daniel Hulme for his endless support, continuous encouragement, valuable suggestions and the working opportunity he offered from Satalia over the years. Furthermore, I thank my UCL colleagues Dr. Theodosis Mourouzis, Dr. Jie Xiong and Yongxin Yang for the sleepless nights when we were working together before deadlines, and for all the fun we have had in the last few years.

I extend my gratitude to Dr. Mark Herbster, Dr. David Clark, Dr. Earl Barr from UCL and Steven Poulson from Cisco, for their kindness and advice, offering internship opportunities in their groups and leading me in working on exciting projects.

Finally, and most importantly, a very special thank you goes to my parents and my wife for their love during all these years of my Ph.D. studies. It would have been impossible to have done it without them.

Contents

List of Figures

List of Tables

List of Publications

- Courtois, N.T., Gawinecki, J.A. and Song, G., 2012. Contradiction immunity and guess-then-determine attacks on GOST. Tatra Mountains Mathematical Publications, 53(1), pp.65-79.
- Courtois, N., Mourouzis, T., Song, G., Sepehrdad, P. and Susil, P., 2014. Combined algebraic and truncated differential cryptanalysis on reduced-round Simon. In Proceedings of the 11th International Conference on Security and Cryptography, pp. 399-404.
- Courtois, N.T., Mourouzis, T., Misztal, M., Quisquater, J.J. and Song, G., 2015. Can GOST Be Made Secure Against Differential Cryptanalysis?. Cryptologia, 39(2), pp.145-156.
- Mourouzis, T., Song, G., Courtois, N. and Christofii, M., 2015. Advanced differential cryptanalysis of reduced-round simon64/128 using large-round statistical distinguishers. Cryptology ePrint Archive, Report 2015/481
- Nicolas Courtois, Guangyan Song, Ryan Castellucci: Speed Optimizations in Bitcoin Key Recovery Attacks, will appear in journal post-proceedings of CECC 2016, Central European Conference on Cryptology, Tatra Mountains Mathematic Publications.
- Courtois N., Sepehrdad P., Song G. and Papapanagiotakis-Bousy I. 2016. Predicting Outcomes of ElimLin Attack on Lightweight Block Cipher Simon. International Conference on Security and Cryptography pp. 465-470.

List of Software Deliverables

- N Courtois, T Mourouzis, G Song. Simonspeck. https://github.com/GSongHashrate/SimonSpeck NSA Simon and Speck C++ implementation, equation generator for all versions of Simon cipher.
- Nicolas Courtois, Jason Papapanagiotakis, Guangyan Song, Chris Park: "Fast Bitcoin data mining Tutorial" (used in GA18 and GA12 teaching/projects at University College London) http://www.nicolascourtois.com/bitcoin/fast_bitcoin_data_mining.pdf
- DeepElimLin: "Java tool for DEEP INSPECTION of equations generated with ElimLin over GF(2) in Cryptalanalysis of Block Ciphers", developed by Guangyan Song and Nicolas Courtois. Available at: http://www.cryptosystem.net/aes/tools.html
- How to crack bitcoin passwords at a very high speed: brainflayer cracker. https://github.com/ryancdotorg/brainflayer written by Ryan Castellucci. Nicolas Courtois and Guangyan Song contributed the code in ec_pubkey_fast.c which more than doubles the speed of public key computations.
- How to crack bitcoin and LinkedIn passwords at home (easy starter project for UCL students and GA18 code breaking competition, by Nicolas Courtois and Guangyan Song. Cf. https://drive.google.com/open?id=0B0iephZbbC3uSU9WRUdWZVZfa1U and database files https://drive.google.com/file/d/0B0iephZbbC3uWERlanN5b3hJak0.

Chapter 1

Introduction

Cryptography is the study of mathematical techniques that ensure the confidentiality and integrity of information. Cryptography is one of the oldest fields of technical study which we can find records of. Going back to 1900 BC, cryptography is found in non-standard hieroglyphs carved into monuments from the Old Kingdom of Egypt [107]. Modern cryptography started out as classified military technology, but now has become very common in our daily lives. Cryptography is not only used in banking cards, secure websites and electronic signatures, but also in public transport cards, car keys, and building passes.

Block cipher is one of the main tools in cryptography; It uses a secret key to transform a plaintext into a ciphertext in such a way that this secret key is needed to recover the original plaintext. During the last 30 years, the academic research on the security of block ciphers has evolved from an empirical way to solve the problem of designing a secure algorithm towards a list of well-understood and well-established security properties that a block cipher must fulfil in order to be secure. Unfortunately, the security of a block cipher is still heavily dependent on the talent, the intuition, and the time at disposal of the people attempting to break it!

Currently, because of the continuously growing impact of mobile phones, smart cards, RFID tags, sensor networks, and the rapid development in the Internet of Things (IoT), there is a huge demand to provide security and to design suitable cryptographic algorithms that can be efficiently implemented in resource-constrained devices. The area of cryptography that studies the design and the secu-

rity of such lightweight cryptographic primitives, called lightweight cryptography, is rapidly evolving and becoming increasingly important.

Most of the cryptographic primitives have been carefully designed, especially those that have been standardized. However, not all of them have been well studied by researchers. Special properties inside widely used cryptography schemes, which might lead to faster attacks, are discovered every year. Also, in real life cryptography applications, bad design, implementation or choice of parameters could lead to huge security issues.

The main aim of my PhD research is to investigate the use of and to develop various optimization tools and software, such as SAT solver and evolutionary algorithms, in the field of automated cryptanalysis; apply cryptanalysis techniques to modern block ciphers, (such as GOST[96], Simon [14] and even elliptic curve cryptography problems) with optimization tools or software; and check if such tools can improve the current best attacks and discover new attacks. We hope to contribute to future government standards and popular cryptography applications. In this thesis, our cryptanalysis targets are the Russian government standard cipher GOST, the NSA newly proposed cipher Simon, and Bitcoin Elliptic Curve.

The first part of this thesis focuses on software algebraic crytanalysis. Automated black box techniques, such as SAT solvers or Gröbner basis computations, have become increasingly sophisticated and powerful. In the domain of algebraic cryptanalysis, they are used to solve equation systems that are converted from the cipher. Solving such equation systems is an NP-hard problem. When the problem becomes larger (e.g., trying to solve a larger numberof rounds), it becomes impossible to solve using a normal PC. This is the fundamental problem of software algebraic cryptanalysis. However, NP-hard problems normally have a “phase transition” point when the problem is suddenly changed from “hard to solve” to “easy to solve”. This phase transition also appears in software algebraic cryptanalysis, for example, using chosen plaintext in a counter mode then solving by ElimLin [56]. In this thesis, we explore different ways to improve software algebraic cryptanalysis. We introduce two possible directions: guessing a set of well chosen key bits

and using well-chosen samples. We study GOST and Simon for a concrete number of rounds, discover properties inside the cipher structure which will lead to more efficient attacks. We will then demonstrate how to make these attacks work better by:

- **Inspection**: Learn what is trivial and non-trivial behavior of ElimLin algorithm by inspecting the equations found by ElimLin.
- **Prediction and interpolation**: Predict when the ElimLin algorithm will terminate and solve the problem.
- **Guess-then-solve**: With some "cost of guessing", we can reduce the solving complexity. Selecting the right set of bits to guess makes the problem easier to solve.
- **Selection of samples**: Use specific plaintexts suggested by independent well-known attacks, such as [generalized] linear attacks, truncated differential properties and cube attacks.

The second part of this thesis is about understanding how elliptic curve cryptography can be efficiently coded for fast implementation and also cryptanalysis. We discussed an open research problem for solving Elliptic Curve Discrete Logarithm Problem and also implemented a dedicated speed optimization for Bitcoin brain wallet attack. Bitcoin is a cryptocurrency that was invented in 2008 and has become extremely popular since 2012. Bitcoin users can deterministically derive the private key used for transmitting money from a password. Such wallets are known as brain wallets. Brain wallets are appealing because they free users from storing their private keys. Unfortunately, brain wallets were not designed carefully enough and allowed attackers to conduct unlimited offline password guessing. In 2015, a white hat hacker published the implementation of the brain wallet attack. The results of this attack were later published in 2016 [159]. We believe that such an attack can be made faster to make brain wallets much more vulnerable. In order to optimize the attack, we study Elliptic Curve secp256k1, which is used in Bitcoin.

We focus on the speed of the key generation process and provide the first detailed benchmarks for all the major implementations of this curve. The key generation process is a fundamental part in the Bitcoin brain wallet attack, which is also the most part cost most time to compute. Our optimized attack improves the state-of-art by a factor of 2.5 in private key checking speed.

Part I

Backgroud and Related Work

Chapter 2

Introduction to Cryptography

Cryptology is the science of hiding and recovering secret information. It is mainly divided into two research areas: the areas of cryptography and cryptanalysis. Traditionally, cryptography is the study and practice of techniques that are used to establish secure communication between two parties in the presence of unauthorized third parties, usually called adversaries or attackers. Cryptography aims to prevent the adversary from learning anything about the original content of the communication, even if the adversary has some type of access to the communication channel.

In general, if two parties would like to share some confidential information, they will share some secret information in advance. The piece of secret information will be used to transfer the original ordinary message (plaintext P) into an unintelligible message (ciphertext C) by the sender S. Additionally, C can be transferred back to P by the receiver R using the same piece of secret information. Formally, this secret information is called the key which is usually a short string of bits that needed to decrypt the ciphertext, while the transformations are called the encryption and decryption algorithms.

Cryptanalysis is the sophisticated analysis and study of the security that a given cryptographic scheme offers. It focuses on the techniques related to recovering either the original content of an encrypted message without the knowledge of the secret key or some fraction of information from the message. This analysis is performed under different scenarios related to the adversary's resources, type of access,

and the adversary's objectives. In general, the main purpose of cryptanalysis is to find the hidden weaknesses of a cryptosystem and develop a method of decryption.

In this chapter, we will give a brief introduction to two types of encryption systems: symmetric encryption systems and asymmetric encryption systems. We will discuss in detail block ciphers in symmetric cryptography, which are widely used and primarily implemented in the real world.

2.1 Symmetric and Asymmetric Encryptions

Cryptographic algorithms are classified based on how key material is used and managed. Normally, they are classified into three groups. There are keyless algorithms, which do not use any key and do not need to trust anyone. Another type of algorithm uses a shared key, which needs to trust everyone that has the key. The third type is private-public key algorithm, in which the private key is only known by one person [37].

Generally, a cryptosystem has a sender S and a receiver R who want to send messages over an insecure channel. S and R are assumed to share a small amount of information beforehand, which is called the key. A cryptosystem is an encryption scheme that aims to protect the communication between S and R over an insecure channel.

A cryptosystem often contains an encryption function E, which takes a plaintext p and a secret key K which is composed of random bits and outputs a ciphertext $c = E_K(p)$, and the decryption function D (inverse of E), which takes the ciphertext c and the secret key $K^{'}$ as input and recovers the initial plaintext, i.e. $D_{K'}(c) = p$. The cryptosystem should be designed in such a way that even when adversaries obtain ciphertext, they cannot gain any information regarding the secret key or the plaintext.

In a cryptosystem, if $K = K^{'}$ which means the same secret key is used for both encryption and decryption, then the cryptosystem is called symmetric cryptosystem.

Definition 1 (Symmetric Cryptosystem)**.** *Let P be the finite set of plaintexts, C be the finite set of ciphertexts, and K be the finite key space. An efficiently computable*

encryption function E takes one plaintext in P and a key $k \in K$ returns a ciphertext in C. We write:

$$E_k : P \to C$$

for the operation of executing E on k and P, and a corresponding efficiently computable decryption function is given byD_k:

$$D_k : C \to P$$

such that $D_k(E_k(p)) = p$ for all plaintext $p \in P$.

If the keys used for encryption and decryption are different to each other, but related in a way such that decryption of a given ciphertext c results in plaintext p, then the cryptosystem is called asymmetric cryptosystem.

Definition 2 (Asymmetric Cryptosystem)**.** *Let P be the finite set of plaintexts, C be the finite set of ciphertexts and K be the key space. An efficiently computable key generation algorithm keyGen() randomly generates a pair of public key p_k and secret key s_k; an efficiently computable encryption function E takes one $p_k \in K$ and a plaintext in P returns a cipher in C. We write:*

$$E_{p_k} : P \to C$$

and an $s_k \in K$ for the operation of executing E on p_k and P, and the corresponding efficiently computable decryption function is given by $D_{s_k} \in D$:

$$D_{s_k} : C \to P$$

such that $D_{s_k}(E_{p_k}(p)) = p$ for all plaintexts $p \in P$.

Note that symmetric cryptosystems have two algorithms: encryption and decryption. Asymmetric cryptosystems normally have at least three algorithms: key generation, encryption, and decryption. In a symmetric cryptosystem, if the key is compromised, then an adversary can decrypt any message passed from sender to

receiver and gains full control over the system. Asymmetric cryptosystems solve this problem by using different, but corresponding keys, for encryption and decryption. However, modern cryptography sometimes requires a huge number of keys that must be distributed securely.

Researchers start to solve the problem by combining both types of cryptosystem, called a *hybrid* encryption scheme. This cryptosystem combines the convenience of asymmetric with the efficiency of a symmetric cryptosystem [148]. In hybrid encryption, symmetric cryptosystem is used for encryption, while the secret key is shared using a protocol based on public-key cryptography, cf. 2.1. There are two main components of a hybrid encryption scheme, Key Encapsulation Mechanism (KEM) and Data Encapsulation Mechanism (DEM). The key feature is that the two parts are independent of one another. The framework was first formalized by Cramer and Shoup in 2003 and we refer the reader to [75] for more details.

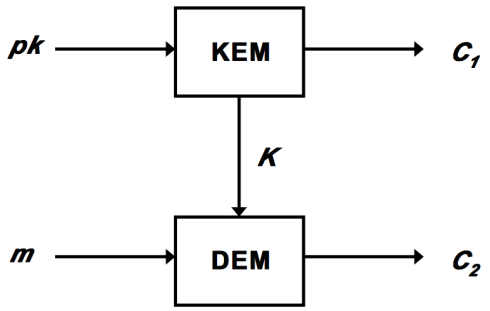


Figure 2.1: Hybrid encryption

2.2 Block Ciphers

A block cipher is a type of symmetric encryption system. In block ciphers, the plaintext is divided into blocks of a fixed length, which are then encrypted into blocks of ciphertexts using the same key. Block ciphers are deterministic algorithms, which means that the same inputs result in the same outputs [78]. Block ciphers are considered as highly secure cryptography. Normally block ciphers are designed to be used for 50 years, while asymmetric cryptosystems are usually obsolete after 10 years (for example, NSA no longer recommends NIST P-256 elliptic

curve in 2016[126]).

The efficiently computable encryption algorithm $E_K(P)$ and decryption algorithm $D_k(C)$ in a block cipher both use blocks of n-bit as input and k-bit as a key K. $D_k(C)$ is the inverse of the encryption map $E_K(P)$. More formally, we have that:

$$C = E_K(P) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

$$D_k(C) = E_K^{-1}(P) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

such that $D(E_k(P)) = P \quad \forall K \in \{0,1\}^k$.

In general, for every key, a block cipher is a permutation of the form $\{0,1\}^n \to \{0,1\}^n$ for n-bit block. So in total there are $(2^n!) \simeq (2^{n-1})^{2^n}$ possible permutations. A block cipher which operates on n-bit blocks and uses k-bit keys is equivalent to a collection of 2^k distinct permutations on n-bit. A good design of a block cipher aims to choose the 2^k permutations uniformly at random[1] from the set of all $(2^n!)$ permutations.

Historical ciphers can be divided into three groups: substitution ciphers, transposition ciphers and product ciphers.

2.2.1 Substitution Ciphers

As indicated in the name, in substitution ciphers, every character in plaintext is substituted by some ciphertext character. There are four types of substitution ciphers: simple substitution, polyalphabetic substitution, homophonic substitution and polygram substitution [114]. In this thesis, we only discuss simple and polyalphabetic substitutions:

Simple Substitution

In a simple substitution cipher, each plaint text character is transformed into a ciphertext character via the same encryption function E. More formally, let $P = p_0, ..., p_{n-1}$ be an n-character plaintext and $C = c_0, ...c_{n-1}$ be a ciphertext, $\forall i : 0 \leq i < n$

$$E : P \to C$$

[1] Or it is impossible to see if it was otherwise.

$$c_i = f(p_i)$$

Around 50 BC, Julius Caesar wrote to Marcus Cicero using a cipher that encrypted messages by shifting every letter in the plaintext three positions to the right in the alphabet. This cipher is based on *shifted alphabets*. For the Caesar cipher, the secret key k is $+3 \mod 26$. In general, the cipher is easily broken by shifting the ciphertexts one position until the plaintext arises.

Polyalphabetic Substitution

In a polyalphabetic substitution, the characters in plaintext are transformed into ciphertext using a j-character key $K = k_0, k_1, ...k_{j-1}$, which defines j distinct encryption functions $E_{k_0}, E_{k_1}, ..., E_{k_{j-1}}$. In this case, $\forall i : 0 \leq i < n$

$$E_{k_l} : P \rightarrow C \qquad \forall l : 0 \leq l < j$$

$$c_i = E_{k_{i \bmod j}}(p_i)$$

The Vigenère cipher [136], first published in 1586, uses polyalphabetic substitution and is defined as follows:

$$c_i = E_{k_{i \bmod j}}(p_i) = p_i + k_{i \bmod j}$$

2.2.2 Transposition Systems

Transposition systems are essentially permutations of the characters in plaintext. Therefore, a transposition cipher is defined as follows $\forall i : 0 \leq i < n$

$$\eta : \{0, ..., (n-1)\} \rightarrow \{0, ..., (n-1)\}, \textit{ a permutation}$$

$$c_i = E(p_i) = p_{\eta(i)}$$

Many transposition ciphers operate by blocks which permute characters with a fixed period j. In that case:

$$\eta : \{0, ..., (j-1)\} \rightarrow \{0, ..., (j-1)\}, \textit{ a permutation}$$

$$c_i = E(p_i) = p_{(i \ div \ j)+\eta(i \ mod \ j)}$$

The Vigenère and in general substitution ciphers can be broken when enough ciphertext is available to the cryptanalyst using the index of coincidence, Kasiski's method, etc. [77, 136, 107]. Transposition ciphers can be broken using the frequency distributions for bigrams, trigrams, and N-grams [77, 136, 107]. This knowledge about natural language is also very useful for our later work in password cracking.

2.2.3 Product Ciphers

To produce much stronger ciphers than the ones we have currently seen, we can combine substitution and transposition ciphers. These ciphers are called *product ciphers*. Most block ciphers that are still used today are product ciphers. An iterated cipher is one kind of product ciphers in which the ciphertexts are computed by iteratively applying a round function several times to the plaintext. In each round, a round key is combined with the text input.

Definition 3. *In an r-**round iterated block cipher**, the ciphertext is computed by iteratively applying a round function g to the plaintext, such that*

$$C_i = g(C_{i-1}, K_i),\ i = 1, ..., r$$

where C_0 is the plaintext, K_i is a round key, and C_r is the ciphertext. Decryption is done by reversing the above function. Therefore, for a fixed key K_i, g must be invertible when K_i is fixed.

Feistel Ciphers

In general, it is not easy to make an invertible function that makes the encryption and decryption process identical. One method was created by the German physicist and cryptographer Horst Feistel, who was a pioneer in this area while working for American IBM. Feistel, together with Don Coppersmith, introduced the concept of Feistel networks while working on IBM's "Lucifer" cipher in 1973 [90]. Their work gained the respect of the United States Federal Government who adapted it to the

Data Encryption Standard (DES), which is based on the Lucifer project with some changes done by the NSA [132].

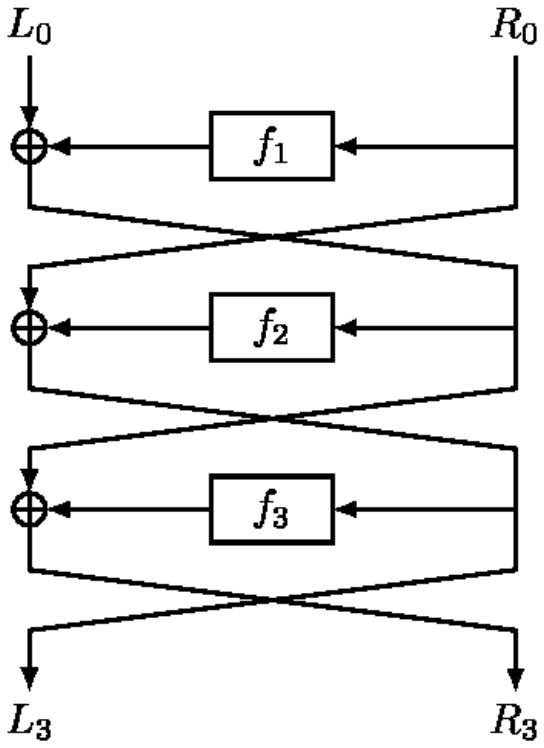


Figure 2.2: Feistel network

Definition 4 (Feistel Network, cf. Figure 2.2)**.** *A Feistel cipher is an iterated cipher that maps a 2t-bit plaintext block (L_0,R_0) where L_0 and R_0 are the left and right t-bit halves respectively, to a 2t-bit block (L_r,R_r) after r-rounds of encryption.*

The result of i-rounds encryption $\forall i : 1 \leq i < r-1$ is computed as follows:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_{i-1})$$

where K_i is the ith subkey derived from the secret key K and f the one-round function which takes a subkey and a t-block as input to map into another t-bit block. This process is iteratively applied for $r-2$ rounds. In the last round, there is no swap between two halves L_{i-1} and R_{i-1}. This makes the decryption of the Feistel network the same as the encryption process, only requires a reversal of the key

schedule. The final output is given by the following:

$$L_r = L_{r-1} \oplus f(R_{r-1}, K_{r-1})$$

$$R_r = R_{r-1}$$

Since the encryption and decryption processes are identical (except for the key order), the software and hardware implementation of Feistel ciphers are much easier.

2.2.4 Courtois Toy Cipher

Chapter 3

Cryptanalysis of Block Ciphers

The history of cryptanalysis is as long and as fascinating as the history of cryptography. For example, in 1917, an article in "Scientific American" claimed that the Vigenère cipher was "impossible of translation" [114]. Today, most cryptography classes at university use the Vigenère cipher as an exercise to illustrate that this claim is not true. When discussing the security of a cryptosystem, one needs to define a model that works in the real world for which we will use the model of Shannon [146].

The sender and the receiver share a common key K over a secure channel in advance. The sender encrypts a plaintext P using the secret key K, sends ciphertext C over an insecure channel to the receiver. The receiver then decrypts C to P using K. The attacker has access to the insecure channel and can intercept the ciphertext. In this chapter we assume that the sender and receiver use a secret key cipher $E_K(\cdot)$ of n-bits block size and k-bits size of key K.

The definition of security is also strongly related to what type of attackers you are defending against. In chapter 5 of Bruce Schneier's book [139], the author categorizes (also appear in slides 35 in [45]) attackers along the following basic lines:

1. Adversarial goals and motivations.

2. Resources: money, human resources, computing power, memory, risk, expertise, etc.

3. Access to the system.

A system could be secure against one type of attackers but not secure to another type. This reflects on the different types of security assumptions and classification of attacks which we will discuss later.

To evaluate the security of a cipher we assume:

Assumption 1. *All keys are equally likely and a key K is chosen uniformly at random.*

Also we will assume that the attacker knows all the details about the cryptographic algorithm used by the sender and receiver, except the secret key. This assumption is known as Kerckhoffs's assumption [107].

Assumption 2.A. *The enemy cryptanalyst knows all details of the enciphering process and deciphering process except for the value of the secret key.*

This is not quite realistic if we look at the history of cryptography and cryptanalysis. As an extension of Assumption 2.A, we discuss the following two assumptions.

Assumption 2.B. *The enemy cryptanalyst knows all the details of the enciphering process and deciphering process except the value of the key and the Substitution-box (S-box) which is a core component of symmetric key algorithms that performs substitution.*

Under this assumption, the S-boxes are like the master key or a high-level key, which is kept secret at a certain parameter (e.g., banks or country) and can be computed by the enemy, but at a high cost. This is similar to the rotors used in the German Enigma machine during World War II. The attacker has to recover the S-boxes first through silicon reverse engineering or try for different sets of known S-boxes, then perform normal cryptanalysis as in Assumption 2.A.

Assumption 2.C. *The enemy cryptanalyst knows all the details of the enciphering and deciphering processes except that the cipher has been tweaked; for example 90% of the cipher is what we know.*

For example, the encryption algorithm used in Chinese Sim cards is known as COMP128-V0, which is defined by China Mobile and only used in China. In this case, the attacker has again two steps under this assumption: first to recover the cipher through reverse engineering and try different sets of known S-boxes; then performs normal cryptanalysis as Assumption 2.A.

3.1 Classification of Attacks

Regarding access to the system, Schneier classifies the possible attacks an attacker can perform as follows: [137]

1. **Ciphertext-only Attack**: The attacker processes a set of intercepted ciphertexts.

2. **Known Plaintext Attack**: The attacker obtains a set of s plaintexts $P_1, P_2, ..., P_s$, and the corresponding ciphertexts $C_1, C_2, ..., C_s$.

3. **Chosen Plaintext Attack**: The attacker chooses a prior set of s plaintexts $P_1, P_2, ..., P_s$ and obtains, in some way, the corresponding ciphertexts $C_1, C_2, ..., C_s$.

4. **Adaptively Chosen Plaintext Attack**: The attacker chooses a set of s plaintext $P_1, P_2, ..., P_s$ interactively as he obtains the corresponding ciphertext $C_1, C_2, ..., C_s$. That is, the attacker chooses P_1, obtains C_1, and then chooses P_2 etc.

5. **Chosen Ciphertext Attacks**: For symmetric ciphers, Chosen Ciphertext Attacks are similar to the Chosen Plaintext Attack and the Adaptively Chosen Plaintext Attack.

6. **Chosen Plaintext and Ciphertext Attack**: The attacker has access to both encryption and decryption oracles and can choose both plaintext and ciphertext.

The chosen text attacks are the most powerful attacks. However, they are also unrealistic in many applications. If redundancy exists in plaintext space, it will

be very hard for an attacker to find an encrypted non-meaningful plaintext sent by the sender, and to get the ciphertexts decrypted. However, if a system is secure against an Adaptively Chosen Plaintext Attack, then it is also secure against all other attacks.

Modern cryptanalysis of block ciphers has focused on finding the secret key K for any scenario, even unrealistic ones. Importantly, there are other serious attacks for public key cryptography, which do not find the secret key. In [113], Knudsen classifies the types of attacks as follows:

1. **Total break**: An attacker finds the secret key K.
2. **Global deduction**: An attacker finds an algorithm A, which is functionally equivalent to $E_K(\cdot)(or\ D_K(\cdot))$ without knowing the key K.
3. **Instance (local) deduction**: An attacker finds the plaintext (ciphertext) of an intercepted ciphertext (plaintext), which was not obtained from the legitimate sender.
4. **Information deduction**: An attacker gains some information about the key, plaintexts or ciphertexts, which was not obtained directly from the sender or before the attack.

This classification is hierarchical, i.e., if a total break is possible, then a global deduction is possible, etc.

Data Requirements

Attacks can also be characterized by the resources they require. These resources include time complexity, memory usage, and data requirements. Although the first two types of resources are very obvious, it is worth pointing out the data requirement is also a key component that makes an attack able to work in practice. Full plaintext and ciphertext pairs are not easy to obtain in the real world. Some of the attacks work only if all possible plaintext and ciphertext pairs for a single key are known which is called single key scenario [21], and some of the attacks (which have only appeared in recent years) are specifically designed to take advantage of a so-called multiple key scenario, where the attacker has access to data encrypted with

more than one key (such attacks typically also work in standard single key scenario, however in this case they will be less advantageous) [42, 61].

3.2 Brute-force Attacks

A brute-force attack or exhaustive key search is the most general attack that can be applied to any block cipher. All block ciphers are totally breakable in a ciphertext-only attack, just simply by trying all the possible keys one by one, and checking whether the computed plaintext is meaningful. If the block size is equal to key size, in the worst case, this attack requires the computation of 2^k encryptions. The dimension of the key space k, which is the length of the key, determines the practical feasibility of performing a brute-force attack. While in 2014 an paper written by Huang and Lai [102] described a brute-force-like attack, with a time complexity faster than the exhaustive key search by going over the entire key space, but performing less than a full encryption for each possible key. The time complexity of this attack is $2^k(1-\varepsilon)$ (where $\varepsilon > 0$) on average regardless the block size, cf. [102, 42]. For modern block ciphers, brute-force is always possible in theory, but computationally infeasible in practice.

3.3 Linear Cryptanalysis

Linear cryptanalysis is a Known Plaintext Attack on block ciphers. It was popularised by Matsui in 1993 [120] and invented earlier by Gilbert and his student [158]. A preliminary version of the attack on FEAL was described in 1992 [121]. A year later, Matsui published another attack on DES[120]. Although the published attack on DES required 2^{43} known plaintexts which is not very practical, it still was a great improvement in experimental cryptanalysis.

Linear cryptanalysis is based on finding affine approximations to the action of a cipher which holds with high probability. The attacker exploits linear approximations of some bits of the plaintext, ciphertext and key. In the attack on the DES (or on DES-like iterated ciphers) the linear approximations are obtained by combining approximations for each round under the assumption of independent round keys.

The attacker hopes to find an expression (equation 3.1), which holds with prob-

ability $p_L \neq \frac{1}{2}$ over the keys [120], such that $\varepsilon = \mid p_L - \frac{1}{2} \mid$, called the bias, is maximal.

$$(P \cdot \alpha) \oplus (C \cdot \beta) = (K \cdot \gamma) \tag{3.1}$$

where $P, C, \alpha, \beta, \gamma$ are m-bit strings and where '$\cdot$' denotes the dot product.

Given an approximation (equation 3.1) a linear attack using N plaintexts and the N corresponding ciphertexts goes as follows[120].

1. for all plaintexts, P, and ciphertexts, C, let T be the number of times the left hand side of equation 3.1 is 0.

2. if $T > \frac{N}{2}$ guess that $K \cdot \gamma = 0$, otherwise guess that $K \cdot \gamma = 1$ (majority vote).

By using the above method, the attacker can find one bit of information about the secret key, $K \cdot \gamma$. However, this is only one step in an attack, as it only finds one bit or a linear equation about the key. In [120], Matsui also showed an extended linear attack which finds more key bits. Instead of approximating the first and last round, the extended linear attack simply repeats the attack for all values of the relevant key bits in those two rounds by using the following approximation equation:

$$(P \cdot \alpha) \oplus (C \cdot \beta) \oplus (F(P_R, K_1) \cdot \alpha_1) \oplus (F(C_R, K_r) \cdot \alpha_r) = (K \cdot \gamma) \tag{3.2}$$

where P_R, C_R are the right halves of the plaintexts and ciphertexts. K_1 and K_R are the key bits affecting the linear approximation in the first and rth rounds. We refer the reader to Matsui's paper [120] for more details.

Kaliski and Robshaw showed an improved linear attack using multiple linear approximations in [108]. In [110] Knudsen and Robshaw introduced a linear attack using non-linear approximations in the outer rounds of a block cipher. Both of these have not been able to show any significant improvement compared to Matsui's linear attack. Kaliski's and Knudsen's attacks seem best suited for ciphers with large S-boxes, such as LOKI [28, 110]. However in 2004 as an extension of Knudsen and Robshaw's work, new attacks were introduced by Courtois [57] to attack Feistel schemes, which makes this non-linear approximation "penetrate" inside the cipher

and achieved a small improvement upon Matsui's work.

3.4 Differential Cryptanalysis

Differential cryptanalysis is based on tracking changes in the differences between two messages as they pass through the consecutive rounds of encryption. It is one of the oldest classical attacks on modern block ciphers. In cryptographic literature, it was first described and analyzed by Biham and Shamir, and applied to the DES algorithm in the early 1990s [18]. However, Coppersmith, a member of the IBM team that designed DES [34, 35, 66], reported that this attack was already known to IBM designers around 1974. It was known under the name of T-attack or Tickle attack, and DES had already been designed to resist this type of attack. A detailed discussion of the specific original design criteria of DES can be found in [66]. Moreover, it appears that IBM had agreed with the NSA that the design criteria of DES should not be made public, precisely because it would "weaken the competitive advantage the United States enjoyed over other countries in the field of cryptography" cf. [34, 35]. In contrast, linear cryptanalysis seems more recent.

Today, differential cryptanalysis is extremely well known. Numerous authors studied various aspects of differential cryptanalysis extensively in the 1990s. Apart from DES, differential cryptanalysis has also been successfully applied to a wide range of iterated ciphers [112, 119]. Recent research work showed great success using differential cryptanalysis to break the Russian standard GOST [69, 51] and the NSA lightweight cipher Simon and Speck [19, 4].

Both linear and differential cryptanalysis require lots of data; however a major difference between these two type of attacks is that differential cryptanalysis works in a so-called multiple key scenario. In a multiple key scenario, where given 2^X devices with distinct keys, 2^Y of data per device, and $M = 2^Z$ of memory, some keys can be recovered at the total cost of $T = 2^Z$ [61]. This also appears in the original Biham and Shamir paper in 1993 [18]. But this attack scenario has only been discussed in recent years [61]. The fact that differential cryptanalysis works with data collected from multiple keys (where linear cryptanalysis requires data on a

single key) makes differential cryptanalysis more practical than linear cryptanalysis, as it's easier for an attacker to get data from multiple keys than a single key in practice.

The main task of differential cryptanalysis is to studies the propagation of input differences from round to round inside the encryption system, and to find specific differences that propagate with high probability. Such plaintext and ciphertext pair (P/C pairs) can be used to recover some bits of the secret key. In general, differential cryptanalysis exposes the non-uniform distribution of the output differences given one or several input differences. However, what really matters in differential cryptanalysis is that some special events can happen at least once within a given attack requirements.

Definition 5. *A **difference** between two bit strings, X and $X^{'}$ of equal length is defined as*

$$\triangle X = X \otimes (X^{'})^{-1}$$

where $\otimes$ is the group operation on the group of bit strings used to combine the key with the text input in the round function and $(X)^{-1}$ is the inverse element of X with respect to $\otimes$

Generally, the selection of the operator $\otimes$ depends on the way the round subkeys are introduced in each round. As many ciphers use XOR for the key application in round function, the operator in Definition 5 is usually exclusive-or ($\oplus$).

The attacker then computes the differences of the corresponding ciphertexts, hoping to detect statistical patterns in their distribution. The resulting pair of differences is called a differential. Their statistical properties depend upon the internal structure of the cipher. The attacker aims to find one particular ciphertext difference that is especially frequent. In this way, the cipher can be distinguished from random.

Some differential cryptanalysis work study multiple input / output differences, such as truncated differentials (which we will discuss later), and the work presented in 2009 by Courtois [58] for cloning MiFare Classic building passes.

Truncated Differentials

Truncated Differential Cryptanalysis is a generalization of differential cryptanalysis developed by Lars Knudsen in 1995 [111]. Unlike differential cryptanalysis, which studies the propagation of the full difference between two plaintexts, truncated differential cryptanalysis considers differences that are partially determined, since in some ciphers it is possible and advantageous to predict the values of parts of the differences after each round. This technique has been successfully applied to many block ciphers, such as Camellia [154], TEA and XTEA [101], and Russian standard GOST [59, 123, 71]. Truncated differential cryptanalysis has also been extensively studied by Mourouzis in his PhD thesis [123]. Truncated differential is defined as follows [111].

Definition 6. *Let $a = a_0a_1...a_{n-1}$ be an n-bit string, then its truncation is the n-bit string b given by $b_0b_1...b_{n-1} = TRUNC(a_0a_1...a_{n-1})$, where either $b_i = a_i$ or $b_i = *$, for all $0 \leq i \leq n-1$ and $*$ is an unknown value*

Definition 7. *A differential that predicts only parts of a n-bit value is called a truncated differential. More formally, let (a,b) be an i-round differential. If $a^{'}$ is a truncation of a and $b^{'}$ is a truncation of b, then $(a^{'},b^{'})$ is called an i-round truncated differential.*

A truncated differential can be considered as a collection of differentials. For example, for an i-round 64-bit block cipher, a truncated differential $\Delta =$ [0000022200000080] in hexadecimal (which appears later in Section 6.2) is a collection of $2^4 - 1$ differentials (a,b) where a and b can take any of the following values:

00000200 00000000	00000020 00000000	00000002 00000000
00000000 00000080	00000220 00000000	00000202 00000000
00000200 00000080	00000022 00000000	00000020 00000080
00000002 00000080	00000222 00000000	00000220 00000080
00000202 00000080	00000022 00000080	00000222 00000080

3.5 Algebraic Attacks

In general, the security of a given block cipher will grow exponentially with the number of rounds and so does the number of required Plaintext/Ciphertext pairs that are needed in a linear and differential cryptanalytic attack. However, in most cases, only a few Plaintext/Ciphertext pairs are available for cryptanalysis, so linear or differential attacks are not expected to succeed. Thus, a new method needs to be invented that will be able to recover the secret key when only a very limited amount of data is available..

Claude Shannon, once suggested that the security of a cipher should be related to the difficulty of solving the underlying system of equations and deriving the key:

> *"if we could show that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type, then we would have a lower bound of sorts for the work characteristic"* [145].

This is the core concept behind algebraic attacks. An algebraic attack is a form of Known Plaintext Attack which consists of the following two steps:

1. **Modelling**: Describe the cipher as a multivariate system of polynomial equations over a small finite field (like $\mathbb{F}_2$) or logical constraints in terms of the secret key K, the plaintext P and the ciphertext C.

$$f_1(K,P,C) = 0, \ldots f_r(K,P,C) = 0 \Longleftrightarrow E(K,P) = C$$

2. **Solving**: Solve the underlying multivariate system of equations and obtain the secret key. In order to reduce the complexity of solving the system, we substitute all known Plaintext/Ciphertext pairs. The more pairs that are substituted, the more the equations we obtain involving the key bits. For many known attacks, this can make the problem easier to solve.

Generally, each module of a block cipher can be described with a set of algebraic equations. By putting these algebraic equations together, we can get a large

precise system representing the whole cipher. The main idea of an algebraic attack is to establish a series of low complexity algebraic equations of initial plaintext and ciphertext, and then find the secret key by solving the equations. Such equation systems are normally very large systems of quadratic multivariate polynomial equations over $\mathbb{F}_2$. Each variable represents a state-bit of the encryption algorithm. Then the variables that represent state-bits from the initial round are set according to the corresponding actual values of the plaintext; similarly variables that represent state-bits from the last round are set according to the corresponding actual values of the bits of the ciphertext. Therefore, the security of a block cipher depends on whether an efficient algorithm exists to solve such large and sparse multivariate polynomial equations. If we can solve the equation system faster than an exhaustive key search attack, the cipher will be broken (in an academic sense).

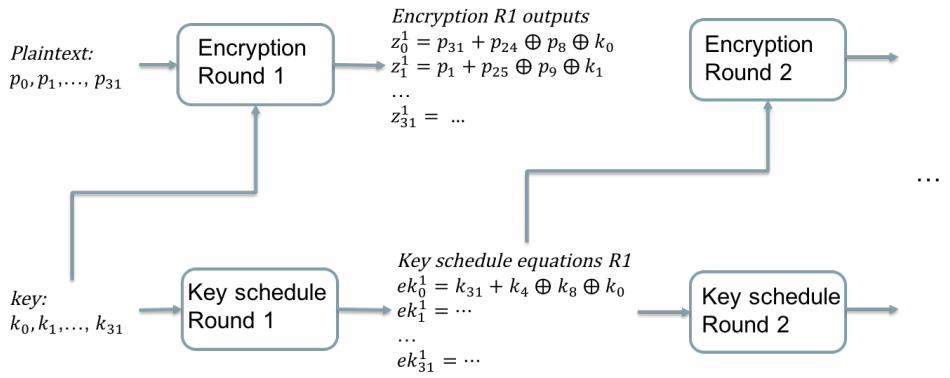


Figure 3.1: Toy example for modelling Simon block cipher with a multivariate quadratic equation system. The upper part is the main block encryption with extended keys generated by a key schedule[1](lower part).

In addition, some high profile cryptanalysis problems in public key cryptography can be written in a form that contains a *block cipher topology* [2]. Such equations have a similar structure to a block cipher, where the beginning inputs and final out-

[1]The key schedule here has a specific recursive form in popular ciphers such as DES, AES and Simon which optimizes storage or chip size and timing.

[2] This is basically a property of the equations proposed by Semaev in 2015 [140, 142], they have the same structure as on Figure 3.2 with variables progressively more and more remote from the constraints which make that the system of equations have a unique solution. The key point is that this sort of configuration leads to a system of equations which is really very hard to solve in the same way as in algebraic attacks on block ciphers.

puts variables are easy to get and the middle part of the equation system is very hard to analyse (see Figure 3.2). One example is Semaev's summation polynomial equations studied in elliptic curve cryptanalysis [140, 142] see Section 4.5.2.

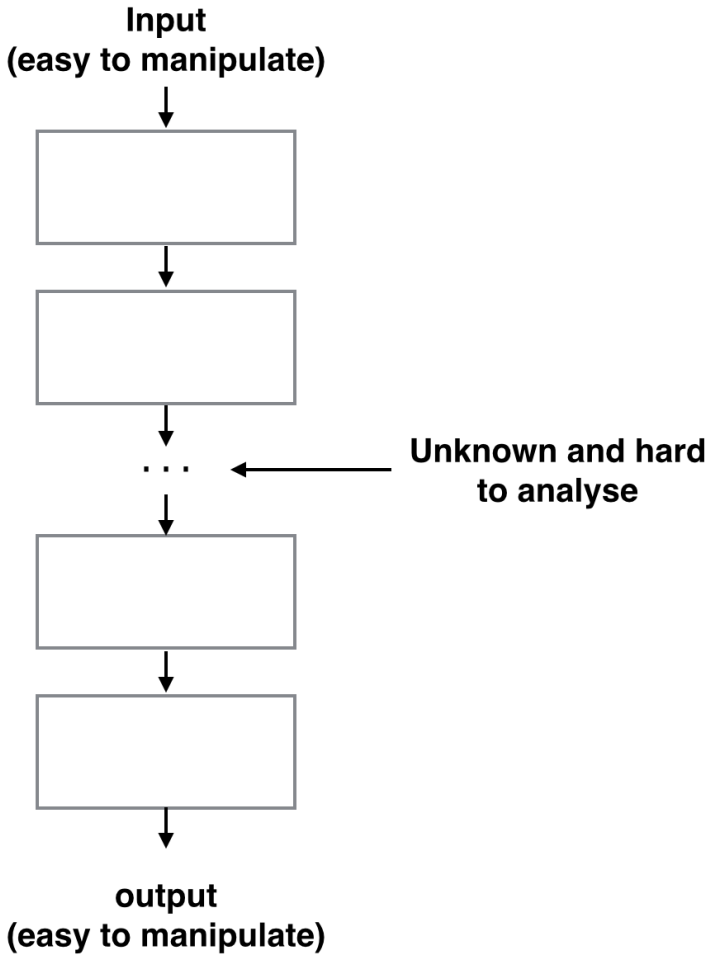


Figure 3.2: Block cipher topology: the attacker can control or manipulate the inputs and the outputs but it is quite hard to say anything about the variables in the middle

3.5.1 Algebraic Attacks Solving Stage

Solving a random system of multivariate non-linear boolean equations is an NP-hard problem [92]. As many cryptographic primitives can be described by a sparse multivariate non-linear system of equations over $\mathbb{F}_2$ or any other algebraic systems, several techniques were developed to tackle the problem of solving these equations. A classic approach is to use techniques from algebraic geometry, especially Gröbner

bases algorithms, to solve the system of equations [86]. But most of the time they do not lead to solutions in practice due to the extremely high memory requirements. Subsequently, some heuristic techniques were developed called *linearization* [50], where all the non-linear terms are replaced by an independent linear variable and the resulting linear system can be solved using Gaussian elimination [144]. However, this requires there to be enough linearly independent equations and that the initial system be highly over-defined and sparse. Then, the XL algorithm [50, 74] was developed to make the system over-defined by adding new equations to the current system. The XL proposal made it possible to solve the multi-order non-linear equations within polynomial time, which accelerated the development of the algebraic attacks. Then researchers focused mainly on fast solutions to the algebraic equation systems.

Since 2006, Courtois and Bard discovered that this problem could be solved using tools and software [64], such as SAT solvers. SAT solvers are automated software solvers which aim to solve one of the original NP-complete problems, the so-called Boolean Satisfiability Problem.

Definition 8. *[Boolean Satisfiability Problem] The SAT problem in Conjunctive Normal Form (CNF) consists of the conjunction ($\wedge$ representing the Boolean AND connective) of a number of clauses, where a clause is a disjunction ($\vee$ representing the Boolean OR connective) of a number of propositions or their negations (literals).*

If x_i represent propositions that can assume only the values True ($\equiv 1 \equiv \top$) or False ($\equiv 0 \equiv \bot$), then an example formula in CNF would be:

$$(x0 \vee x2 \vee x3) \wedge (x3) \wedge (x1 \vee \neg x2)$$

where $\neg x_i$ is the negation of x_i. Given a set of clauses $C_0, C_1, ..., C_{m-1}$ on the propositions $x_0, x_1, ..., x_{n-1}$, the problem is to determine whether the formula

$$F = \bigwedge_{j<m} C_j$$

has an assignment of truth values to the propositions such that it evaluates to True.

In the past decade, the rapid improvements in SAT algorithms has made SAT solvers increasingly popular. Modern SAT solvers have a significant impact on the fields of electronic design automation, software verification, constraint solving in artificial intelligence, and operations research, among others. Nicolas Courtois is the pioneer of bringing SAT solvers into action in the area of symmetric cryptanalysis. In his paper [10], the authors described how to convert a multivariate quadratic equation system to CNF which can be solved automatically by SAT solvers. The advantage of this technique is that SAT solvers can perform reasonably well and do not require a lot of memory as compared with Gröbner basis-based techniques [87]. The only disadvantage is the unpredictability of its complexity. The first algebraic attack on reduced-round block cipher DES was done by Courtois and Bard in [47]. Later in 2008, the first algebraic attack on full block cipher KeeLoq [65] took place. In the past 10 years, SAT solvers have been used for attacks on block ciphers, such as GOST[68, 42], KATAN32 [11] and the Chinese block cipher SMS4 [84], stream ciphers, such as Crypto-1, HiTag2 and Bivium [150, 72], and MiFare Classic smart cards [55].

Another method is to use the ElimLin algorithm [56]. ElimLin stands for **Elim**inate **Lin**ear, and it is a simple algorithm for solving polynomial systems of multivariate equations over small finite fields. It was initially proposed as a single tool by Courtois to attack DES and CTC/CTC2 ciphers [47]. It is also known as the *inter-reduction* step in all major algebra systems. Its main aim is to reveal some hidden linear equations existing in the ideal generated by the system of polynomials. ElimLin is composed of two sequential stages:

- **Gaussian Elimination:** To discover all the linear equations in the linear span of initial equations.
- **Substitution:** Variables are iteratively eliminated in the whole system based on the linear equations found until no linear equation is left.

Given an initial multivariate system of equations over S^0 in $\mathbb{F}_2[x_1, x_2, .., x_n]$,

then the ElimLin is formally described in algorithm 1.

Algorithm 1 ElimLin Algorithm

Input: $S^0 = \{f_1, f_2, ..., f_m\} \in \mathbb{F}_2[x_1, x_2, .., x_n]$
Output: An updated system of equations S^T and a system of linear equations S_L
1. Set $S_L \leftarrow \emptyset$ and $S^T \leftarrow S^0$ and $k \leftarrow 1$
2. **Repeat**
For some ordering of equations and monomials perform $Gauss(S^T)$ to eliminate non-linear monomials
Set $S_{L'} \leftarrow$ Linear Equations from $Gauss(S^T)$
Set $S^T \leftarrow Gauss(S^T) \backslash S_{L'}$
for $\forall l \in S_{L'}$, l non-trivial (if l=1and unsolvable then *terminate*) **do**
  Let x_{i_k} be a monomial in l
  Substitute x_{i_k} in S^T and $S_{L'}$ and replace by $l - x_{i_k}$
  Insert l in S_L
end for
$k \leftarrow k + 1$

The study of ElimLin is interesting [155, 156] precisely because it is simpler to understand than more complex polynomial algebra techniques [50, 73, 163, 12, 100]. In recent years, ElimLin has been applied to NSA block cipher Simon, LBlock, KATAN32 and PRESENT [54, 133, 156, 124]. The main characteristic of ElimLin is that it quietly dissolves and makes non-linear equations disappear and generates linear equations. Non-linearity is the main and only thing which makes cryptographic schemes not broken by simple linear algebra. Intuitively, ElimLin seems to work better in cases where there is low non-linearity, since this implies the existence of more linear equations. Multiplicative complexity (MC) is another notion of non-linearity which was studied in [22, 52, 123] and possibly Elimlin may work sufficiently well in cryptographic primitives with low MC. However, the complexity of ElimLin attack or software algebraic attacks in general is not well studied. It is not clear why this works and how well the ElimLin attack scales for larger systems of equations. This is a major topic of interest in this thesis.

3.5.2 Connection with other Cryptanalysis Techniques

Many researchers consider algebraic cryptanalysis as an independent cryptanalysis research area called deterministic cryptanalysis, while differential and linear cryptanalysis methods are called probabilistic methods in cryptography [128]. However,

we argue that there are a lot of connections between algebraic cryptanalysis, linear and differential cryptanalysis. In Figure 3.3, we describe our view of the connections between these cryptanalytic techniques.

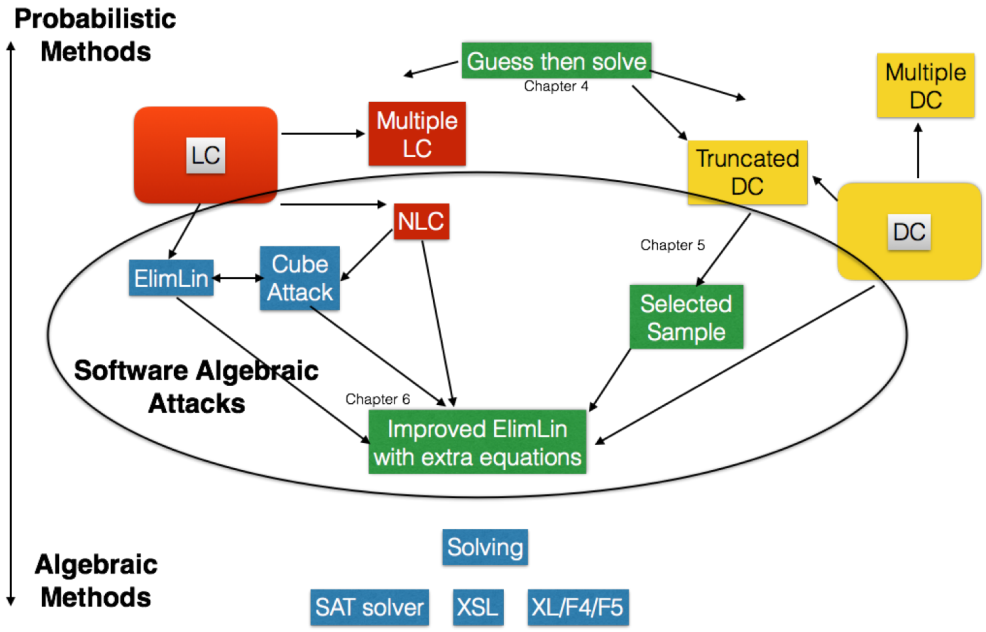


Figure 3.3: Classification of cryptanalysis methods and their connections. Most of the existing literature considered algebraic cryptanalysis as an independent research area compare to probablisitic cryptanalysis methods such as differencial cryptanalysis (DC) and linear cryptanalysis (LC). We believe Algebraic Attacks has many connections. probablisitic method are often used together with Algebraic methods and in some attacks presented in this thesis they improved Algebraic Attacks performance by introduction new equations to the mathematic model and reduced solving complexity at the solving stage of Algebraic Attacks. Boxes colored in red are normally considered as LC. Yellow boxes are normally considered as DC methods. Green boxes are the parts included in this thesis and blue boxes are considered as specific solving methods

A link between ElimLin and linear cryptanalysis in Figure 3.3 reflects the fact that both of these methods are looking for linear relations inside the cipher. However, a major difference between these two approaches is ElimLin looks for equations that work only for a particular set of P/C pairs[3]. Cube attack which was introduced by Dinur and Shamir in 2009 [82] chooses a subset of 'public' input bits such that the sum of an output bit value is a linear combination of the key after a certain summation over a cube composed of plaintexts where a subset of bits varies.

[3]although some equations might exist for any P/C pairs

Cube attack can sometimes be linear and sometimes non-linear which can be seen as related to both ElimLin and Non-Linear Cryptanalysis (NLC) sometimes called Generalized Linear Cryptanalysis (GLC). In [157] the authors have studied how well-selected samples from cube attack can be used to improve the performance of ElimLin. Our research in general more or less accidentally identifies some of the equations which are also found in cube attacks and many more. One key focus is how different cryptanalysis techniques can be used to improve software algebraic attacks by adding more equations which makes that these attacks themselves will find yet more equations (amplification effect). In Chapter 6 we show how well-selected samples following a truncated differential property can reduce the solving complexity of ElimLin and SAT solvers. Guess-then-solve method is a general cryptanalysis method which can be used in many attack scenarios. In Chapter 5 we will explore this type of attacks on Russian GOST cipher and show this method can be used to improve algebraic cryptanalysis with SAT solvers.

3.5.3 On Complexity of Algebraic Attacks

Algebraic cryptanalysis attacks allow the cryptanalyst to recover secret key bits given only one or very few plaintext / ciphertext pairs. However, one of the fundamental problems of algebraic cryptanalysis is that the runtime of algebraic attacks against block ciphers is not well understood. In 2007, Courtois and Bard said in the first paper describing an algebraic attack on block cipher:

> "Very little is known about what approach would make an algebraic attack efficient and why."

Up to today, this question still remains in algebraic cryptanalysis. This is the main motivation of my research in Chapter 7 and our (partial) answer is to study the asymptotic growth in the equations generated.

3.5.4 Algebraic Complexity Reduction

In order to break a full cipher, algebraic attacks are normally combined with other cryptanalysis techniques to reduce the solving complexity. This attack scenario consists of two independent tasks: one is how a reduced-round cipher can be solved

by software algebraic cryptanalysis which we will discuss in Chapter 5-7. The other task is called algebraic complexity reduction [60, 42], which focuses on how the complexity of solving a full round cipher can be reduced to a problem of breaking a cipher with much fewer rounds. Algebraic complexity reduction raises an important optimization problem in algebraic cryptanalysis: one needs to minimize the costs (regarding the probability that our assumptions hold) and to maximize the benefits (regarding the number and the complexity of interesting relations which hold under these assumptions). Amplification which was introduced by Courtois and Debraize in 2008 [67] is a notion which occurs in such optimization problems.

Definition 9 (Amplification, Informal)**.** *The goal of the attacker is to find a reduction where he makes some assumptions at a certain initial cost. For example they are true with probability* 2^{-X} *or work for certain proportion* 2^{-Z} *of keys. Then the attacker can in constant time determine many other internal bits inside the cipher to the total of* Y *bits.*

We are only interested in cases in which the values X *and* Z *are judged realistic for a given attack, for example* $Z < 32$ *and* $X < 128$*.*

We call amplification the ratio $A = Y/X$*.*

The idea of amplification is to gain additional information inside the cipher with low cost. Amplification is also a general cryptanalysis principle which can apply to many cryptanalysis attacks. For example, a guess-then-determine process can be seen as a form of amplification. With the cost of guessing, attackers gain additional information about the key bits which might lead to knowledge about many other bits inside the cipher. Gordon Welchman's diagonal board is also a form of amplification which makes Turing's Bombe machine gain additional information about Enigma encryption settings, cf. [36].

Algebraic complexity reduction can be very successful if the cipher has certain special properties. For example, in [42], Courtois found the Russian cipher GOST had many self-similarity properties, which reduced the problem of breaking the full GOST cipher from 2^{64} key pairs for 32 rounds to 4 key pairs for 8 rounds and similar results. In chapter 4, we will explore the idea of amplification in 8 rounds

GOST with a software algebraic attack as the last solving step.

3.6 Cryptanlysis of GOST Block Cipher

The Russian encryption standard GOST 28147-89 is an important government standard [96]. Its large key size of 256 bits makes GOST a plausible alternative for AES-256 and 3-key triple DES. This indicates that GOST means to be a serious cipher for serious applications and at least two sets of GOST S-boxes have been explicitly identified as being used by the most prominent Russian banks, cf. [138, 1].

3.6.1 GOST And ISO Standardisation.

The cost of cryptography is still an important problem for the industry. For example it was only around 2010 that Intel implemented an encryption algorithm in some of its CPUs, and nowadays both Intel and AMD have very good support for AES-NI instructions. It is therefore very important to notice that, in addition to the very long bit keys, GOST has a much lower implementation cost than AES or any other comparable encryption algorithm. For example, in hardware GOST 256 bits requires less than 800 GE[4], while AES-128 requires 3100 GE [131]. Thus it is not surprising that GOST became an Internet standard. It is part of many crypto libraries such as OpenSSL [1], and is also increasingly popular outside its country of origin [131]. It is hard to think about a better algorithm for the industry because of its ultra-low implementation cost and 20 years of cryptanalysis efforts behind it [131]. In 2010 GOST was submitted to ISO 18033 to become a worldwide encryption standard. Less than 10 block ciphers have ever become an ISO standard. Unhappily in 2011 several key recovery attacks on GOST were found [105, 60, 42, 69, 59].

3.6.2 Cryptanalysis of GOST

The turning point in the security of GOST was the discovery of the so called "Reflection" property described in [109]. Initially at Indocrypt 2008 only a weak-key attack with time complexity of 2^{192} was proposed, with a large proportion of 2^{-32} of weak keys. Then in 2011 several attacks on regular GOST keys were discovered,

[4]GE: (informally) 1 GE is equivalent to 1 AND gate

and more than half of these new attacks use this reflection property [105, 81], sometimes twice, three or four times [42]. Most of these attacks follow the principle of algebraic complexity reduction (see Section 3.5.4) where from some data for the full 32 rounds GOST we obtain a certain number of pairs for 8 or less rounds of GOST. The quantity of data available after reduction is very small,for example 2,3 or 4 pairs for a reduced round cipher.

In Courtois' initial attack on breaking full 32 rounds GOST, the author described a guess-then-solve attack solving 8 rounds GOST using 4 known P/C pairs as the last step of breaking full 32 rounds of GOST [60]. The time complexity of breaking 8 rounds GOST was 2^{120}. In this thesis we look precisely at questions pertaining to cryptanalysing 8 rounds GOST with 2,3,4 P/C pairs which can be used as a plugin to replace already known attacks on full GOST [60, 42, 62].

Many attacks which do not use any reflections have also been proposed [42, 60, 81] and also differential attacks which do not fall into the algebraic complexity reduction category. The most recent advanced differential attack on GOST has a time complexity of 2^{178} [69, 59] which is also the best single-key attack known.

3.6.3 The Internal Structure of GOST

GOST is a block cipher with a simple Feistel structure, 64-bit block size, 256-bit keys and 32 rounds. Each round contains a key addition modulo 2^{32}, a set of 8 bijective S-boxes on 4 bits, and a simple rotation by 11 positions.

GOST has 32 identical rounds such as the one described on Figure 3.4 below. They differ only by the subsets of 32 key bits which they use. GOST has a weak key schedule which is the main source of all the attacks on full 32-round GOST [60, 42, 105, 70, 69, 51, 59, 81]. In this thesis we only look at up to 8 rounds of GOST which have independent 32-bit keys and don't repeat, therefore we can ignore the GOST key scheduling totally.

We number the inputs of the S-box Si for $i = 1, 2, \ldots, 8$ by integers from $4i+1$ to $4i+4$ out of 1..32 and its outputs are numbered according to their final positions after the rotation by 11 positions: for example the inputs of S6 are $21, 22, 23, 24$ and the outputs are $32, 1, 2, 3$.

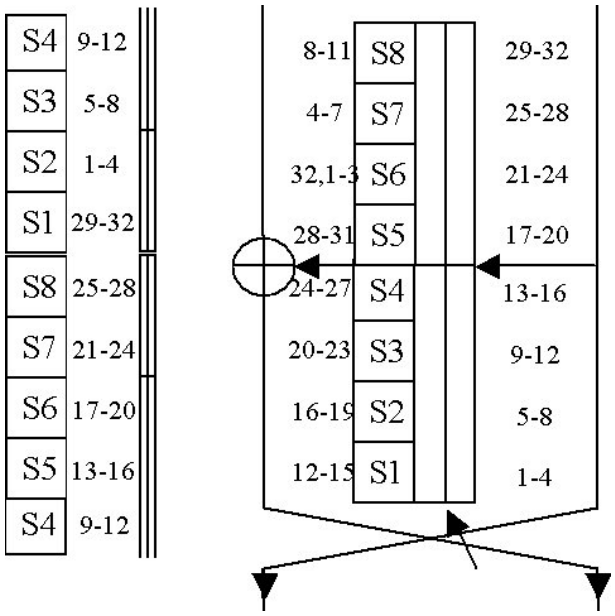


Figure 3.4: One Round of GOST And Connections in The Following Round. This figure describes the encryption process for one round of GOST. Firstly, the 32-bit right half is added with k_i (modulo 2^{32}, showed in figure as $\boxplus$). Then, the result is divided into eight 4-bit consecutive blocks and each block is given as input to a different S-box. The first 4 bits go into the first S-box S1, bits 5-8 go into S2 and so on. Then, the 32-bit output undergoes a 11-bit left circular shift and finally the result is xored to the left 32-bit half of the data.

In Figure 3.4 we also show S-box numbers in the next round in the left margin. This is very helpful in order to see which bits are successfully determined in our attacks on GOST. In a great simplification, in most cases, one S-box in one round affects essentially only two consecutive S-boxes in the next round. Additional propagation is obtained due to the Feistel structure and due to carries in the modular addition.

3.7 Cryptanalysis of SIMON Block Cipher

Nowadays lightweight cryptography is rapidly evolving and becoming more and more important due to the increasing demand from mobile phones and the Internet of Things. These lightweight cryptographic primitives are designed to be efficient (in both hardware and software) when limited hardware resources are available and at the same time to guarantee a desired level of security. The design of such primitives is a great challenge and can be seen as a non-trivial optimization problem,

where several trade-offs are taken into account. They need to maintain a reasonable balance between security and efficient software and hardware implementation with very low overall cost with respect to several meaningful metrics (power consumption, energy consumption, size of the circuit [49, 23, 25]).

The research community has proposed many lightweight hash functions, block ciphers and stream ciphers which are reasonably good and satisfy at a reasonable level the trade-off between efficiency and security. Nowadays, in cryptographic literature we find lots of such lightweight cryptographic primitives such as KATAN [30], KLEIN [95], ICEBERG [153], HIGHT [79], LED [98], mCrypton [118], PRESENT [20], Piccolo [147] and many others.

In July 2013, a team from the NSA proposed two new families of particularly lightweight block ciphers, Simon and Speck, both coming in a variety of blocks and key sizes [14]. We have developed a basic reference implementation of both ciphers which can be found in Github [53], as well as a generator of algebraic equations to be used in algebraic attacks.

The designers of Simon and Speck published the full specifications and presented only performance and implementation footprints, without providing any advanced security analysis against known cryptanalytic attacks. Both of them offer excellent performance on both hardware and software platforms and perform exceptionally well across the majority of lightweight applications and not only on a single platform. Compared to other lightweight cryptographic primitives, these two are meant to have better performance with respect to the area needed for a given throughput, code size and memory usage. Simon is designed for optimal performance in hardware, and Speck for optimal performance in software. According to [7], Simon with an equivalent security level as AES, is 86% smaller than AES, 70% smaller than PRESENT and its smallest hardware architecture only costs 36 slices (72 look-up tables, 30 registers). Recent results about hardware implementation of block ciphers emphasize reducing the size and/or Multiplicative Complexity (MC) further possibly leads to optimal implementations [24, 49].

However, in the original NSA paper [14], there is no analysis of the security

of these 2 ciphers against major well-known attacks. In the same paper [14], the authors briefly said that Simon and Speck were designed to provide security against traditional adversaries who can adaptively encrypt and decrypt large amounts of data, and some attention was given so that there are no related-key attacks. Except for these comments, no more analysis against common attacks such as linear or differential cryptanalysis was presented and the task of analyzing the resistance of the ciphers against known attacks was left to the academic community. Immediately after the release of the specifications we had the first attempts using differential, linear and rotational cryptanalysis [85, 4]. Our attacks on Simon described in Chapter 6were the first algebraic cryptanalysis attacks attempted on Simon. The work was published in 2014 [54]. In recent years Simon was studied heavily by a lot of researchers, including differential attacks introduced by Biryukov et al [19], Mourouzis et al [3] and Wang et al [160, 161], also combined differential and linear attacks by Farzaneh et al [85] and Alkhzaimi et al [4]. Most of them are using statistical cryptanalysis techniques, the best results break around 70% rounds of different versions of Simon. In 2015, Raddum [134] published another algebraic cryptanalysis work on Simon for most of the versions (not including Simon 64/128 version). Raddum's work uses more P/C pairs than our attack and breaks 16 (out of 72) rounds of Simon 128/256 version with ElimLin. This attack shows that there is a need to understand better how ElimLin attacks can scale to larger attacks.

3.7.1 SIMON Structure

SIMON is a family of lightweight block ciphers with the aim of havingan optimal hardware performance [14]. It follows the classical Feistel design paradigm[5], operating on two n-bit halves in each round and thus the general block size is $2n$. The Simon block cipher with an n-bit word is denoted by Simon-$2n$, where $n = 16, 24, 32, 48$ or 64, and if it uses an m-word key (equivalently mn-bit key) we denote it as Simon-$2n/mn$. In this chapter, we study the variant of Simon with $n = 32$ and $m = 4$ (i.e. 128-bit key).

[5]Note that in classical Feistel structure computation is done on left side, see 2.2.3. In Simon computation is done on right side see Figure 3.5

Each round of Simon applies a non-linear, non-bijective (and as a result non-invertible) function

$$F : GF(2)^n \rightarrow GF(2)^n \tag{3.3}$$

to the left half of the state which is repeated for 44 rounds. The operations used are as follows:

1. bitwise XOR, $\oplus$
2. bitwise AND,
3. left circular shift, S^j by j bits.

We denote the input to the i-th round by $L^{i-1}||R^{i-1}$ and in each round the left word L^{i-1} is used as input to the round function F defined by,

$$F(L^{i-1}) = (L^{i-1} <<< 1) \wedge (L^{i-1} <<< 8) \oplus (L^{i-1} <<< 2) \tag{3.4}$$

Then, the next state $L^i||R^i$ is computed as follows (cf. Fig. 3.5),

$$L^i = R^{i-1} \oplus F(L^{i-1}) \oplus K^{i-1} \tag{3.5}$$

$$R^i = L^{i-1} \tag{3.6}$$

The output of the last round is the ciphertext.

3.7.2 Key Schedule

The key schedule of Simon is based on an Linear-Feedback Shift Register(LSFR)-like procedure, where the nm-bits of the key are used to generate the keys $K_0, K_1, ..., K_{r-1}$ to be used in each round. There are three different key schedule procedures depending on the number of words that the secret key consists of $(m = 2, 3, 4)$.

At the beginning, the first m words $K^0, K^1, ..., K^{m-1}$ are initialized with the secret key, while the remaining are generated by the LSFR-like construction. For the

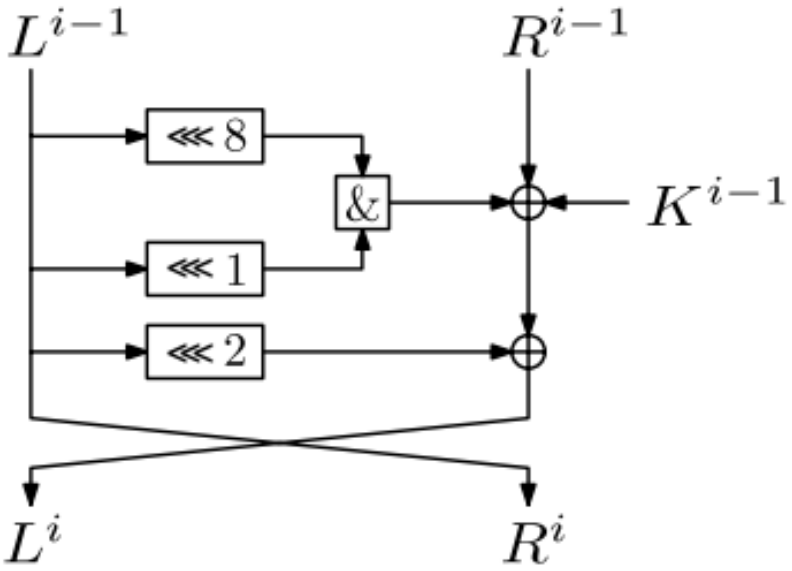


Figure 3.5: The round function of Simon

variant of interest, where $m = 4$, the remaining keys are generated in the following way:

$$Y = K^{i+1} \oplus (K^{i+3} >>> 3) \tag{3.7}$$

$$K^{i+4} = K^i \oplus Y \oplus (Y >>> 1) \oplus c \oplus (z_j)_i \tag{3.8}$$

The constant $c = 0xff...fc$ is used for preventing slide attacks and attacks exploiting rotational symmetries [14]. In addition, the generated subkeys are XORed with a bit $(z_j)_i$, that denotes the i-th bit from the one of the five constant sequences $z_0, ..., z_4$. These sequences are defined in the NSA's orginial paper [14] and for our variant we use z_3. We have implemented a basic reference implementation of Simon and Speck ciphers and a basic generator of equations that are used in algebraic attacks [53] .

The Feistel network, the construction of the round function and the key generation of Simon, enables bit-serial hardware architectures which can significantly reduce the cost of implementation [7]. Additionally, encryption and decryption can be done using the same hardware.

3.8 Summary

Algebraic Cryptanalysis is powerful as it only requires a small amount of data, however it can only break a small number of rounds compare to other cryptanalysis methods due to large computation complexity at solving stage. At the moment most of the exising Algebraic Cryptanalysis applications are just coverting target ciphers to equation systems and then solved directly by a software solver (i.e ElimLin or SAT solvers). Very limited research has been done in order to understand the behavior of a solver and how to make the coverted equation system become easier to solve.

On cryptanalysis of block ciphers we introduced two well known encryption standards: Russian GOST and NSA SIMON. We reviewed the state of art cryptanalysis works on these encryption standards: best algerbraic cryptanalysis attack on 8 rounds GOST require time complexity of 2^{120} and no algerbraic cryptanalysis has been done on SIMON.The above facts motive our research work decribed in Part II of the thesis.

In Chapter 5 of Part II we will introduce a fundemental notion of "contradiction immunity" and describe how conradiction can be used in algebraic cryptanalysis. We will demostrate a mixed SAT/UNSAT attack on GOST which improve the time complexity of current best attack on breaking 8 rounds of GOST from 2^{120} to 2^{92}. In Chapter 6 we will describe a new Algebraic Cryptanalysis approach combined with well selected P/C paris in a Chosen Plaintext Attack scenario and benchmark our attacks with randomly selected P/C pairs. We will apply this method in newly proposed NSA cipher SIMON and provide the first Algebraic Cryptanalysis on SIMON block cipher. Finally in Chapter 7 we will study the behavior of ElimLin method with regards to the number of data samples availiable to an attacker. By studying the number of linear independent equations found by ElimLin, we will show ElimLin has a phase transition process where the number of equations found by ElimLin increase much faster than linear and eventually break the cipher.

Chapter 4

Introduction to Elliptic Curves

4.1 Mathematical Foundations

Finite Groups

A Finite Group is a set G with a finite number of q elements, which has a binary operation $* : G * G \rightarrow G$ and satisfies the following properties [99]:

1. Associativity: $(a * b) * c = a * (b * c)$ for all elements $a, b, c \in G$

2. Existence of an identity: there exists an element $e \in G$ such that $a * e = e * a = a$ for all $a \in G$. Element e is called the neutral element (or identity element) of the group.

3. Existence of inverses: for each element $a \in G$, there exists an element $b \in G$ such that $a * b = b * a = e$. Element b is called the inverse of a.

q is called the group order. In addition, a group is called Abelian group (or commutative group) if it satisfies the commutativity law which is $a * b = b * a$ for all elements $a, b \in G$.

If the binary operation is called addition (+), then the group is additive. In this case, the neutral element is usually denoted by 0 and the additive inverse of an element a is denoted by $-a$. If the binary operation is called multiplication $(\cdot)$, then the finite group is multiplicative. In this case, the identity element is usually denoted by 1 and the multiplicative inverse of an element a is denoted by a^{-1}.

Finite Fields

Abstractly, a finite field consists of a finite set of objects called field elements together with the description of two operations, addition and multiplication, that can be performed on pairs of field elements. These operations must have certain properties: [99]

1. $(\mathbb{F},+)$ is an Abelian group with (additive) neutral element denoted by 0.
2. $(\mathbb{F},\cdot)$ is an Abelian group with (multiplicative) neutral element denoted by 1.
3. The distributive law holds: $(a+b)\cdot c = a\cdot c + b\cdot c$ for all $a,b,c \in \mathbb{F}$.

Prime Fields

Let p be a prime number. The residues modulo p, consisting of the integers $\{0,1,2,\ldots,p-1\}$ with addition and multiplication performed modulo p, is a finite field of order p.

Binary Fields

Finite fields of order 2^m are called binary fields or characteristic-two finite fields. One way to construct $\mathbb{F}_{2^m}$ is to use a polynomial basis representation. The elements of $\mathbb{F}_{2^m}$ are the binary polynomials (polynomials whose coefficients are in the field $\mathbb{F}_2 = \{0,1\}$) of degree at most $m-1$:

$$\mathbb{F}_{2^m} = \left\{ a_{m-1}z^{m-1} + \cdots + a_2 z^2 + a_1 z^1 + a_0 : a_i \in \mathbb{F}_2; +, \cdot \right\}.$$

An irreducible binary polynomial $f(z)$ of degree m is chosen. Irreducibility of $f(z)$ means that $f(z)$ cannot be factored as a product of binary polynomials each of degree less than m. Addition of field elements is the usual addition of polynomials, with coefficient reduced modulo 2. Multiplication of field elements is performed as multiplication of polynomials modulo the polynomial $f(z)$.

Cyclic Groups

Let G be a finite group of order q with multiplication $(\cdot)$ as binary operation, g be a group element of G, then the order of g is the smallest positive integer r for which

$g^r = 1$. G is called cyclic if there exist a g such that $\langle g \rangle = \{g^i : 0 \leq i \leq r-1\}$ is the subgroup of G generated by g. If $r = q$ then G is a cyclic group with generator g if $G = \langle g \rangle$. The set $\langle g \rangle$ is also a group itself under the same binary operation and is called the cyclic subgroup of G generated by g.

4.2 Elliptic Curves

An elliptic curve over a field K is defined by a set of points which satisfy the following equation (also known as standard Weierstrass form) and a group operation which will be defined later.

$$y^3 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.1}$$

Here: $a_1, a_2, a_3, a_4, a_6 \in K$, and the discriminant is defined as

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \neq 0$$

where

$$d_2 = a_1^2 + 4a_2$$

$$d_4 = 2a_4 + a_1a_3$$

$$d_6 = a_3^2 + 4a_6$$

$$d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

The condition $\Delta \neq 0$ guarantees that there does not exist more than one tangent line for a given point on the curve.

For an elliptic curve over a field K of characteristic $\neq 2$ or 3, without loss of generality, one can assume that $a_1 = a_2 = a_3 = 0$, we have $d_2 = 0$, $d_4 = 2a_4$, $d_6 = 4a_6$ and $d_8 = -a_4^2$ [99]. Accordingly, the condition $\Delta = -16(4a_4^3 + 27a_6^2) \neq 0$ can be simplified to $4a_4^3 + 27a_6^2 \neq 0$. Consequently in practical application of elliptic curves, the curve equation is written in the following form:

$$y^3 = x^3 + ax + b \tag{4.2}$$

where $a = a_4$ and $b = a_6$, cf. Equation 4.1.

For an elliptic curve over binary fields $\mathbb{F}_2^m$, without loss of generality, one can assume that $a_1 = 1$, $a_3 = a_4 = 0$ [99]. Then we have $d_2 = 1$, $d_4 = 0$, $d_6 = 0$ and $d_8 = a_6$ [1], and $\Delta = -a_6 \neq 0$. The short form of curve equation is normally written as follows:

$$y^2 + xy = x^3 + ax^2 + b \tag{4.3}$$

where $a = a_2$ and $b = a_6$ and $a, b \in \mathbb{F}_2^m$.

4.2.1 Elliptic Curves Over $\mathbb{F}_p$

The finite field $\mathbb{F}_p$ uses the numbers from 0 to $p-1$, and computations are done modulo p. An Elliptic Curve over finite field $\mathbb{F}_p$ where p is a large prime, can be formed by choosing the constants a and b within the field $\mathbb{F}_p$. The elliptic curve includes all points (x, y) which satisfy the elliptic curve equation modulo p (where x and y are numbers in $\mathbb{F}_p$). It is typically defined in the short Weierstrass form:

$$y^2 \bmod p = x^3 + ax + b \bmod p$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \bmod p$ is not 0, which guarantees $x^3 + ax + b$ contains no repeated factors and then the elliptic curve is a group. The elliptic curve contains all points $P = (x, y)$ for $x, y \in F_p$ that satisfy the elliptic curve equation with addition of (for Weierstrass curves) a special point $\mathcal{O}$ called the point at infinity [2].

To give an example, consider an elliptic curve over the field F_{19}, where $a = 1$ and $b = 6$, the curve equation is: $y^2 = x^3 + x + 6$, an example used in [8]. There are 18 points:

$$(0,5),\ (4,6),\ (2,4),\ (3,6),\ (14,3),\ (12,13),$$
$$(18,2),\ (10,3),\ (6,0),\ (10,16),\ (18,17),\ (12,16),$$

[1] In binary fields anything multiplied by 2 equals to 0.

[2] In code implementation, $\mathcal{O}$ is normally be represented as point (0,0), but not always, as (0,0) might satisfy the curve equation.

$$(14,16),\ (3,13),\ (2,15),\ (4,13),\ (0,14),\ \mathcal{O}$$

The point $P = (4,6)$ satisfies this equation since:

$$6^2 \mod 19 = 4^3 + 4 + 6 \mod 19$$
$$17 = 17$$

Remark: In mod p Weierstrass curves $-(x,y) = (x,-y)$

4.2.2 Binary Elliptic Curves

Elements of the field $\mathbb{F}_{2^m}$ are m-bit strings. The rules for arithmetic in binary field can be defined by either polynomial basis or by so-called (more efficient) [optimal] normal basis[2]. An elliptic curve E with the underlying field $\mathbb{F}_{2^m}$ is given through the following equation:

$$y^2 + xy = x^3 + ax^2 + b$$

where $x, y, a, b \in \mathbb{F}_{2^m}$ and $b \neq 0$. The elliptic curve E includes all points (x,y) which satisfy the curve equation over $\mathbb{F}_{2^m}$, together with a point at infinity $\mathcal{O}$.

To given an example, assume the finite field $\mathbb{F}_{2^4}$ has irreducible polynomial $f(x) = x^4 + x + 1$ (or `0x13` in hex). The element $g = (0010)$ is a generator for the field. The powers of g are:

$$g^0 = (0001), g^1 = (0010), g^2 = (0100), g^3 = (1000), g^4 = (0011), g^5 = (0110)$$

$$g^6 = (1100), g^7 = (1011), g^8 = (0101), g^9 = (1010), g^{10} = (0111), g^{11} = (1110)$$

$$g^{12} = (1111), g^{13} = (1101), g^{14} = (1001), g^{15} = (0001)$$

Consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. The points on E are the following and shown in Figure 4.1.

$$(1, g^{13}), (g^3, g^{13}), (g^5, g^{11}), (g^6, g^{14}), (g^9, g^{13}), (g^{10}, g^8), (g^{12}, g^{12}),$$

$$(1, g^6), (g^3, g^8), (g^5, g^3), (g^6, g^8), (g^9, g^{10}), (g^{10}, g), (g^{12}, 0), (0,1), \mathcal{O}$$

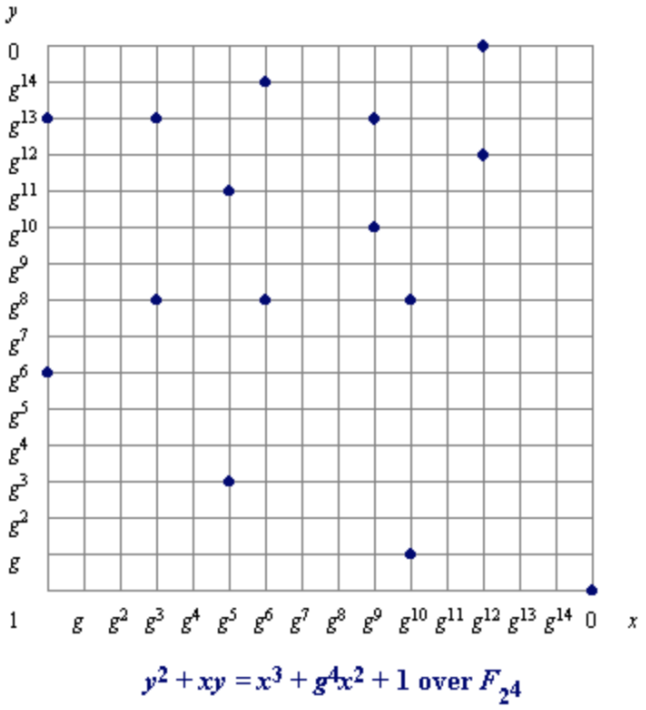


Figure 4.1: Example of elliptic curve over $\mathbb{F}_{2^4}$

4.3 Point Arithmetic

Point Addition for Prime Curves

Let P and Q be two distinct points on an elliptic curve, and P not equals to $-Q$. To add the points P and Q, a line [3] is drawn through the two points. This line will have exactly one additional intersection point with the elliptic curve, which we call $-R$. The point $-R$ is "reflected" in the x-axis to obtain point R. The law for point addition in an elliptic curve group is $P+Q=R$. An example of geometrical graph is given in Figure 4.2.

For point addition of P and $-P$, the line through P and $-P$ is a vertical line which does not intersect the elliptic curve at a third point. Thus the point P and $-P$ cannot be added using the above method. In this case we define $P+(-P)=\mathcal{O}$.

[3] line is a set of points which satisfy the equation $Ax+By+C=0$ where $A,B,C \in \mathbb{F}_p$

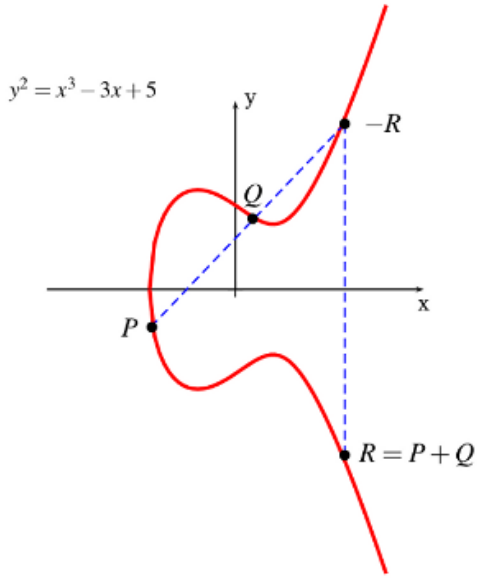


Figure 4.2: Elliptic curve point addition

Point Doubling for Prime Curves

Adding a point $P(x,y)$ to itself, a so-called tangent line to the curve is drawn at point P. If y is not zero, then the tangent line has exact one intersection with the curve at point $-Q$. $-Q$ is reflected in the x-axis to point Q. This operation is called doubling the point P, and the law for doubling is the following (also shown in Figure 4.3):

$$P + P = 2P = Q$$

Doubling the point $P(x,y)$ while $y = 0$ then the tangent line to the curve is vertical and does not intersect the curve on any other point. For such a P, by definition, $2P = \mathcal{O}$, and $3P$ in this case, is $2P + P = \mathcal{O} + P = P$.

Explicit Formulas for Prime Curves

For elliptic curves over $\mathbb{F}_p$, consider two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, $P \neq \pm Q$, the point $P + Q = (x_3, y_3)$ is given by:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

$$x_3 = \lambda^2 - x_1 - x_2$$

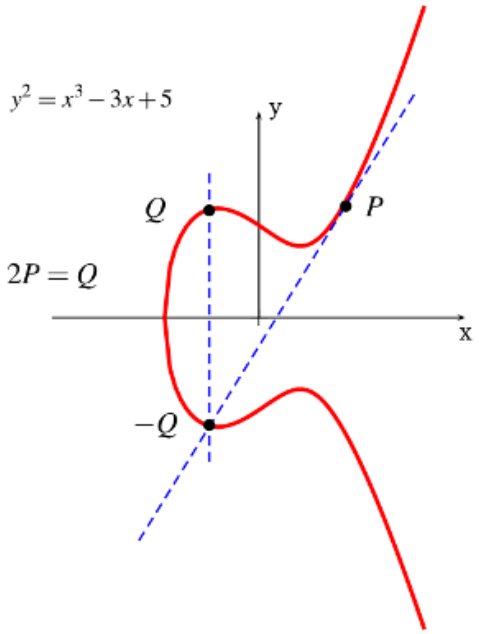


Figure 4.3: Elliptic curve point doubling

$$y_3 = \lambda(x_1 - x_3) - y_1$$

When $P = -Q$, $P + Q$ equals point at infinity $\mathcal{O}$. When $P = Q$, we apply the doubling formula:

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

$$x_3 = \lambda^2 - 2x_1$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

Scalar Multiplication

Given an elliptic curve E defined over a finite field $\mathbb{F}_p$, if $P \in E$ is a point of order r, the cyclic subgroup of E generated by P is $\mathcal{O}, P, 2P, \ldots, (r-1)P$. Then if we define the scalar k as an integer within the range $[1, r-1]$, we can multiply a point by the scalar k and we obtain: $Q = kP$, where Q is also a point which belongs to the subgroup generated by P.

4.4 ECDLP

The hardness of cryptosystem using elliptic curve point multiplication is based on the Elliptic Curve Discrete Logarithm Problem, which is an adaptation of traditional discrete logarithm problem to elliptic curves.

Definition 10. *Elliptic Curve Discrete Logarithm Problem (ECDLP): Given an elliptic curve E defined over a finite field and two points $P, Q \in E$, find an integer k such that $Q = kP$ if such k exists.*

The ECDLP is believed to be harder to solve than other recognized problems such as integer factorization and the discrete logarithm problem in the multiplicative group of a finite field, which are the foundations of RSA [135] and the ElGamal [83] cryptosystems. "Harder to solve" implies shorter keys are needed to provide the same level of security as recommended by [13]. Table 4.1 shows the key size comparison for elliptic curves and RSA.

Table 4.1: NIST's recommendation for practical applications revision 4 [127]

Security level in bits	Block cipher	$\mathbb{F}_p$	$\mathbb{F}_2^m$	RSA
80	SKIPJACK	192	163	1024
112	Triple-DES	224	233	2048
128	AES Small	256	283	3072
192	AES Medium	384	409	7680
256	AES Large	521	571	15360

REMARK: In Jan 2016, NSA has updated this table. Key changes are [127]: security level less than 112 (shaded in red) are no longer approved for applying cryptographic protection on Federal government information. Algorithms (shaded in yellow) are not included in the NIST standards for interoperability and efficiency reasons. At the same time NSA announced changes from Suite B cryptography to the Commercial National Security Algorithm Suite which no longer recommends algorithms shaded in orange for national security systems [126].

Common methods for solving ECDLP are Pollards rho algorithm and index-calculus method. We refer reader to [99, 129] for more details.

4.5 An Interesting Research Question - Semaev Cipher

When we cryptanalyse a block cipher, we write algebraic equations. Is it possible to also describe ECDLP and other EC cryptography problems by simple polynomial equations mod p?

4.5.1 Summation Polynomials

Let E be a general elliptic curve over field $\mathbb{F}$ in Weierstrass form given by the Equation 4.1 we define

$$S_2(X_1, X_2) = X_1 - X_2 \in F[X_1, X_2]$$

The third summation polynomial to be the polynomial $S_3(X_1, X_2, X_3) \in F[X_1, X_2, X_3]$ of degree 4 by [116]:

$$\begin{aligned} S_3(X_1, X_2, X_3) = &\left(X_1^2X_2^2 + X_1^2X_3^2 + X_2^2X_3^2\right) - 2\left(X_1^2X_2X_3 + X_1X_2^2X_3 + X_1X_2X_3^2\right) \\ &- d_2(X_1X_2X_3) - d_4(X_1X_2 + X_1X_3 + X_2X_3) - d_6(X_1 + X_2 + X_3) - d_8 \end{aligned} \quad (4.4)$$

then for $m \geq 4$ in any case:

$$S_m(X_1, \ldots, X_m) = \mathrm{Res}_X\left(S_{m-r}(X_1, \ldots, X_{m-r-1}, X), S_{r+2}(X_{m-r}(X_{m-r}, \ldots, X_m, X))\right)$$

where $1 \leq r \leq m-3$.

Summation Polynomials were first introduced by Semaev in 2004 [140]. Semaev's summation polynomials have the property that if $S_m(a_1, \ldots, a_m) = 0$ for some field elements $a_1, \ldots, a_m \in F$ if and only if there are elliptic curve points $(a_1, b_1), \ldots, (a_m, b_m)$ on E such that $(a_1, b_1) + \cdots + (a_m, b_m) = 0$. The idea is to represent point addition in elliptic curves using a multivariate equation system and try to solve the equation system. This topic has been studied by a lot of researchers trying to solve the ECDLP problem [80, 94, 88, 89, 130, 103]. In 2015 a new method was introduced by Semaev [142] and the idea is trying to solve the equation system by introducing new variables that lower the degree of the system of equations. In this section we will look at Semaev's summation polynomials $S_3(X_1, X_2, X_3)$ for

curves over $\mathbb{F}_p$ and $\mathbb{F}_2^m$, then introduce some open research questions.

Elliptic Curves Over $\mathbb{F}_p$

For an elliptic curve over a field K of characteristic > 3 we recall the curve equation 4.2 $y^3 = x^3 + a_4x + a_6$ and $S_3(X_1, X_2, X_3)$ equation can be simplified to: [116]

$$\begin{aligned} S_3(X_1,X_2,X_3) &= (X_1^2X_2^2 + X_1^2X_3^2 + X_2^2X_3^2) - 2(X_1^2X_2X_3 + X_1X_2^2X_3 + X_1X_2X_3^2) \\ &\quad - 2a_4(X_1X_2 + X_1X_3 + X_2X_3) - 4a_6(X_1 + X_2 + X_3) + a_4^2 \\ &= (X_1 - X_2)^2X_3^2 - 2((X_1X_2 + a_4)(X_1 + X_2) + 2a_6)X_3 + (X_1X_2 - a_4)^2 \\ &\quad - 4a_6(X_1 + X_2) \end{aligned} \tag{4.5}$$

and it is also easy to get for point doubling when $X_1 = X_2$ we have

$$-2\left(X_1^3 + 2a_4X_1 + 2a_6\right)X_3 + \left(X_1^2 - a_4\right)^2 - 8a_6X_1 = 0 \tag{4.6}$$

Special Curve secp256k1

For curve secp256k1 [4] where $a_4 = 0$ and $a_6 = 7$, equation 4.5 and equation 4.6 can be written as the following:

$$\begin{aligned} S_3(X_1,X_2,X_3) &= (X_1 - X_2)^2X_3^2 - 2((X_1X_2)(X_1 + X_2) + 14)X_3 + X_1^2X_2^2 \\ &\quad - 28(X_1 + X_2) \end{aligned} \tag{4.7}$$

and for point doubling when $X_1 = X_2$

$$-2\left(X_1^3 + 14\right)X_3 + X_1^4 - 56X_1 = 0$$

$$X_3 = \frac{X_1^4 - 56X_1}{2X_1^3 + 28}$$

Elliptic Curves Over $\mathbb{F}_{2^m}$

For an elliptic curve over a $\mathbb{F}_2^m$, we have curve equation $y^2 + xy = x^3 + a_2x^2 + a_6$ We

[4] Elliptic curve used in bitcoin; we will give more details in Section 4.8

have $d_2 = 1$, $d_4 = 0$, $d_6 = 0$ and $d_8 = a_6$ (see Section 4.2). In binary field anything multiply by 2 equal to 0, $A - B = A + B$, thus equation 4.4 can be simplified to:

$$S_3(X_1, X_2, X_3) = (X_1^2X_2^2 + X_1^2X_3^2 + X_2^2X_3^2) - X_1X_2X_3 - a_6$$

$$= (X_1X_2 + X_1X_3 + X_2X_3)^2 + X_1X_2X_3 + a_6$$

and when $X_1 = X_2$ we have:

$$X_1^4 + X_1^2X_3 + a_6 = 0$$

$$X_3 = X_1^2 + \frac{a_6}{X_1^2}$$

Research question: What is the hardness of solving summation polynomials? Is it possible to solve (or improve solving) summation polynomials with similar algebraic techniques we described in previous chapters?

4.5.2 Solving Semaev Equations with Extra Variables

In Section 4.5.1 we explained the Semaev polynomials which have equations in very large degree. An interesting idea which is also used in algebraic cryptanalysis is to reduce the degree by adding new variables. In Semaev's recent paper [142] he introduced a new algorithm trying to solving ECDLP using summation polynomials for curves over F_{2^m}. By recursively computing summation polynomials, instead of trying to write $R = P_1 + \cdots + P_m$ for point P_i, we can write $Q_1 = P_1 + P_2, Q_2 = Q_1 + P_3, \ldots, R = Q_{m-2} + P_m$, where the Q_i are completely arbitrary points (see equation

4.8). Then solve the multivariate equation system under some assumptions.

$$\begin{cases} S_3(Q_1, P_1, P_2) = 0 \\ S_3(Q_1, Q_2, P_3) = 0 \\ S_3(Q_2, Q_3, P_4) = 0 \\ \vdots \\ S_3(Q_i, Q_{i+1}, P_{i+2}) = 0 \\ \vdots \\ S_3(Q_{m-2}, P_m, R) = 0. \end{cases} \tag{4.8}$$

From this one can hope to do point splitting and index-calculus. We refer reader to [63, 129] to see how ECDLP might be solved if one can solve such equation efficiently.

However the final result of Semaev's new work is still uncertain. Some researchers believes the complexity analysis in Semaev's paper are not quite correct, cf. later work by Kosters and Yeo [116] and blog posts [93, 43]. This leads to some open research questions.

Semaev Cipher

The new equation system introduced in Semaev's new paper has clear **block cipher topology** (see Figure 3.2 in Section 3.5) [74]. It is very similar to the block cipher equations we have tried to solve for GOST and SIMON [68, 54]. It is potentially able to be solved by methods used in algebraic cryptanalysis [74]. We call Semaev's new summation polynomial equations **Semaev cipher**.

It is possible to see that solving such equations can be done by similar algebraic techniques to those we studied in this thesis, and hardness of all these problems is closely related.

4.6 Elliptic Curve in Cryptography

Elliptic curve cryptography (ECC) was independently proposed by Neal Koblitz[115] and Victor Miller [122] in 1985. It is a public-key cryptography

protocol where each of the participant has a pair of keys. One private key which is kept as a secret by the owner and one public key which is public potentially for everyone. In the past 10+ years ECC has been increasingly used in practice since its inclusion in standards by organisations such as ISO, IEEE, NIST,etc. Elliptic curves are more efficient [17] and offer smaller key sizes [117] at the same security as other widely adopted public key cryptography schemes such as RSA [135].

There are many widely used elliptic curve cryptographic schemes such as Elliptic Curve DiffieHellman (ECDH) key agreement scheme based on the DiffieHellman scheme, Elliptic Curve Integrated Encryption Scheme (ECIES), and Elliptic Curve Digital Signature Algorithm (ECDSA) etc. In this thesis we only focus on ECDSA [106] (key generation part in particular) which is used in Bitcoin and we refer the readers to [99] for details of other schemes.

In Section 4.6.1 we will describe elliptic curve domain parameters, and in Section 4.6.2 we will discuss elliptic curve key pair generation process, and briefly introduce ECDSA in Section 4.6.3.

4.6.1 Domain Parameters

Elliptic curve cryptographic schemes need to agree on a fixed elliptic curve and a finite field. The fixed elliptic curves are normally chosen from curves which are suggested by standard organisations, such as ISO, IEEE etc. Domain parameters for an elliptic curve scheme describe an elliptic curve E defined over a finite field $\mathbb{F}_p$, a base point $G \in E(\mathbb{F}_p)$, and its order n. The parameters should be chosen so that the ECDLP is resistant to all known attacks. Domain parameters are defined as the following $D = (p, \text{FR}, S, a, b, G, n, h)$ where

1. p is the field order
2. FR (field representation) is an indication of the representation used for the elements of $\mathbb{F}_p$
3. If the curve is deterministically generated, S is the seed used to generated the curve
4. $a, b \in \mathbb{F}_p$ that define the curve equation over field $\mathbb{F}_p$
5. G is the base point where $G = (G_x, G_y) \in E(\mathbb{F}_p)$

6. The order n of G

7. The cofactor $h = \frac{\#E(\mathbb{F}_p)}{n}$

An example can be found in Section 4.8.

4.6.2 Key Pair Generation

An elliptic curve key pair is defined for a particular set of valid domain parameters (cf. [99] page 180 for generating and verify EC domain parameters). The public key is a random generated point Q in the group $\langle G \rangle$ generated by G. The corresponding private key is $d = \log_G Q$. The key pair generation algorithm is the following:

Algorithm 2 Key pair generation [99] page 180

Input: Domain parameters $D = (p, FR, S, a, b, G, n, h)$
Output: Public key Q, private key d

1: Select $d \in_R [1, n-1]$.
2: Compute $Q = dG$.
3: Return (Q, d).

Note that the process of computing a private key d given public key Q is exactly the elliptic curve discrete logarithm problem. Hence it is very import to chose a set of domain parameters so that the ECDLP is hard to solve. In addition the number d should be **random** in the sense that the probability of any particular value being selected must be sufficiently small to prevent an adversary from gaining advantage through optimizing a search strategy based on such a probability.

Not all the key pairs are valid keys. A public key $Q = (Q_x, Q_y)$ is valid if it satisfies all the following requirements:

1. $Q \neq \mathcal{O}$ ($\mathcal{O}$ is the point at infinity).

2. Q_x and Q_y are properly represented elements of $\mathbb{F}_p$ (i.e., integers in $[0, p-1]$ for prime field and bit strings of length m for binary field $\mathbb{F}_2^m$).

3. Q satisfies the elliptic curve equation defined by a and b.

4. we can also verify that $nQ = \mathcal{O}$.

4.6.3 Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is a cryptographic scheme based on elliptic curve cryptography that authenticates a message (and a signer), and checks that the content of the message is authentic and comes from the signer. ECDSA is the most widely standardised elliptic curve based signature scheme, appearing in the FIPS 186-2[91], IEEE 1363-2000 [97], ANSI X9.62 [6] etc. Typically, ECDSA consists of three parts: key generation, signing and verification. We have discussed the key generation part in the previous section, see Algorithm 2. Now we look at signature and verification algorithms.

Let m be the message that the sender want to send, the message sender obtained his EC key pair d and Q using the key generation algorithm using an elliptic curve defined by a set of domain parameters $D = (p, FR, S, a, b, G, n, h)$. The process for signature generation is described in Algorithm 3. In the following algorithms, H denotes a cryptographic hash function whose outputs size is at least n (if longer than n, H can be truncated).

Algorithm 3 ECDSA signature generation [99] page 184

Input: Domain parameters $D = (p, FR, S, a, b, G, n, h)$, private key d, message m
Output: Signature (r, s)

1: Select $k \in_R [1, n-1]$.
2: Compute $kG = (x_1, y_1)$ and convert x_1 to an integer $\bar{x}_1$.
3: Compute $r = \bar{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6: Return (r, s).

Anyone can verify the sender signature by using the sender's public key and the verification process is described as follows:

Algorithm 4 ECDSA signature verification [99] page 184

Input: Domain parameters $D = (p, FR, S, a, b, G, n, h)$, public key Q, message m, signature (r, s)

Output: Acceptance or rejection of the signature

1: Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails, reject the signature
2: Compute $e = H(m)$.
3: Compute $w = s^{-1} \bmod n$.
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5: Compute $X = u_1 G + u_2 Q$.
6: If X is point at infinity $\mathcal{O}$ then reject the signature
7: Convert the x-coordinate x_1 of X to an integer $\bar{x}_1$; compute $v = \bar{x}_1 \bmod n$.
8: If $v = r$ then accept the signature otherwise reject the signature.

4.7 Bitcoin and Brain Wallet Attacks

Bitcoin is a cryptocurrency, an electronic payment system based on cryptography. It was created by Satoshi Nakomoto[5] in 2008 [125]. In 2009, Bitcoin was launched as open-source software. Bitcoin is designed to be a fully decentralised peer-to-peer network — self-governing without support from trusted entities such as banks or governments. Bitcoin transactions are like cheques but signed cryptographically instead of using ink. Transactions are broadcast to the peer-to-peer network and verified by each node. A public ledger called a "blockchain" records transactions pseudonymously.

Creation of new bitcoins[6] is through a process called mining and participants are called miners. Miners offer their computation power to solve a hard mathematics problem, and winners will be rewarded the newly created bitcoin. The chance of winning a reward is directly proportional to the miner's computing power. During the process of mining, transactions have been processed and recorded into the blockchain.

Ownership of bitcoins implies that a user can spend bitcoins associated with a specific address (equivalent to a bank account). In order to spend the coins, a payer

[5]It is not known whether Satoshi Nakomoto is a real or pseudonym name or if it represents one person or a group

[6]Following convention, lowercase "bitcoin" refers to a unit of currency within the uppercase Bitcoin system.

must digitally sign a transaction using their private key. The signed transaction is then broadcast to the peer-to-peer network. Everyone on the network can verify the signature that has been sent out. Anyone can spend all the bitcoin in a bitcoin address as long as they hold the corresponding private key. Once the private key is lost, the Bitcoin network will not recognize any other evidence of ownership.

Bitcoin uses digital signature to protect the ownership. Thus it is very important to look at the technical details of the digital signature scheme used in Bitcoin. The popularity of Bitcoin, especially with large populations who had not previously used cryptographic software, has meant that lots of users have attempted to manage private keys for the first time in the context of Bitcoin. In the next section, we study the Bitcoin brain wallet, our main attack target in this part of my research work. We will discuss the technical details of how Bitcoin addresses are generated, using which elliptic curve, and how the curve is defined.

4.8 Bitcoin Elliptic Curve

Bitcoin uses Elliptic Curve Digital Signature Algorithm (ECDSA, see Section 4.6.3). The elliptic curve used in Bitcoin is called secp256k1. In the FIPS 186-2 standard [91] NIST recommends five elliptic curves for use in ECDSA, targeting five different security levels (192,224,256,384,521). In this standard, these curves are named as P-192, P-224, P-256, P-384, and P-521 [7]. The Bitcoin elliptic curve is proposed in Certicom [32] in addition to NIST curve for 256 bits prime. Secp256k1 is defined over prime field $\mathbb{F}_p$ where the domain parameters (p,a,b,G,n,h) are defined as following:

$$p = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F}$$
$$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

The curve equation E is $y^2 = x^3 + ax + b$ where $a = 0$ and $b = 7$. The base point $G : (G_x, G_y)$ is defined as:

$$G_x = \text{79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}$$

[7] In practice these also appear as nistp192, nistp224 etc; in Certicom recommended curves they are named as secp***r1, and in OpenSSL they are called prime***v1

G_y = 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

the cofactor h = 1, the order n of G are:

n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

4.9 Brain Wallets

A Bitcoin wallet is a collection of Bitcoin addresses and stores the corresponding keys for those addresses. Bitcoin wallets come in different forms, including desktop software, mobile apps, online services, hardware, smart card and paper.

As we discussed earlier in Section 4.6.2, the private key is a number which we presume to be totally random. Normally the private key will be a long hex string which is very hard for a person to remember and store safely. No matter what form of wallet we are using, there always exists a chance that someone might lose his wallet in a cybersecurity breach.

Brain wallets are another solution, which do not need the users to keep anything in a safe and still be able to recover their private key. Instead of storing a private key and protecting it, one can store it in a human mind. A brain wallet creates a private key from a (typically) human chosen password or a passphrase, and using the SHA-256 hash algorithm to turn it into a 256-bit number. As SHA-256 is a deterministic method, users can always use the same password to recreate their private key. Note that since brain wallets use the hash directly as the private key, the security of storing private keys now depends only on how unpredictable the passwords are.

Here we give an example show how Bitcoin brain wallet can be generated by using password “password”:

1. Private key: SHA256(“password”)
   5E884898DA28047151D0E56F8DC6292773603D0D6AABBDD62A11EF721D1542D8
2. Public key (uncompressed) : Elliptic curve secp256k1 key pair generation
   04B568858A407A8721923B89DF9963D30013639AC690CCE5F555529B77B83CBFC7
   6950F90BE717E38A3ECE1F5558F40179F8C9502DECA11183BB3A3AEA797736A6
3. SHA256 (Public key)

1D8ED6551EE910136EB0EA735106E137565E8F5EBF8DF73A6A877C92C049F922

4. Hash160 : RIPEMD160

   3E546D0ACC0DE5AA3D66D7A920900ECBC66C2031

   (used for transaction)

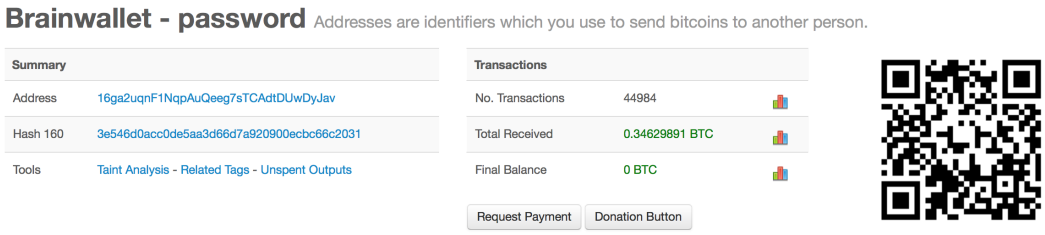


Figure 4.4: Brainwallet generated by password "password"

Brain wallet users are normally urged to use strong passwords or passphrases. Websites provide a brain wallets generation service often using Figure 4.5 to tell the users what is a strong password. However, this figure is a little bit misleading which makes users feel it is safe to use brain wallets with a passphrase. We have actually cracked quite a few such passphrases. In this thesis we mainly focus on speed optimization of password guessing.

In the next section we will discuss about existing methods of Bitcoin elliptic curve implementation, benchmarking the state-of-art attack and show an improved method of running the attack with much faster speed on a laptop.

4.9.1 Related Work

We are not the first ones try to crack Bitcoin brain wallets; a lot of other security researchers are doing it. Many victims have found their money stolen and posted it in forums. The first ethical/research brain wallet cracker was announced publicly in a recent hacking conference DEF CON 23 (Aug 2015). Ryan Castellucci, a whitehat hacker, presented his research on cracking brain wallets, and also published his software [31]. Ryan's attack was done on an Intel i7 PC with 4 hyper-threaded cores. The attack speed can reach approximately 16,250 passwords per second on each thread and he had cracked more than 18,000 brain wallet addresses.

The software Ryan has published uses an existing open source secp256k1 Bitcoin elliptic curve implementation mainly written by Pieter Wuille, one of Bitcoin

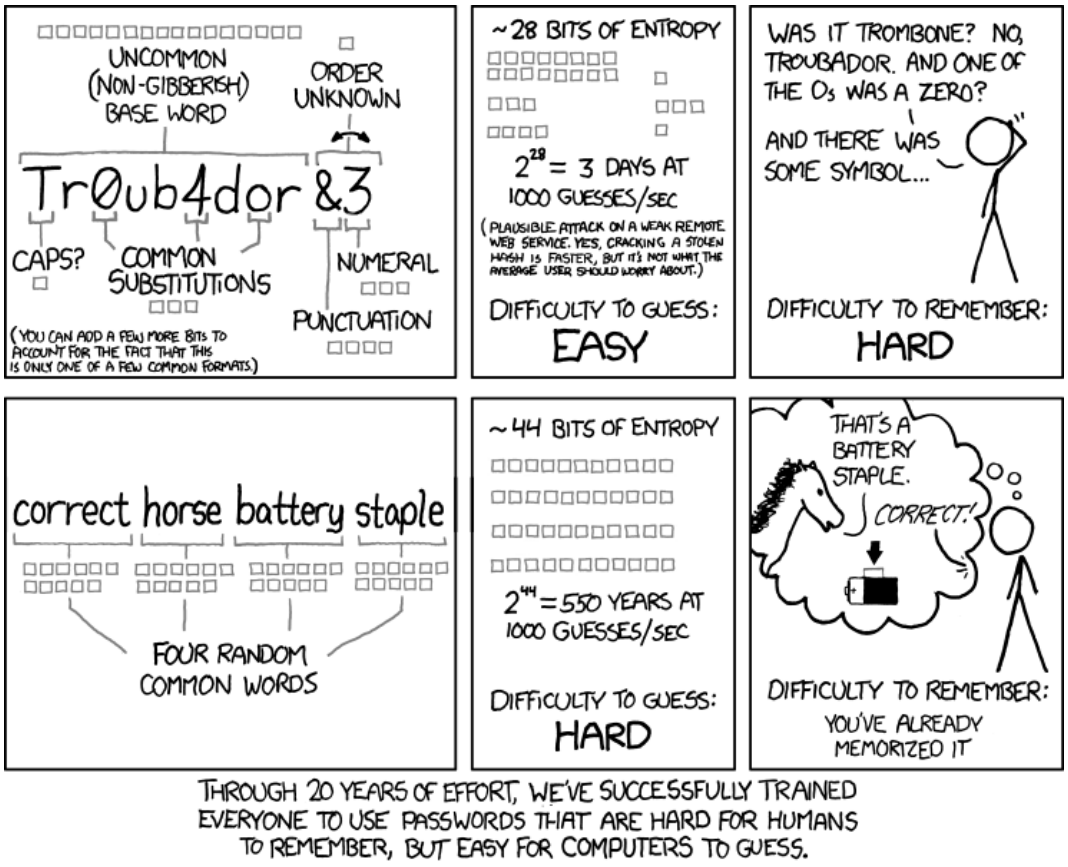


Figure 4.5: Password strength comparison between using password and passphrase, source: `xkcd.com`. This cartoon gives us good advice about how hard it is to guess one user's passwords. But sometimes the speed of guessing can be misleading.

core developers. This implementation is widely used in Bitcoin clients and is considered the current best in terms of code level optimization (detailed benchmarks are given in table 8.2).

Later Vasek et al. published their cybercrime analysis results on brain wallets addresses cracked using Ryan's software implementation in FC 2016. Their work was more focused on brain wallets usage measurements and did not try to improve the speed of the attack.

Part II

The Path to Better Software

Algebraic Cryptanalysis

Chapter 5

Contradiction Immunity and Application to GOST

In this chapter, we will introduce a key fundamental notion of Contradiction Immunity of a block cipher and a related notion of SAT Immunity. These notions lead to new computational optimization problems in cryptography. The main idea is to look for an optimal software guess-then-determine attack. We also provide a concrete example of how this attack can be accomplished on the Russian federation encryption standard GOST with a SAT solver.

5.1 Contradiction Immunity and SAT Immunity

5.1.1 Software Algebraic Attack with SAT Solver

As we discussed in Section 3.6.2, due to the self similarity property inside the cipher structure, the problem of solving full GOST can be reduced to solving an 8 rounds GOST [42]. In Section 3.5.4 we described algebraic complexity reduction and the notion of amplification in general. In this section we look at how the amplification principle can be applied to GOST. Here the question is what is the best possible software attack with tools such as the ElimLin algorithm [67, 47, 56] SAT solvers [49, 9], Gröbner bases [87] and others [141]. In all these algorithms we observe the phenomenon of amplification in various forms. For example, we can study and count linearly independent linear equations and try to amplify their number by the ElimLin algorithm, see [67, 47, 56]. When the ElimLin algorithm is itself the last

step of the attack, or if the SAT solver is the last step of the attack, this amplification phenomenon becomes very important. We observe an avalanche-like phenomenon where more and more new linear equations are generated in the ElimLin algorithm, until the system is solved. Similarly, with a SAT solver there is a point of phase transition where the problem becomes really easy to solve. If we want to understand algebraic cryptanalysis, we need precisely to work on this phase transition phenomenon itself. What happens after this threshold, when the problem is just very easy to solve, is less important.

In this chapter we focus more specifically on cryptographic attacks with SAT solvers, and on GOST, which is a nice example of a weak government standard cipher with excessively poor diffusion. There are two main approaches in SAT cryptanalysis or two main ways to break a cipher with a SAT solver:

1. **The SAT Method:** Guess X bits and run a SAT solver which, if the assumption on X bits is correct, takes time T. Abort all the other computations at time T. The total time complexity is about $2^X \cdot T$.

2. **The UNSAT Method:** Guess X bits and run a SAT solver which, if the assumption on X bits is incorrect, finds a contradiction in time T with large probability $1 - P$ say 99 %.

   With a small probability of $P > 0$, we can guess more key bits and either find additional contradictions or find the solution.

   The idea is that if P is small enough, the complexity of these additional steps can be less then the $2^X \cdot T$ spent in the initial UNSAT step.

3. **A Mixed UNSAT/SAT Attack:** In practice maybe P is not as small as we wish, and therefore we may have a mix of SAT and UNSAT methods: where the final complexity will be a sum of two (or more) terms, none of which can be neglected. We will see a very nice example of how a combined attack can be better than any of SAT and UNSAT methods in isolation in Section 5.2.4.

Note that the UNSAT method does not belong to amplification, but it reduces solving complexity by quickly rejecting wrong guesses. This can be seen as cutting

the branches in a binary search tree, and finally makes finding contradictions easier by a SAT solver. The idea behind the UNSAT method also appears in Cryptanalysis history. For example, Turing's Bombe machine which explored the daily setting of the Germany Enigma, used a similar idea by rejecting wrong guesses which lead to a contradiction (to be precise, one letter is encrypted to itself).

5.1.2 Contradiction Immunity and SAT Immunity

In order to quantify the resistance of a cipher against the two attacks described above, it is natural to define the two following numbers:

Definition 11 (Contradiction Immunity or UNSAT Immunity)**.** *We define the Contradiction Immunity of a given cipher and for $M = 1$ plaintext/ciphertext pairs of the cipher as being the smallest possible number of key bits which can be fixed so that given $M = 1$ P/C pair we can obtain a contradiction with probability at least 50 % by just examining the logical consequences of these key bits. We require this contradiction to be found in a very short time, less than 1 second for the best SAT solver available.*

Definition 12. *[SAT Immunity or Satisfaction Immunity] We define the SAT Immunity of a given cipher and for M plaintext/ciphertext pairs of the cipher as being the smallest possible number of key bits which can be fixed so that given M P/C pairs we can compute the secret key by the best available SAT solver in a relatively short time, say less than 1000 seconds.*

These notions are as precise as they can be. Although they depend on the software used, but it is not to excessive extent. Because we can only hope to provide upper bounds for this quantity by concrete attacks with concrete software, it makes sense to use the best available software and improve these bounds slightly each time as the software improves. Importantly, we should consider that the first notion is much more robust and more fundamental: it expected to depend only on the connections between the components with the "optimal" subset of key bits, we do not expect that the contradiction will be found by examining too many other bits, but just by simple step-by-step local analysis. We also expect that the time taken

to find a contradiction will be essentially zero and will not depend too much on the software used. Some SAT solvers are good at solving contradiction problems (known as UNSAT problems), but in this case, we believe it is not very important which SAT solver will be used to obtain the contradiction. In contrast, the SAT Immunity can only be determined by somewhat "solving" the whole cipher, with the avalanche effect. Unless we are able to determine all the bits in the whole cipher, we do NOT know if the cipher is really solvable. Our experience shows that the results for the second notion will depend a lot on the SAT solver software used and where some software works well, some other does not seem to work at all [1]. This is because when solving UNSAT problems (espically in crypanalysis) a contridiction can be easily found in early round of the encryption, thus the solving complexity is very low and problem normally get solved within seconds. Even different algorthims perform differently on a given problem, in most of the case, the difference is very small in order to find a contridiction. However when solving a SAT problem the aim is to solve the whole equeation system with and find the key, (in Definition 12, we set solver time out as 1000 seconds). Due to the complexity of the problem, some SAT solvers can't find any solution within the time limit.

A small technicality is that in order to determine the key uniquely in many ciphers where the key size is bigger than the block size, it is necessary to use some $M > 1$. Because in order to find a unique solution within a multivariable equations system, the number of equations should be larger than the number of unknowns, otherwise there could exist multiple solutions in the equation system. However, for the first notion, for finding contradictions, we can frequently limit ourselves to considering the case where $M = 1$ because most of the case contridiction can be found easily within early round of the encryption if key bits are wrong.

5.1.3 Applications of UNSAT/SAT Immunities

Cryptanalysis. SAT and UNSAT immunities will allow us to evaluate the security of the cipher against cryptanalytic attacks with a SAT solver. Upper bounds we

[1] Performance of a SAT solver solving cryptanalysis problems is largerly depends on how the cipher is represented using CNF format. The number of variables and constrains in a SAT problem can dertimin the solving complexity and each solver perfroms differently on different problem.

obtain do translate, more or less, as we will see, into concrete attacks with a complexity of about 2^X. The two figures will also indicate which of the three strategies (SAT/UNSAT/Mixed) is more likely to work.

Design of Block Ciphers. It is easy to see that the designer of a cipher can very effectively lower-bound these quantities. This will be achieved by making sure that each S-box in each round influences as many S-boxes as possible in the next round. This is not all very different to designing a cipher which is provably resistant to linear and differential cryptanalysis. In 2007, Schneier once claimed that "Against differential and linear cryptanalysis, GOST is probably stronger than DES" [138]. Therefore we should also expect that Contradiction Immunity of DES and GOST are comparable. Happily, similar attacks with SAT solvers have been developed for both DES [47] and GOST [49]. In fact, it is obvious that the diffusion in DES is much better than in GOST and so is the Contradiction Immunity in DES. However, we need to be careful about drawing any conclusions and direct comparisons do not mean much. If the contradiction immunity is 78 for 8 out of 32 rounds of GOST with 3 P/C pairs and 256-bit keys, is it better or less good than contradiction immunity being 20 for 6 rounds out of 16 of DES with 1 P/C pair and 56-bit keys? It is very hard to say.

5.2 Applying SAT/UNSAT Immunity to GOST and DES

5.2.1 Application to DES

Here we give some basic results for DES obtained by the methods of [47] with, however, a better SAT solver CryptoMiniSat 2.92 [152]. In DES the key bits are spread more or less uniformly in different rounds, and they tend to repeat many times. Therefore it is difficult to choose really good sets of bits using our discovery method and, for now, we just report upper bounds obtained when choosing the key bits at random, and letting the SAT solver do the job.

There are multiple ways of write DES s-boxes into algebric equations which will leads to different solving complexity for AC solving stages. In 2007 Courtois

and Bard studied the best s-box encodings with low I/O degrees [64], and provided ready software for generating such equations and solve the SAT problem using MiniSat [151]. Our experiments set up uses the same software for generating low I/O degree DES equations, and the results are obtained by using a better performing SAT solver CryptoMiniSat 2.92 [152].

For Contradiction Immunity we experiment with 8 rounds of DES for 1 P/C pair using random generated key and randomly fix a large number of the key bits x , then reduce x until we no longer able to obtain a contradiction with probability at 50% using SAT solver. Finally, report the smallest possible number of key bits. For SAT immunity, the experiment was using 8 rounds of DES and 1 P/C pair, random key and randomly fix the key bits. We run CryptoMiniSat until we obtian the smallest possible number of key bits that can be fixed so that we can solve the converted SAT problem and obtain the secret key within 1000 seconds time limit.

Our final results are:

1. The Contradiction Immunity is at most 44 for 8 rounds of DES.

2. The SAT Immunity is at most 34 for 8 rounds of DES and 1 P/C pair.

DES has key size of 56 bits, for SAT method, we have to fix 34 bits (request 2^{34} GOST encryption) and solve within 1000 seconds by a SAT solver. Our PC can run approximately 2^{22} 8 rounds DES encryptions per seconds. The total time complexity of solving 8 rounds of DES using 1 P/C pair is larger than brute force. For ultra low-data complexity attacks, 8 rounds of DES seem already secure or secure enough.

5.2.2 Contradiction Immunity of GOST

Now we are going to provide some results on the Contradiction Immunity and SAT Immunity of GOST. These results are constructive upper bounds.

First we started in a similar set up as DES, with randomly selected key bits for guessing. The initail Contradiction Immunity we got was about 128. Previous cryptanalysis work on GOST done by Coutrios in 2012 [59] analysed the internal structure on GOST and how key bits and S-boxes are connected for each round of

the encryption. Courtios' work [59] shows that GOST splits very neatly into two nearly independent ciphers with 128-bit key each, which are only loosely connected. With this idea it is easy to understand why a software/algebraic attack on 8 rounds of GOST with complexity of 2^{120} seems plausible and natural. Randonly guessed key bits will not create a closely connected structure, thus the chance to find a contridiction for wrong guesses become harder compared to closely connected key bits. However, our research found that it is much lower than 128, and much closer to 64.

Proposition 1. *The Contradiction Immunity for 8 rounds of GOST is at most 78.*

Notation, cf. Figure 5.1: We denote by $S^1 3$ just 1 higher ranking bit at S-box 1 in a given round. Similarly we denote by $S_3 3$ the 3 lower ranking bits of S3.

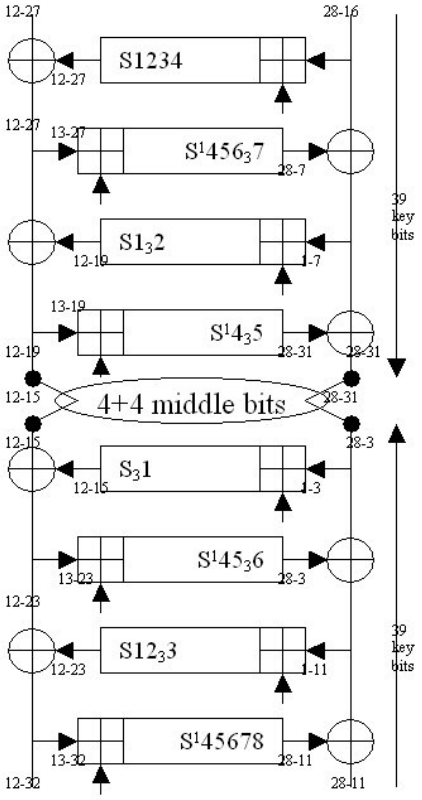


Figure 5.1: Our best set of 78 bits for UNSAT. We denote by $S^1 3$ just 1 higher ranking bit at S-box 1 in a given round. Similarly $S_3 3$ the 3 lower ranking bits of S3. $\boxplus$ is the modulo 2^{32} operation. Best set we found is fixing key bits 0-15,47-58,64-70,111-114,128-130,175-182,192-202,239-255. The inner rounds output bits that are determined by these key bits are showed in the figure

We have constructed and tried many different sets aiming at a contradiction in the middle. Our best set is as follows (cf. Figure 5.1): 0-15,47-58,64-70,111-114,128-130,175-182,192-202,239-255. The contradictions can be found in time of 0.06 s with CryptoMiniSat 2.92 software [152] with a probability of about 50 %. It is easy to see that they could be obtained in essentially constant time by a dedicated algorithm with some small precomputed tables.

5.2.3 SAT Immunity of GOST

It turns out that a set which is good for UNSAT is typically NOT good at SAT. No SAT solver software we dispose of is able to find the missing bits if the 78 bits of Figure 5.1 are fixed. Happily we have found sets which are very good at SAT and they are in fact smaller than 78. Our best result is as follows:

Proposition 2. *The SAT Immunity for 8 rounds of GOST and 4 P/C pair is at most 68.*

We use the following set of bits depicted on Fig 5.2 0-15,51-55,64-66,128-130,179-183,192-207,224-231,244-255. All the remaining 256-68 bits can be determined in time of about 400 seconds using GOST encodings described in [49] and with CryptoMiniSat 2.92.

From here a naive "SAT strategy" attack on GOST would be to run a SAT solver for 400 seconds 2^{68} times. This would be about 2^{99} GOST encryptions.

5.2.4 Low Data Complexity Meet-In-The-Middle Attack for 8 Rounds GOST

In our attack with 4 P/C pairs we want to find a contradiction for all the 4 pairs simultaneously. This will be easier than contradiction with 1 P/C pairs we studied previously. This leads to the following improved attack which mixes the SAT and UNSAT strategies.

1. We use our set of 68 bits as on Figure 5.2.

2. We run the software 2^{68} times for all possible assignments of the 68 bits.

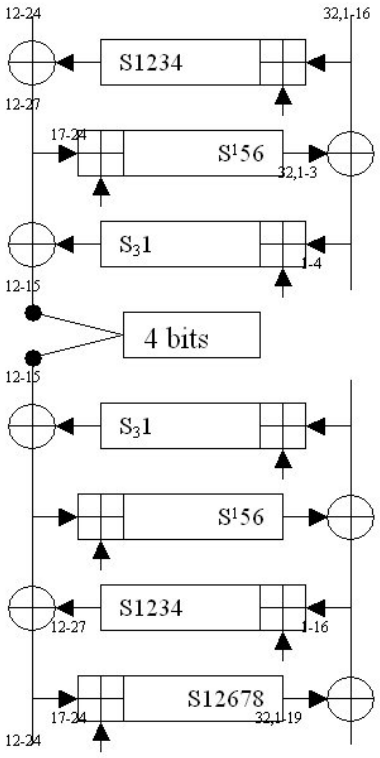


Figure 5.2: Our best set of 68 bits for SAT

3. Computer simulations with the timeout of 7 seconds, a proportion of $1-2^{-5}$ of cases on 68 bits terminates with UNSAT within 2 s on average.

4. Overall, we only need to run a proportion of 2^{-5} of all the 2^{68} cases for as many as 400 seconds; in other cases it simply terminates automatically within 2 s which is 2^{23} GOST encryptions on the same CPU.

5. Assuming that all the other cases run for 400 s (some still terminate earlier) our conservative estimate of the attack time is $2^{68+23}+2^{68+31-5} \approx 2^{94}$ GOST computations.

5.3 Conclution

As we discussed in Section 3.6.2, the above attack scenario can be used as the last step of many other cryptanalysis attacks following amplification principle. For example, following the previous attack described in [62], the 8 round attack can be transformed into an attack on GOST in the multiple key scenario. If there is a

suitable population of at least 2^{64} different keys generated at random (which can be collected from multiple encryption devices), then one can find one valid 256-bit key, in time of about 2^{94} GOST encryptions for one (weaker) key. There are many other ways to perform an attack on full GOST with our attack which is outside of the scope of my thesis. We refer to [60, 62, 42] for additional details.

Chapter 6

Algebraic Cryptanalysis of Simon

In Chapter 5 we have introduced the notion of contradiction immunity and SAT immunity. We have shown how an optimized guess-then-solve attack can be done on GOST. In this chapter, we are going to explore how to improve algebraic attack with well chosen data. Our target is the new NSA block cipher Simon[14]. The lightweight block cipher Simon was introduced by NSA in 2013 and later submitted to ISO to become an international standard. Similar to GOST, Simon has very low diffusion and small S-boxes, which makes it an excellent target for algebraic cryptanalysis. Our aim is to use the very rich algebraic structure with additional data provided (e.g. pairs $\{(P,P'),(C,C')\}$ which follow a certain highly probable truncated differential property) in order to solve the underlying multivariate system of equations. We attempt to solve the system by either using a SAT solver (after converting the system to its corresponding CNF-SAT form with the Courtois-Bard-Jefferson method [9]) or by the ElimLin algorithm [22, 49, 52]. We are able to break up to 10 (/44) rounds of the cipher using a SAT solver and the usual guess-then-determine techniques. Surprisingly, in most cases we are able to obtain the key without guessing any key bits when truncated differentials are used. This is a very remarkable result as it gives a good hope that the attack will scale up well for a larger number of rounds. This is possibly due to the very low non-linearity of the cipher and suggests that it is worth studying a specific strategy for P/C pairs which have a certain structure and decrease even more the non-linearity of the system by introducing more linear equations (e.g. truncated differential properties) until the

key can be obtained even for a larger number of rounds. We will discuss in details our results in Section 6.3.

6.1 How to Write Simon Equations

In Figure 3.1 we showed a general example of modelling a block cipher into MQ equation systems. Here we give a concrete example for writing Simon64/128 in a single round.

Recall the Simon structure in Figure 3.5, let ek_i be the i-th key bit used in Nr-th round of Simon64/128 encryption. The 32-bit key has a binary representation: $ek = (ek_{31}, ek_{30}...ek_0)$. Let ZR_i^{Nr} and ZL_i^{Nr} be the i-th bit of Nr-th round right and left side input, and similarly ZR_i^{Nr+1}, ZL_i^{Nr+1} for the i-th bit of output. Then we have:

$$\begin{gathered}
ek_0 + ZL_{30}^{Nr} + ZR_0^{Nr} + ZL_0^{Nr+1} + ZL_{24}^{Nr} * ZL_{31}^{Nr} = 0 \\
ZR_0^{Nr+1} + ZL_0^{Nr} = 0 \\
ek_1 + ZL_{31}^{Nr} + ZR_1^{Nr} + ZL_1^{Nr+1} + ZL_{25}^{Nr} * ZL_0^{Nr} = 0 \\
ZR_1^{Nr+1} + ZL_1^{Nr} = 0 \\
\cdots \\
ek_{31} + ZL_{29}^{Nr} + ZR_{31}^{Nr} + ZL_{31}^{Nr+1} + ZL_{23}^{Nr} * ZL_{30}^{Nr} = 0 \\
ZR_{31}^{Nr+1} + ZL_{31}^{Nr} = 0
\end{gathered}$$

Similarly, one can simply write equations for the key extension function, cf. [14]. More details can be found in our Simon implementation code (including equation generator for all versions) which is available online [149].

6.2 Differential-Algebraic Cryptanalysis of Simon

We evaluated the security of Simon against algebraic attacks under the following three settings (cf. Fig. 6.1), where S=Similar and R=Random as explained in the introduction.

- **Setting 1:** Known Plaintext Attack. Random P/C pairs are available (RP/RC).
- **Setting 2:** One type of Chosen Plaintext Attack. Random P/C pairs are available with plaintexts of low Hamming distance (SP/RC).

- **Setting 3:** Some form of Chosen Plaintext and Ciphertext Attack. Random P/C pairs which satisfy a truncated differential property in the input and output of the reduced-version of the cipher we study. (SP/SC)

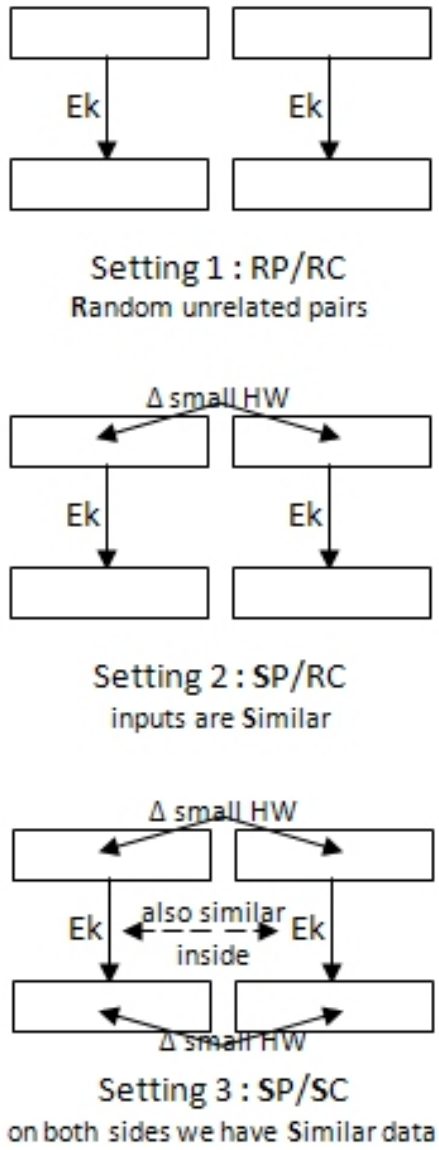


Figure 6.1: Our three attack scenarios

Setting 1 is the simplest setting for understanding how many rounds of Simon can be broken by simple techniques, assuming availability of a few P/C pairs. This setting helps us to understand the maximum number of rounds we can break by guessing as few key bits as possible and using as few P/C pairs as possible. It is a non-trivial step in order to set the benchmark for attacking larger number of rounds.

Setting 2 is a form of known-plaintext attack. Setting 2 requires the existence of P/C pairs with plaintexts of low Hamming distance (or similar plaintexts) such that many variables are eliminated in the first few rounds due to weak diffusion. By eliminating some variables from the initial equations we expect that the system will be solved faster using any solving technique. This is a form of chosen plaintext attack.

Lastly, Setting 3 assumes the existence of P/C pairs

$$\{(P_1, C_1), (P_2, C_2), ..., (P_n, C_n)\}$$

such that $P_i \oplus P_j \in \Delta P$ and $C_i \oplus C_j \in \Delta C$, for all $1 \leq i, j \leq n$ and some truncated differential masks $\Delta P, \Delta C$ of low Hamming weight which holds with comparatively high probability. In our attacks we always use 2 pairs which satisfy a given truncated differential property and then more P/C pairs are generated by using the first 2 plaintexts and computing the encryptions of new plaintexts which have small Hamming distance from the first ones.

The difference from Setting 2 is that in this case we also eliminate variables from the last rounds of the cipher, expecting that the system is even easier to solve. In this attack, we assume (to make it simple) that the entire codebook is available and thus the data complexity is 2^{64}.

In all of our attacks we start with an 8-round truncated differential property (see Section 3.4 and more details can be found in [123]) $\Delta = [0000022200000080]$ with 4 active bits found by heuristic method:

$$Prob(\Delta \rightarrow \Delta) \simeq 2^{-20.51} \tag{6.1}$$

The mask $[0000022200000080]$ denotes the set of $2^4 - 1$ possible differences, excluding the zero difference. Our detailed results and discussion will be presented in the following section.

6.3 Algebraic Attacks experiments and results

We run experiments using SAT solvers and ElimLin Algorithm on a machine with the following characteristics

- CPU: Intel i7-3520m 2.9GHz
- RAM: 4G
- OS: 64-bit Windows 8

Our open source implementation of Simon also includes an equation generator which generates an algebraic equitation system for n round Simon encryption. Once the equation is generated, we use Nicolas Courtois' software [41] to either solve the system by ElimLin or convert to CNF file and then solve by a SAT solver. For the reader to check and verify our results, here are the command lines we used to get our experimental results, software are available online [53, 41] :

- Equation generation: Simon.exe NumberOfRounds fixedKey
- ElimLin: xl0.exe /deg2 fileName
- SAT solver: xl0.exe /deg2 /sat /bard /timeout600 /cryptominisat64296 fileName

6.3.1 Experiments with 2 P/C pairs

The initial benchmark experiments were done with only 2 P/C pairs and solved by a SAT solver using 8 round Truncated differential mask $[0000022200000080] \Rightarrow [0000022200000080]$. Table 6.1 presents our 2 P/C pairs experiment results. The average time (in seconds) taken $T_{average}$ to solve the underlying problem by a SAT solver is presented, while the time complexity C_T (in terms of Simon encryptions) is computed by the following formulae,

$$C_T = 2^k \times 2^{log_2(T_{average} * N_{8REnc})}, \tag{6.2}$$

where k is the number of bits we guess initially, N_{nREnc} is the number of n round Simon encryptions the experiment PC can run in 1 second.

We start from setting 1 using random Plaintext and random Ciphertext pair (RP/RC), until we cannot solve the equations using SAT solver. We successfully break 8 rounds Simon with guessing some key bits. The best result for 8 rounds is fixing 80 key bits with a complexity of $2^{106.53}$. Fixing less than 80 key bits will not be solved by SAT solver within 600 seconds. Then we try setting 2 and

Table 6.1: Best results obtained by a SAT solver using 2P/C pairs for 8 rounds of SIMON64/128. k is the total number of randomly guessed key bits. The time complexity for guessing is 2^k. #(P/C) shows the number of plaintext-ciphertext pairs. $T_{average}$ is the average solving time using a solver. C_T is the total time complexity for breaking the cipher, calculated based on equation 6.2. For settings see detail in figure 6.1

#(Rounds)	k	#(P/C)	$T_{average}$(s)	C_T	Setting
8	84	2	63.76	$2^{110.08}$	RP/RC
8	80	2	87.38	$2^{106.53}$	RP/RC
8	75	2	156.60	$2^{102.37}$	SP/RC
8	75	2	515.60	$2^{104.09}$	SP/SC

setting 3 to compare the results. Setting 2 using selected Plaintext and Random Ciphertext (SP/RC) and setting 3 using selected Plaintext and selected ciphertext (RP/RC) are better than RP/RC. However, the results in Table 6.1 show that SP/SC is not performing any better than SP/RC. Both SP/RC and SP/SC are not improving RP/RC a lot.

The experiment results show that for setting 2 and setting 3, using only 2 P/C pairs cannot provide enough information (additional linear equations) to perform a better attack than RP/RC. In the next subsection we try to increase the number of P/C pairs and compare the results using different settings.

6.3.2 Experiments with more P/C pairs

We start by using more P/C pairs for 8 rounds Simon and we show our results in Table 6.2. Here we start to record some features of the converted CNF files. In Table 6.2 and other SAT solver results table, n, s, h stand for number of variables, average sparsity (the average number of literals in each clause) and hardness respectively and m is the number of clauses in the converted CNF file. We define hardness as a number h such that h^n is the running time, where n is the number of variables. It is known that $h \leq 1.47$ for 4-SAT problems (cf. Table 1 in [143])

Then we start to extended the current 8 rounds attack to 9 and 10 rounds. We first extend our 8R truncated differential mask to 9 and 10 rounds as the following:

- 9R: $[0000002200000080] \rightarrow [00022e4c00000222]$
- 10R: $[0000002200000080] \rightarrow [002eff9a00022e4c]$

Table 6.2: Best results obtained by a SAT solver for 8 rounds with 6 P/C pairs. n, s, h stand for number of variables, average sparsity (the average number of literals in each clause) and hardness respectively and m is the number of clauses in the converted CNF file. We define hardness as a number h such that h^n is the running time, where n is the number of variables.

#(Rounds)	k	#(P/C)	$T_{average}$(s)	C_T	Setting	n	$x = \frac{m}{n}$	s	h
8	45	6	207.31	$2^{27.78}$	RP/RC	8576	6.5118	4.2761	1.0032
8	0	6	12.21	$2^{23.76}$	SP/RC	8576	6.5065	4.2787	1.0029
8	0	6	11.84	$2^{23.57}$	SP/SC	8576	6.5513	4.2631	1.0028

We manage to break 9 rounds without guessing any key bits and break 10 rounds with some key bits guessing. Our best results are shown in Table 6.3

Table 6.3: Best results obtained by a SAT solver. Table set up is the same as table 6.2. The upper part results are using SP/RC P/C pairs, and the time compliexity for breaking 10 rounds SIMON64/128 is $2^{118.5}$ while using SP/SC with truncated differencial properities gives $2^{98.79}$. The results show the power of using well selected data samples in algerbic cryptanlysis.

#(Rounds)	k	#(P/C)	$T_{average}$(s)	C_T	Setting	n	$x = \frac{m}{n}$	s	h
9	0	6	222.50	$2^{27.9}$	SP/RC	9536	6.70	4.31	1.0029
9	0	7	94.7	$2^{26.6}$	SP/RC	11104	6.70	4.31	1.0024
10	90	8	346.0	$2^{118.5}$	SP/RC	13952	6.90	4.32	1.0020
9	0	6	24.24	$2^{24.7}$	SP/SC	9536	6.69	4.31	1.0026
9	0	7	18.56	$2^{24.3}$	SP/SC	11104	6.70	4.31	1.0022
10	70	10	417.73	$2^{98.79}$	SP/SC	17408	6.88	4.31	1.0022

Moreover, assuming Setting 3 we can break 10 rounds by guessing 70 bits of the key initially with a time complexity of $2^{98.79}$ encryptions. Note that in SP/SC setting we always generate two P/C pairs which satisfy the truncated differential property, and the rest pairs are generated at random.

The other two settings have failed to produce good results for 10 rounds in reasonable time and this reflects the power of using pairs which follow strong truncated differential properties. We conjecture that for a cipher of low non-linearity, there exists a certain amount of pairs which depends on the linear relations in the cipher which can be used to break any round.

6.3.3 ElimLin Results

Table 6.4 presents the results using the ElimLin algorithm for solving the underlying system of equations.

Table 6.4: Best results obtained by a ElimLin Algorithm

#(Rounds)	k	#(P/C)	$T_{average}$(s)	C_T	Setting
8	0	6	824.4	$2^{29.8}$	SP/RC
8	0	6	583.2	$2^{29.3}$	SP/SC

ElimLin exploits the existence of linear equations in order to solve the system. We have been able to break up to 8 rounds in Setting 3 without guessing any key bits initially. Setting 1 fails, while Setting 2 is much weaker than Setting 3. The best attack we have obtained is of time complexity $2^{29.3}$ encryptions against 8 rounds of Simon and requires pairs which satisfy the truncated differential property presented in the previous section extended for 8 rounds.

Adding pairs which follow a strong truncated differential property is equivalent to adding linear equations in the system and this is exploited by the ElimLin algorithm. An immediate improvement is to use additional intermediate truncated differences. This will also eliminate variables in intermediate rounds and introduce more linear equations in the intermediate rounds. We conjecture that ElimLin is more powerful in cases where a strong characteristic is found.

6.4 Conclusion

Chapter 7

Re-Designing Algebraic Attacks Beyond ElimLin

7.1 ElimLin Overview

Our initial algebraic cryptanalysis work on Simon was published in 2014. In 2015, Raddum [133] published another algebraic cryptanalysis on several different versions of Simon and have broken 16 out of 72 rounds on the largest existing version Simon128/256 using Elimlin with 20 P/C pairs of chosen plaintexts. Raddum's work also shows that ElimLin performs better when adding more P/C pairs.

However, it's hard to know what happens when we add even more P/C pairs. This is due to the limited computation power we have. A major difficulty with ElimLin is that so far it has been successful only for relatively simple lightweight ciphers. For more complex ciphers it seems to do things which are relatively trivial, e.g. equations generated do not penetrate deeply inside the cipher, or very slowly, cf. slide 153 in Courtois' lecture notes [44].

Our work aims to make some definite progress in the direction of distinguishing between trivial and non-trivial behavior for ElimLin. This is about how deeply an ElimLin attack penetrates. Previous experience shows that ElimLin only starts to work at a certain threshold. Before this threshold, again, nothing non-trivial can be observed even though slow penetration occurs. This is not really apparent in any of the current works or is lost in vast quantities of data generated in the computer

simulations.

In this chapter, we study ElimLin behaviours with randomly selected, increasing number of data samples. By collecting data from large simulations and deeply inspecting the different types of equations generated by ElimLin, we study

1. The growth rate of equations generated by ElimLin

2. Where the higher growth rate comes from

3. If it is possible to predict when ElimLin will break a cipher

We define a new criterion which shows that it is possible to see that there exist two very different and easily distinguishable patterns in ElimLin. Either the attack follows one pattern, and does nothing non-trivial, or it follows another pattern and it is very clearly doing well. Then we look deeply inside the large number of linear equations found by ElimLin and try to classify different type of equations and identify where the higher growth rate come from.

7.1.1 Phase transitions

It is known that many NP-hard problems are subject to "phase transition"; with certain parameters that problem is hard, and then it will rather abruptly transition from "hard" to "easy to solve". This is what we observe with ElimLin. Let K be the number of Plaintext/Ciphertext (P/C) pairs used in an ElimLin attack. We are going to discover that at a certain threshold the number of new linear and linearly independent equations generated at various stages of the attack can follow one curve, and then switch to another curve with a different asymptotic growth rate.

Conjecture 7.1.1 Consider a system of multivariate equations derived from a block cipher (see toy example in 3.5 Figure 3.1). Consider a simple known plaintext attack with K Plaintext/Ciphertext (P/C) pairs. Consider a case such that the cipher is eventually broken by ElimLin, cf. [40, 39, 48, 46, 133]. The number of new and linearly independent linear equations generated by the ElimLin algorithm goes through several distinct stages St0-St3:

St0 Initially it grows linearly with K, and for certain individual stages of the attack is simply equal to 0 and does NOT grow, cf. our later s_i notation in Section 7.2.1.

St1 Then it switches to another curve where it grows faster than linearly in K.

St2 This is until it reaches a **saturation** stage where the cipher is completely broken by ElimLin. Here we sometimes have a very rapid phase transition (cf. Section 7.3) where the number of equations s_i generated at one stage becomes 0 again simply because an earlier stage of the attack reaches a certain threshold where combinatorial explosion in additional equations generated makes it complete the whole attack and does not requite the next stage to be executed.

One (old) example from 2007 which shows that the number of equations grows faster than linear as a function of the data complexity K in ElimLin can be found in Courtios and Debraize's work in 2008 [48].

In this chapter we would like to see if this conjecture is verified in real life and how exactly we can approximate different stages of the attack by precise closed formulas which allow us to predict the outcomes of the attack as exactly as possible.

7.2 How to Predict the Success of ElimLin

7.2.1 Experimental Setup and Notation

We recall the two main and only steps of ElimLin:

1 Find s_i linear equations in the linear span, $i = 0, 1, 2, 3, \ldots$.

2a If $s_i > 0$ eliminate some s_i variables, increment i and try again Step 1.

2b Algorithm terminates when $s_i = 0$ for some i.

In addition and by convention we are going to define a step $i = 0$ which is different than above, but the same as which is implemented in a common implementation of ElimLin [38]. We will assume that s_0 will be the number of linear equations

which already appear in the equations, without executing any linear algebra. This is a convention which allows researchers to distinguish more easily between a "misleading" starting number of variables (which is sometimes artificially inflated due to methods used for equation generation and formal coding) and the "real" or intrinsic number of variables which is there prior to execution or ElimLin.

Definition 13. *[ElimLin progress indicators]*
Let V_{start}, or simply V if there is no confusion, be the initial number of variables. We define by s_i the number of linear/affine equations over GF(2) generated at each stage of the algorithm where by convention s_0 is the number of linear/affine equations over GF(2) already present. We define

$$V^i_{broken} = \sum_{j=0}^{i} s_j \qquad V_{broken} = \sum_{i=0}^{\infty} s_i \tag{7.1}$$

where by convention $s_i = 0$ if algorithm has reached $V^i_{broken} = V$ at an earlier stage. We define also

$$V^i_{broken} = V - \sum_{j=0}^{i} s_j \tag{7.2}$$

and accordingly let $V_{unbroken} = V - \sum_{i=0}^{\infty} s_i$. Overall we will say that the algorithm terminates if $V_{broken} = V$ and $V_{unbroken} = 0$ and we deliberately ignore the fact that some variables could be subject to a brute-force step, cf. FXL method introudced by Courtois and Patarin [73]. Overall our goal is to achieve for a certain i that

$$\frac{V^i_{broken}}{V_{start}} = 1 \qquad or \qquad \frac{V^i_{unbroken}}{V_{start}} = 0 \tag{7.3}$$

To give an example of how the experiments are done[1]: our Simon implementation is available on GitHub [46, 38]; The ElimLin is executed using one of our implementations of ElimLin [46, 38] which has the nice ability to display on screen the number of linear equations generated at each stage/iteration of the algorithm.

[1]The same software setup has also been used at UCL to run a hands-on student lab session on algebraic cryptanalysis of block ciphers [46], which is part of GA18 course on cryptanalysis taught at UCL.

Finally we have developed a deep inspection tool in JAVA available online [38] which looks deeply inside the equations generated by ElimLin, group the equations based on different selection criteria then provide statistics and predictions for the growth rate of each group. For detailed instructions about how to get these values and full experiment results, see appendix A

7.3 The Big Picture

As already explained in Conjecture 7.1.1, we expect that there are at least 3 distinct stages in the ElimLin algorithm. We start with the big picture, by looking at how the value of $V_{unbroken}$ evolves with growing K. Here we use a known plaintext setting with no guessed key bits to break 8 round Simon64/128. As we know, adding new P/C pairs to the ElimLin should create a linear growth in the total number of variables. In our experiment setup and SIMON encodings, each P/C pair adds 192 variables to the starting equation system. We study the unsolved variables left at the end of ElimLin run. The results are shown in Figure 7.1 with a fitting curve using polynomial regression.

In Figure 7.1, we can see a distinction between St0 and St1. In St0 the number of unbroken variables grows linearly with K. When the curve switches to St2, the number of unbroken variables grows much more slowly. This curve shows that the number of new generated equations is growing much faster than linear in St1 and causing a lot of variables eliminated by ElimLin. Clearly, St1-2 is the most fundamental stages of ElimLin. It contains the phase transition from "hard" to "easy".

7.3.1 On Growth Rate in ElimLin

In section 7.3 we looked at the overall unbroken variables as a function on K P/C pairs. We know the total varible growth linearly with K. So based on equation 7.2 we know the solved variables increase faster than linear in St2. Here we look at the growth rate on the number of newly generated equations as a function of K. On Figure 7.2 we show that early stages $s_{0,1,2,3}$ (see definitions in Section 7.2.1) of ElimLin algorithm on 8 rounds of Simon block cipher grows linearly. This means

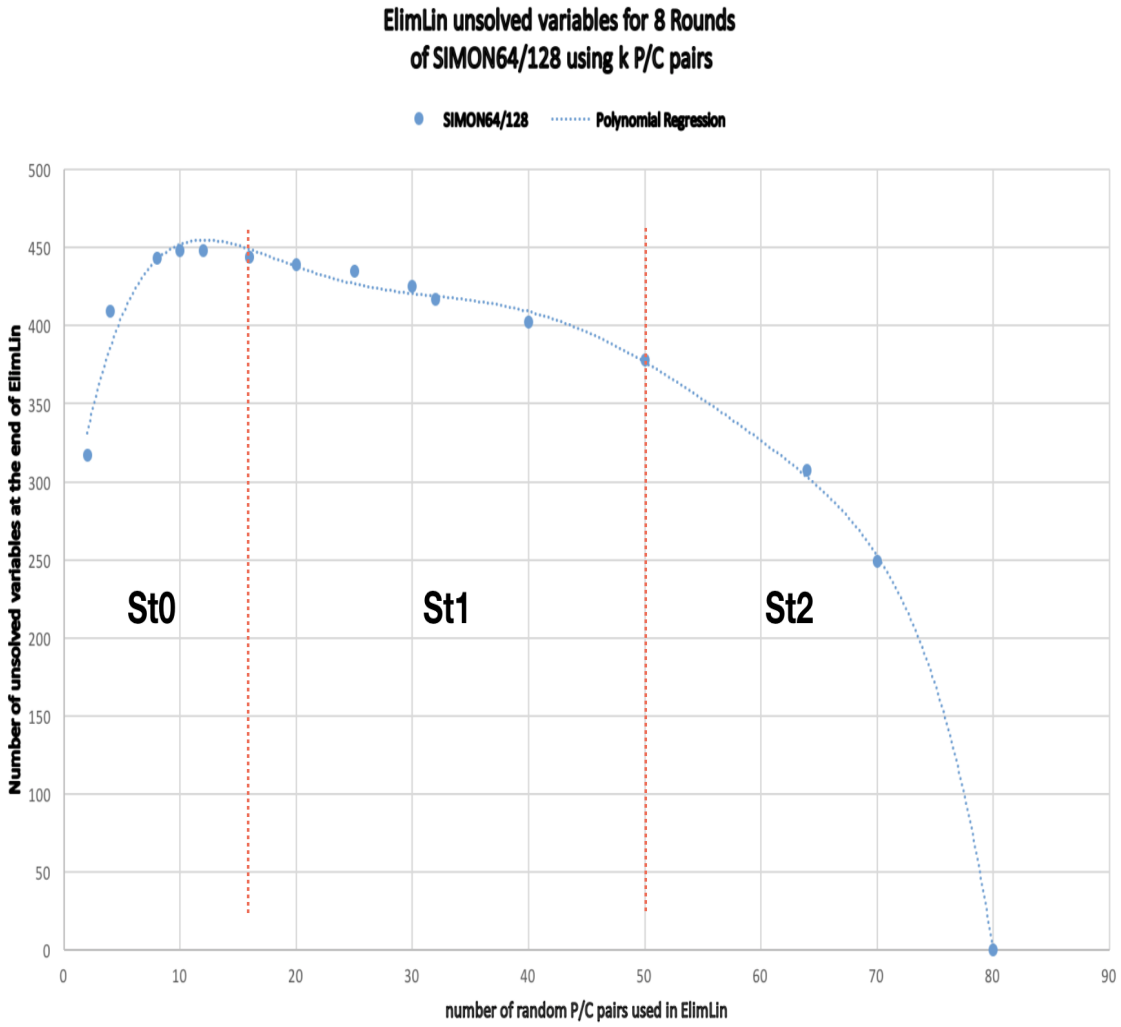


Figure 7.1: Number of variables when ElimLin terminates $V_{unbroken}$ for 8 rounds of Simon 64/128 obtained with our experiments. When $K \leq 16$ the unbroken variables increase. Variables solved by ElimLin are less or equal than variables increased due to added new P/C pairs. We consider this is St0 in Conjecture 7.1.1. When $16 < K \leq 50$ the unsolved variables start to slowly decrease, we consider this is St1. When $K > 50$ unbroken variables decrease much faster than linear and we consider this stage as St2.

at the early stage, ElimLin algorithm only finds trivial equations and the results are very easy to predict.

Moreover a linear progress curve would not be sufficient to break the cipher, because the number of variables also increases linearly. In addition, initially the attack starts with a handicap: below a certain threshold no equation exist at all, or many types of equations (equations with similar size and characteristics, topic which we will study later) seen in larger simulations do not yet exist at all (0 equa-

tion found).

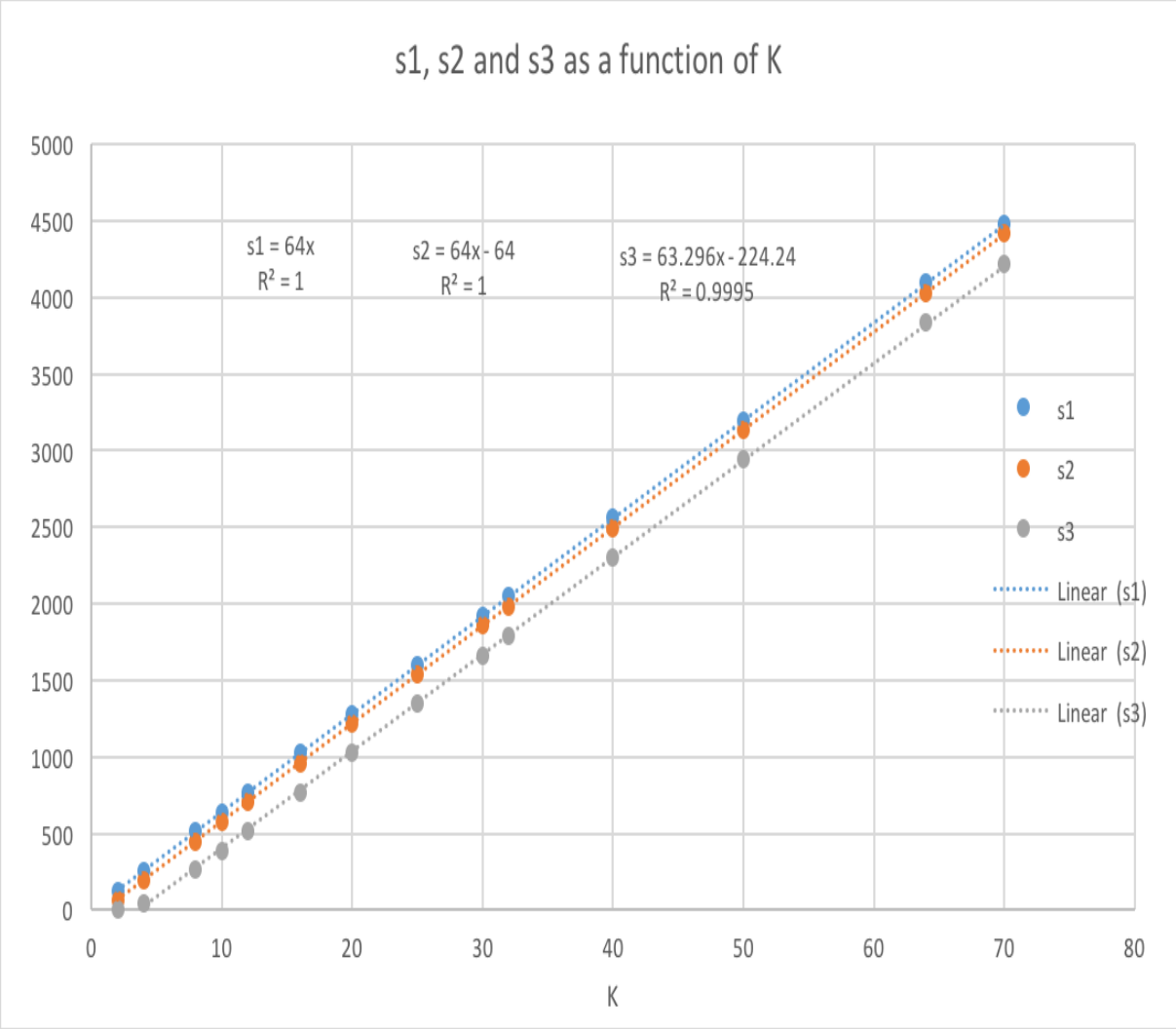


Figure 7.2: Number of linearly independent equations generated at step 1,2,3 of the ElimLin algorithm for 8 rounds of Simon 64/128. The results show in step 1,2,3 of ElimLin, number of equations increase linearly and does not create the faster than linear growth we oberseved in Figure 7.1.

In Figure 7.3, we show the number of linear equations generated at step 4 of the ElimLin algorithm for 8 rounds of Simon block cipher. We have then produced a polynomial interpolation for this data series. It grows faster than linear which is the first stage of the attack with a positive and non-trivial outcome.

Note that we start to have $s_4 > 0$ when $K > 16$. This also aligns with Figure 7.1 when we see a switch from St0 to St1. This is where ElimLin starts to create a growing number of equations which we want to grow faster than linear.

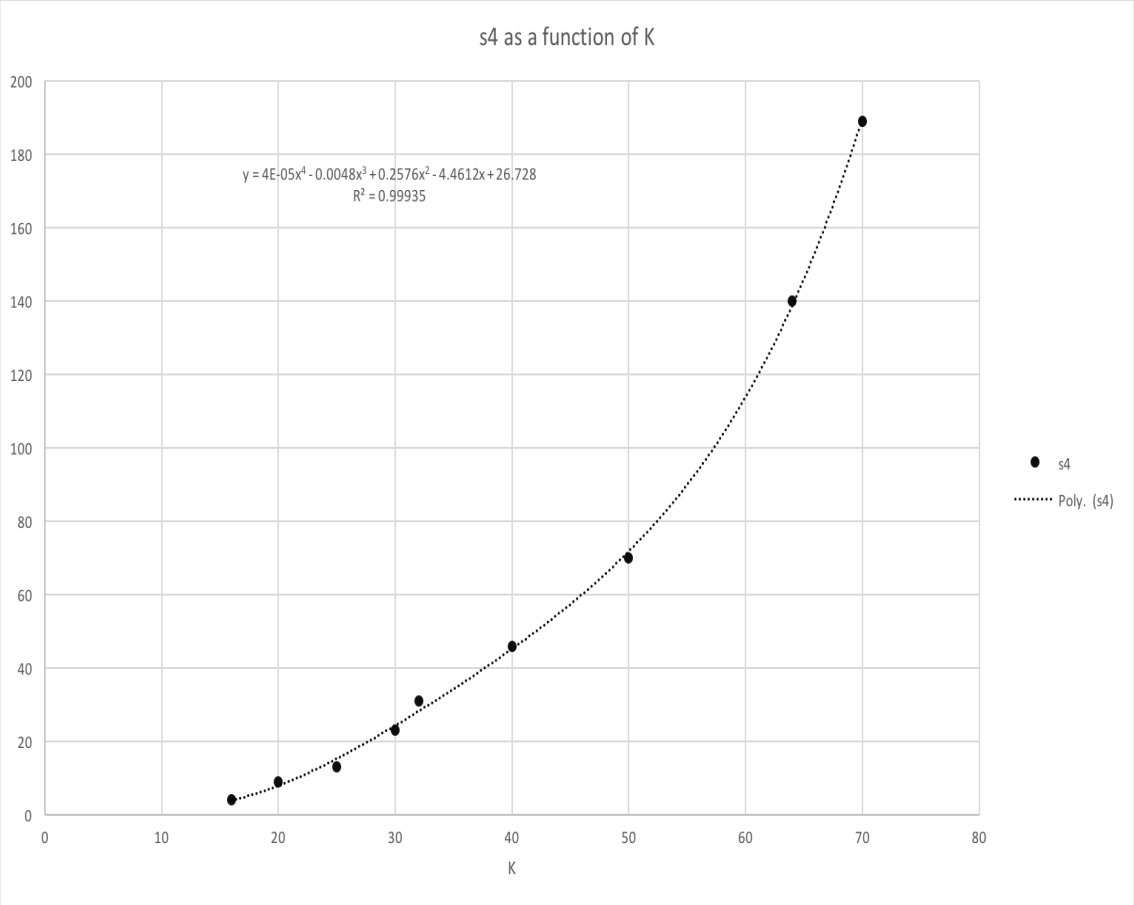


Figure 7.3: Number of linearly independent equations generated at step 4 of the ElimLin algorithm for 8 rounds of Simon 64/128. Step 4 equations appear at K=16, and shows faster than linear increase when $K > 50$, this algains with the stages appeared in Figure 7.1 when curve switch from St0-1 and St1-2.

7.3.2 Deep Inspection

We start to wonder **which** set or type of equations in s_4 grows faster than linear. Up till now we still don't know what kind of equations are found when ElimLin starts to work, and we conjecture that precisely those equations with a fast growth rate are particularly significant and could be a sort of - up to - primary reason why we can break the cipher for some parameters, or in combination with other equations. So we have designed and programmed an open-source inspection tool called "DeepElimlin" [38] to look inside the large number of equations generated by ElimLin and to classify these equations into many detailed types or classes and to visualize and analyse these types in detail. We conjecture that the hardest equations that can be found by are equations that use many different P/C paris and involes variables in the

middle of the encryption. Thus, we decided to group the equations in subcategories by

1. The number of distinct instances (P/C pairs) used: J, where $J \leq K$.
2. The ElimLin execution stage: r.
3. The penetration rounds: maximal and minimal round involved in any of the variables: R_{max} and R_{min}.

We postulated that in most cases the maximal round is the round which is the most deeply embedded or close to the middle round. It is the hardest round to penetrate for any attack, as a matter of fact. Importantly, sometimes we will find equations use variables from both sides (penetrating from both plaintext and ciphertext). In this case we just use the smallest round as minimal and largest round as maximal. It is also important to see that the cipher cannot be broken as long as equations which combine penetration from both sides are not yet generated. This is because equations on the plaintext side could be computed without the knowledge of the ciphertext and vice versa. Then we don't expect to be able to mix these equations very well in the solving process, and identify smaller equations having maybe a unique solution at the end of the solving process, and we don't expect to increase the degree so much that the equations would really interact.

We create sub-categories based on each distinct combination of these values. An example of one equation found in s_4 is given in Figure 7.4, where key variables are compressed as $k\_\{Keybit + Keybit + ...\}$; the intermediate variables (input at each round) are named based on $ZR[\text{instance}]\_[\text{round}]\_[bit]$. In this example we have $J = 3$, $s = 4$, $R_{max} = 4$ and $R_{min} = 5$. This equation belongs to subcategory "s4J3R5-4".

Then we found a few sub-categories in s_4 in which the number of equations grows much faster than linear, see Table 7.1:

We discover that faster-than-linear growth happens when the variables meet in the middle. We have also discovered that the curve of Figure 7.3 splits into different categories in a relatively stable way, so that the curve can be viewed as essentially

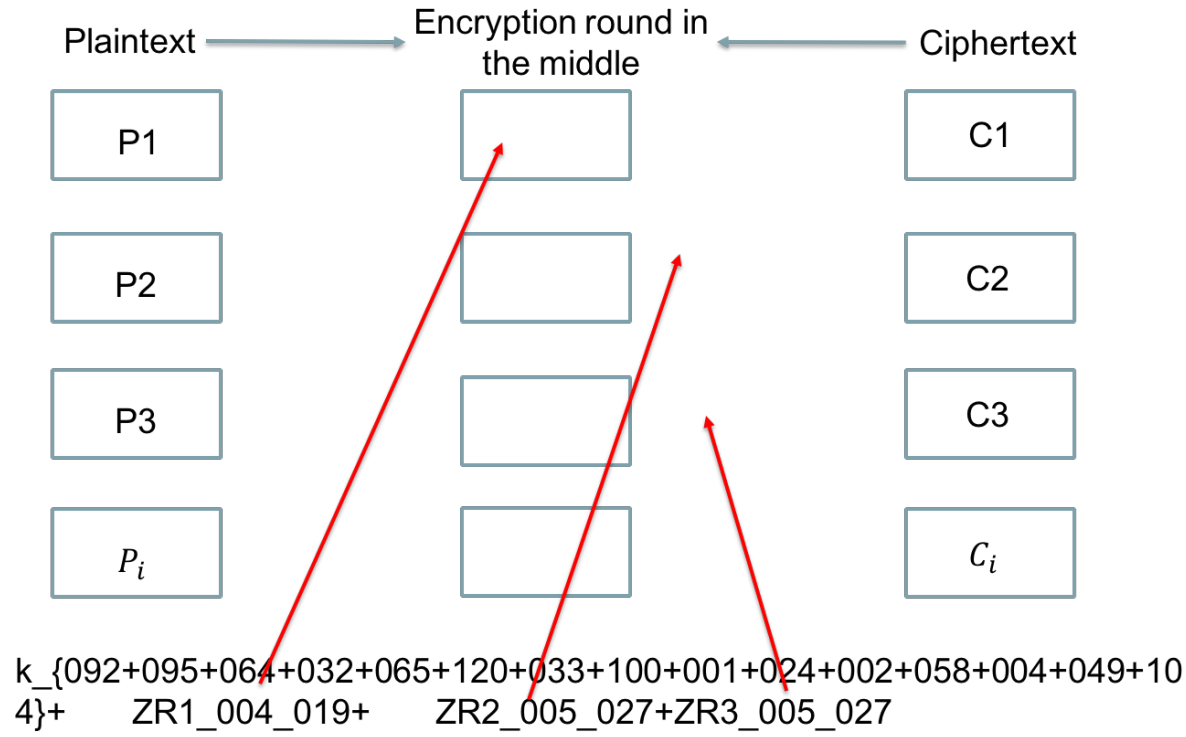


Figure 7.4: Example of a non-trivial equation found by ElimLin. The example equation contains variables from 3 different P/C pairs (ZR1,ZR2 and ZR3), variables appears in mutiple middle rounds (4th and 5th round of encryption). Such equations are considered as non trival equations and increasing faster than linear during ElimLin calculations

Table 7.1: Example of equations growing faster than linear as a function of K. This table shows a selected group of equations that grow faster than linear as a function of K. These equations start to appear after $K \geq 16$ in step 4 (cf. Figure 7.3). Equation group defintaion is explain above, subcategroy s4J4R3-6 means equations contains variables using 4 P/C pairs, variables in 3 to 6 round of SIMON encryption.

# of P/C pairs	16	32	64
s4J2R3-6	2	4	31
s4J3R3-6	2	6	54
s4J4R3-6	1	6	17

a sum of 3 curves of Table 7.1 and other smaller terms. However it appears that this splitting process depends on ElimLin's order of elimination. More precisely linear combinations of certain types of equations will sometimes (rare cases) not be reported in our statistics because they are linearly dependent w.r.t other categories we study. A fully objective classification would require costly computations of the

shortest possible basis for our linear space. This is actually not a problem we have observed here, but we have seen such ambiguities in experiments with our later scaling down heuristic, cf. next section. Current theory only guarantees that the full result of ElimLin is independent on the order of variables [155].

7.4 Known Plaintext vs Chosen Plaintext

Scaling Down Method

One idea behind the scaling method is we can find equations generated for a smaller K inside a larger simulation, if the order of elimination allows it (sometimes they could be linearly combined with other equations). A second idea is that, therefore, just one large ElimLin simulation reveals a lot of information about smaller simulations and about the scalability of the results as K grows. We developed a scaling down software (inside DeepElimLin) which will estimate the number of equations which could be found in a smaller K'. However, such predictions are not always very accurate.

Given a set of equations E generated by ElimLin with K instances, we estimate the number of equations for $K' < K$ as follows:

For each equation $e \in E$, count the equation if the largest index of instance $k < K'$

One example which we found scaling down method works well is the following: in a chosen plaintext model for 32 round of Simon64/128, where we chose $K = 64$ plaintexts in a counter mode, we obtain structured plaintext data and random ciphertext data. We call a cube any such set of plaintexts. Some of these equations will be actually adding the ciphertexts for all the points in the cube, which will be exactly as in the so-called cube attack well-known in the literature [82]. However, many others just consider some of the outputs, and therefore we also discover lots of equations which have plaintexts which form a cube, but which are a lot more complex and may contain arbitrary sums of arbitrary single output bits. Below we present some results extracted from a very large simulation series.

R2 is a collection of results which has the deepest round of 2. R28 works

Table 7.2: Example of how scaling down method works well in a Chosen Plaintext Attack (CPA) for 32 round Simon64/128. As running ElimLin for large number of rounds takes a lot of computational resource, we argue that one can run one experiment (say K=64) and scale down to predict the performance of equations generated by a smaller number of P/C pairs (say K=32). Instead of running both experiments. Scaling down method can save us time to run experiment and the results are relatively relaiable at least for the earlier rounds. This table also shows that when selecting structured P/C pairs, ElimLin can penetrates more rounds and found more linear relations in deeper rounds compare to Known Plaintext Attack (KPA) in the ciphertext side. The results also align with our work in Chapter 6.

	Real 32	Predicted 32	Real 64
R2	224	225.17	448
R3	448	419.67	848
R4	372	336.17	716
R5	104	68	279
R6	123	56.17	327
R7	118	61.83	314
R8	63	41	176
R28	1	0.83	4
R31	1024	1029.33	2048

backwards. The predicted results are averaged based on all possible subsets with the size of K' out of K.

We observe that Chosen Plaintext side penetrates more rounds than the Known Plaintext side, and the equations are shorter and less complex. Predictions are less accurate in the deeper rounds.

7.5 Equations In ElimLin vs. Direct Approximation

Given K random plaintexts, how many linear equations are true for every key which combine variables from K copies of 32 variables inside round Nr of encryption? In addition we know the K plaintexts and will study the equations which exist for some fixed K plaintexts, which means that we will in general find more equations than those true for every plaintext. However, such equations are allowed in ElimLin attack where the plaintext is known. We should also note that many such equations do not depend on the plaintext (some do), and typically the total number of such equations does not depend a lot on the choice of K random plaintexts, though it will be substantially bigger in a Chosen Plaintext Attack (CPA) which penetrates deeper

inside the cipher (see Chapter 6).

These equations can be computed by standard linear algebra method: we need to compute a kernel of a certain Macaulay matrix [classical method in algebraic cryptanalysis]. We write a matrix in which lines are examples of internal data for different keys and columns represent various monomials, which are all linear in this case. The number of keys should be slightly higher than the number of columns, e.g. 100 more. The computation of the kernel of this matrix gives a basis for the equations which are true for every key and related to our monomials. This is a heuristic method which works with a large probability close to 1. For example, if we have P_i/C_i being the plaintext/ciphertext variables, and $R_1, \ldots R_{100}$ are the internal variables, we might discover that for every key and for a fixed plaintext P we have $R_2 \oplus R_7 = 1$. This is, of course, just an example.

7.5.1 Polynomial Approximation in Practice

We have programmed this method of recovery of the equations from the data. The command line is:

```
ax64 9051777 32 128 Nr K 0 /fix80 /seed1
```

This recovers the equations in a totally general Known Plaintext (KP) attack with random plaintexts. For example we obtain:

```
ax64 9051777 32 128 6 2 0  /fix80 /seed1-100
0.05 found on average
ax64 9051777 32 128 6 4 0  /fix80 /seed1-100
0.45 found on average
ax64 9051777 32 128 6 8 0  /fix80 /seed1-100
3.4 found on average
```

We have analyzed the results obtained for 6 and 7 rounds and we obtained:

$$F(6) \approx 0.0002K^2 + 32K - 724$$

$$F(7) \approx 0.00003K^2 + 32K - 12848$$

More generally we conjecture that the quadratic part would disappear for larger K, and it is possible to show that:

Proposition 3 (Theoretical Upper Bound on ElimLin Attack)**.** *The number linear equations which ElimLin can find in KPA scenario is at most:*

$$F \leq 32K - (T - \kappa)$$

where κ is the key size, T is the cardinal of the union of the sets of monomials in k_i which appear in the union of ANF expressions (Algebraic Normal Forms are simply polynomials over $\mathbb{F}_2$) of all outputs of the cipher as a function of the key variables k_i and plaintext variables. of K plaintexts.

Moreover (in practice) this number $T - \kappa$ does not change when we select another set of K random plaintexts and for larger K we have:

$$F_{K \to \infty} \to 32K - (T - \kappa)$$

and starting from a certain threshold K we typically have $F = 32K - (T - \kappa)$ exactly (experimental).

Proof (Sketch): For one plaintext P_i each output is a Boolean function which involves a certain number $T(P_i)$ of monomials in the k_j. Moreover if we concatenate sets of monomials for several plaintexts we expect to quickly reach an upper limit $T(P_i) \leq T$ where T is the total number of monomials in the k_j in the 32 polynomials which are functions of both P_i and the k_j. For every P_i the polynomials contain a subset of these T monomials.

For a fixed plaintext we form a Macaulay matrix in which lines are different keys and different encryptions, and columns are $32K$ variables corresponding to ZL_i after round Nr (left hand side created in Nr round of Simon encryption i out of K), plus κ key variables. Then it will have at most T linearly independent columns, as all columns could be written as linear combinations of other columns which could be added for all possible monomials in key variables, and the κ linear monomials in key variables are those which we already had in equations found by our software.

It is hard to prove $F = 32K - (T - \kappa)$ exactly for larger values, but this is what we observe in practice in algebraic cryptanalysis, for example [73] and here.

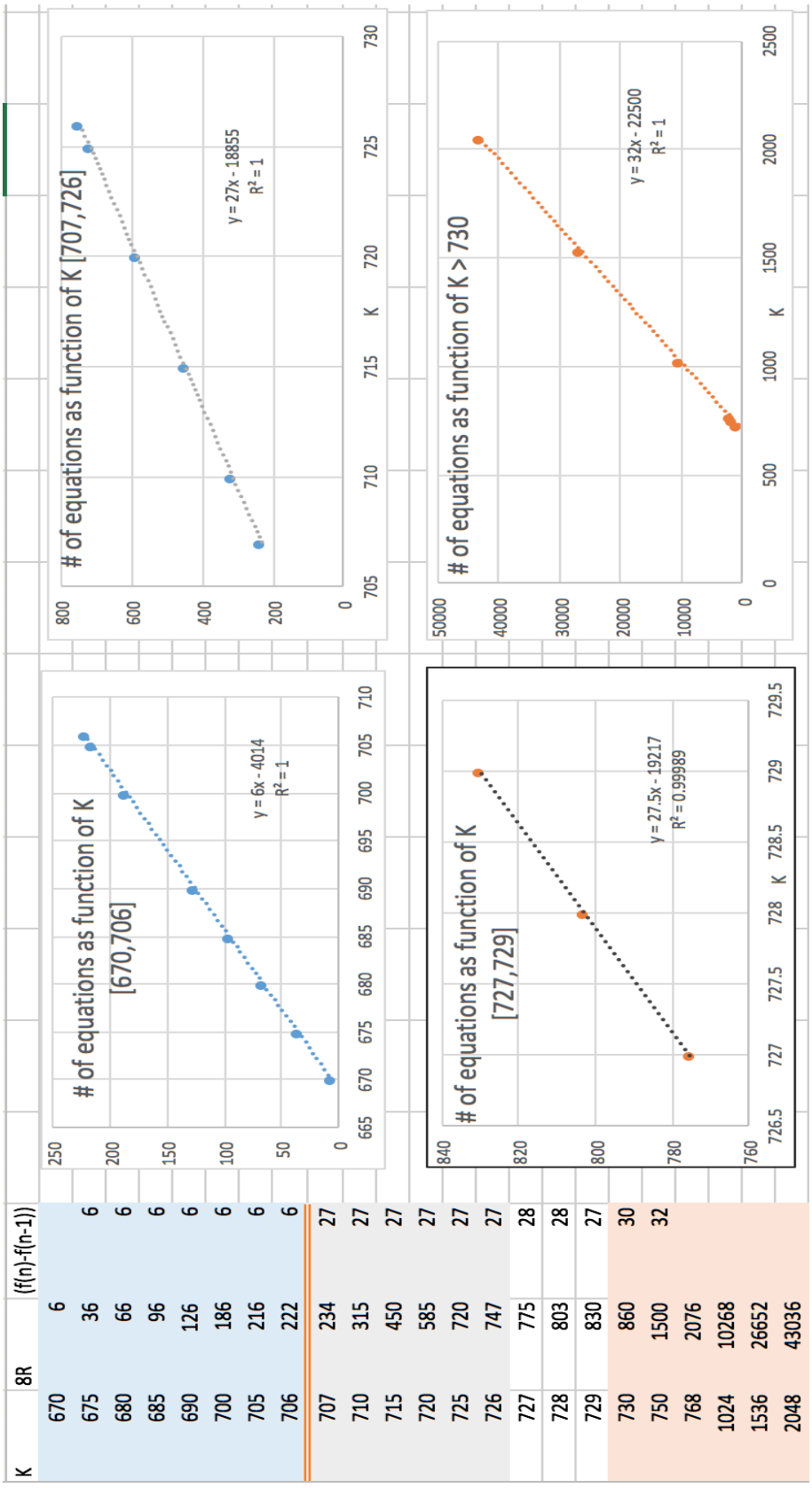
Experimental Results

Here we present a more elaborate example than in our Proposition 3. Our experience show that results of experiments can be predicted with near 100% accuracy and there is typically more than one interval where the number of linear dependencies follow a linear curve with different values of T and for a subset of 32 bits. Our interpretation is that the same sort of result holds for a subset of output bits however for some subsets the value T is smaller because they depend on less key bits. Our experiment results for 8R Simon64/128 KP are given in Figure 7.5. It is clear to see that there are three different stages before it reaches $F = 32K - (T - \kappa)$. For a smaller K we observe $F = 6K - T'$ and then increase to $F = 27K - T''$, where $T' < T'' < T$.

Application — Prediction of Attack Complexity In our research we have NOT yet achieved our goal to be able to reliably predict the complexity of ElimLin attacks which we cannot yet run; but our SECRYPT paper results covered in this and previous sections suggest that this is feasible and here is one possible method to achieve it.

Conjecture 7.5.1.1 (Application of Proposition 3)**.** *We can use the estimation of type $32K - C$ for the number of linearly independent equations which propagate from the plaintext side, another $32K - C$ which will be obtained from the ciphertext side, and estimate that the whole cipher is broken by a meet-in-the middle attack when $2(32K - C) \approx 32K$. Then we can estimate how many additional equations need to be added to ElimLin to complete, and we should be able to conclude that ElimLin+ attack (cf. next section) will break $2Nr$ of rounds as soon as this lower threshold is met.*

This method requires more development in terms of comparing the estimations to actual large ElimLin runs but we are the first to ever propose a plausible method to extrapolate the complexity of an improved form of ElimLin attack.



K	8R	(f(n)-f(n-1))
670	6	
675	36	6
680	66	6
685	96	6
690	126	6
700	186	6
705	216	6
706	222	6
707	234	27
710	315	27
715	450	27
720	585	27
725	720	27
726	747	27
727	775	28
728	803	28
729	830	27
730	860	30
750	1500	32
768	2076	
1024	10268	
1536	26652	
2048	43036	

Figure 7.5: Experiment results for 8R Simon64/128 for theoretical upper bound on ElimLin attack

7.5.2 ElimLin+

We call ElimLin+ any attack which combines ElimLin with additional equations generated by polynomial approximation as above. It is not trivial to see when these extra equations will not already be contained in the linear span found by ElimLin.

An example of ElimLin not finding as many equations as in theoretical upper bound can be found in Table 7.3.

Table 7.3: ElimLin vs Upper bound. We show the number of linear equations actual found by ElimLin and the theoretical upperbound for $K = 1, 2, 4...64$. ElimLin can only found a subset of equations compare to the theoretical upperbound

# of P/C pairs	1	2	4	8	16	32	64
ElimLin R4	11	28.82	46.63	86.39	167.77	336.17	716
Max Theor. R4	23	95	119	247	503	1015	2039

We believe ElimLin, or software algebraic cryptanalysis in general, works well if an attacker can provide additioanal linear equations to the converted equation system. In previous chapters we have discussed two different methods that can improve software algebraic cryptanalysis performance: fixing a selected good subset of key bits in guess-then-solve attack and using well selected P/C pairs that follows truncated differetial properties. Both of these methods can be considred as adding addtional information so that ElimLin or a SAT solve can found more linear independent equations. With these additional equations ElimLin can either solve more variables or speed up the phase transition process. Similarly, SAT solves can find contraditions faster or solve the equation system faster. We conjecture that one can precompute additional linear equations that cannot be found by ElimLin, and provids those equations at the begining of solving step. This will speed up the entire solving process in software algebraic cryptanalysis. However, in this thesis we don't have a concrete example to proof our conjecture. Such work can be considered as future work for this thesis.

7.6 Conclusion

We have identified the source and the nature of cryptanalytic success of ElimLin. It comes from several phenomena: equations emerge by thresholds, they meet in

the middle, they exhibit super-linear growth in some (crucial) cases, and we have discovered how to efficiently generate a lot more equations which ElimLin does not typically find. We discovered that J values can be quite large, but this does not translate into generating particularly large numbers of equations. This is somewhat contrary to intuition: for example equations which exist, even with a very small probability for every triple of encryptions, could grow up to as fast as K^3. This does not work as expected. In fact we have established an upper bound on such extra equations in the Known Plaintext Attack (KPA) scenario. In addition the Chosen Plaintext Attack(CPA) scenario remains a lot more promising. For example it can include cube attacks by Shamir and Dinur [82] , some of which equations we have also already encountered in ElimLin runs and which opens a lot more hard optimization problems with regard to the basis of the cube.

We have all the ingredients. However, designing a really optimal algebraic attack on a block cipher is not an easy task due to the large complexity of the equations we have discovered. We feel that each class of equations could be computed faster by a dedicated method, and we have already computed many substantially faster by our Polynomial Approximation method above. Moreover such methods could also compute several types of equations at once. Future research is needed on how ElimLin needs to be enhanced in practice and what kind trade-offs can be observed between the cost of computing various classes of equations we have already seen and others we have not yet fully integrated in ElimLin+ sort of attacks, as well as the cost of ElimLin itself. Due to sparsity and the specific character of various types of equations, it is not easy to know what is the most efficient method to compute all the equations or their linear equivalents, and it could be true that focusing on some fast growing and very sparse types/classes and ignoring some long equations which we already have will allow us to have an overall faster attack. If we had just one dominating type of equations with fast asymptotic growth, the analysis would be a lot easier. This is not what we observe for Simon and this could be different for another cipher. Similarly, if the statistic were flat maybe there would be little advantage focusing on some equations. We don't observe this either and we are in

the case with several strong classes of equations.

Part III

Speed Optimisation for Bitcoin Brain Wallet Attacks

Chapter 8

Improving Brain Wallet Attacks

In Chapter 4 we described the general concept of Ellipitic Curve and Bitcoin brain walltes. Brain wallets is a deterministic method to generate Bitcoin private keys from a password. This means once an attacker knows the password, they can recover the pravite key and get control of the Bitcoins within the wallet. Brain wallets exist because Bitcoin users prefer to rememeber a password instead of keeping a long random string in a safe place. However, existing research discussed in Section 4.9.1 shows an attacker can simpely use a dictionary attack to guess the passwords. The security of Bitcoin Brain wallets only depends on the strength of the password. The speed of checking if a guessed password leads to a used brain wallet only depends on the key generation process in the Bitcoin elliptic curve secp256k1. In this chapter we will review the current implementation of Bitcoin elliptic curve and provide detailed benchmarks on each elliptic curve opteration during the key geneartion process. At the end we will implement a new attack in order to speed up the key geneartion process and improve the existing attack.

8.1 Bitcoin Elliptic Curve Implementation and Benchmarking

8.1.1 Dedicated Scalar Multiplication Method

The process of cracking Bitcoin brain wallets is to repeatedly generated public keys using guessed passwords. The key generation method as we described in 4.6.2,

is to compute $Q = dP$. Here d is a SHA256 hash of the generated password, P is a fixed point which is the base point G for secp256k1 (see Section 4.8). We first benchmark the current best implementation secp256k1 library. All benchmark results are running on a Intel i7-3520m 2.9GHz laptop (win8 x64).

The time cost for computing one public key given a random private key takes

47.2 us.

8.1.1.1 Fixed Point Multiplication Methods

The most basic and naive method for point multiplication $Q = kP$ with a unknown point P is the double-and-add method [99]. The idea is to use binary representation for k:

$$k = k_0 + 2k_1 + 2^2k_2 + \cdots + 2^m k_m$$

where $[k_0 \ldots k_m] \in \{0,1\}$ and m is the length of k, in Bitcoin elliptic curve, $m = 256$.

Algorithm 5 Double-and-add method for point multiplication of unknown points [99] page 96

INPUT: $k = (k_m, ..., k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$.
OUTPUT: kP.

1: $Q :=$ infinity
2: **for** i from 0 to m **do**
3: if k_i = 1 then $Q := Q + P$ (using point addition)
4: $P := 2P$ (using point doubling)
5: **end for**
6: **return** Q

The expected number of ones in the binary representation of k is approximately $\frac{m}{2}$, so the double-and-add method will need $\frac{m}{2}A + mD$ computations (where A means point addition and D means point doubling) in total. However, if the point P is fixed and some storage is available, then the point multiplication operation $Q = kP$ can be accelerated by pre-computing some data that depends only on P. For example if the points $2P, 2^2P, \ldots, 2^{m-1}P$ are precomputed, then the double-and-add method (Algorithm 5) has an expected running time of $(\frac{m}{2})A$, and all doublings are eliminated.

In [26] the authors introduced a new method for fixed point multiplication. The precomputing step stores every multiple $2^i P$. Let $(K_{d-1},\ldots,K_1,K_0)_{2^w}$ be the base-2^w representation of k, where $d=[m/w]$, and let $Q_j=\sum_{i:K_i=j}2^{wi}P$ for each j, $1\leq j\leq 2^w-1$, Then

$$kP=\sum_{i=0}^{d_1}K_i(2^{wi}P)=\sum_{j=1}^{2^w-1}(j\sum_{i:K_i=j}2^{wi}P=\sum_{j=1}^{2^w-1}jQ_j$$
$$=Q_{2^w-1}+(Q_{2^w-1}+Q_{2^w-2})+\cdots+(Q_{2^w-1}+Q_{2^w-2}+\cdots+Q_1) \quad (8.1)$$

Algorithm 6 Fixed-base windowing method for point multiplication[99]

INPUT: Window width w, $d=[m/w]$, $k=(K_{d-1},\ldots,K_1,K_0)_{2^w}$
OUTPUT: kP

1: Precompute $P_i=2^{wi}P, 0\leq i\leq d-1$
2: $A\leftarrow$ infinity, $B\leftarrow$ infinity
3: **for** j from 2^w-1 down to 1 **do**
4: For each i for which $K_i=j$ do: $B\leftarrow B+P_i$
5: $A\leftarrow A+B$
6: **end for**
7: **return** A

Algorithm 6 has expected running time of

$$(2^w+d-3)A.$$

By reviewing the literature and checking some other existing methods in [99] we noticed they are all memory friendly implementations which do not take a lot of memory for precomputation. However, we are working on a different task and aim at repeatedly running point multiplication method for great many times. We have implemented an extreme version of the window method which will take much more precomputation space than methods introduced in [99].

In our implementation, the precomputation step will compute $P_j=jP$ where $1\leq j\leq 2^w-1$ then for each P_j we compute $P_{i,j}=2^{wi}P_j$, which will cost 2^w-1 times more memory space than algorithm 6, but expected running time for each point multiplication will reduce to approximately $(d-1)A$

Algorithm 7 Our implementation of windowing method with larger precomputation table

INPUT: Window width w, $d = [m/w]$, $k = (K_{d-1}, \ldots, K_1, K_0)_{2^w}$
OUTPUT: kP

1: Precompute $P_{i,j} = 2^{wi} jP, 0 \leq i \leq d-1$ and $1 \leq j \leq 2^w - 1$
2: $A \leftarrow$ infinity
3: **for** i from 0 to $d-1$ **do**
4: $\quad A \leftarrow A + P_{i,j}$ where $j = K_i$
5: **end for**
6: **return** A

We have implemented a code that can take any window width w from 1 to 24[1], our benchmark results are shown in Table 8.1. Note that precomputation stores elliptic curve point $P = x, y$ where x and y are 32 bits integer array of size 10. So one stored point needs 80 Bytes memory space.

Table 8.1: Time cost for different window width w, point addition method secp256k1 library [162] secp256k1_gej_add_ge

	w=4	w=8	w=12	w=16	w=20
d	64	32	22	16	13
number of additions	63	31	21	15	12
precomputation memory	81.92 KB	655.36 KB	7.21 MB	83.89 MB	1.09 GB
time cost	46.36 us	22.76 us	15.35 us	11.23 us	9.23 us

8.1.2 Point Representation

As we described in Section 4.3, representing a point in affine coordinate $P(x, y)$ on an elliptic curve over $\mathbb{F}_p$, the field operations for calculating point addition needs 2 multiplications, 1 square and one modular inverse (for short, 2M+1S+1I). Modular inverse is a more expensive operation compared to multiplication and square. We list our benchmarks using different package in C to demonstrate the difference for modular inverse computation compare to multiplication and square. The packages we have benchmarked are: OpenSSL-1.0.2a (released in March 2015) and mpir-2.5.2 (released in Oct 2012), and the Pieter Wuille's implementation on github [162] [2] .

[1] larger than 22 will take too long for precomputation and my laptop start to have slow response
[2] with the following configuration: USE_NUM_GMP USE_FIELD_10x26 USE_FIELD_INV_NUM USE_SCALAR_8x32 USE_SCALAR_INV_BUILTIN

Table 8.2: Benchmarking OpenSSL and MPIR library for field multiplication, square and modular inverse in affine coordinate

	multiplication	mod p	square	mod p	mod inverse
MPIR	0.07 us	0.15 us	0.13 us	0.15 us	18.0 us
OpenSSL	0.08 us	0.43 us	0.06 us	0.43 us	1.8 us
secp256k1	0.049 us		0.039 us		1.1 us

The results are shown in Table 8.2. The benchmarking shows modular inverse is much more expensive than multiplication and square. It is also important to notice, for OpenSSL Big Number library, a square operation is more expensive than multiplication, and for MPIR library, 1 square = 0.75 multiplication. As modular inverse is more expensive than multiplication, it may be advantageous to represent points using other coordinates.

Projective Coordinates

For elliptic curve over $\mathbb{F}_p$ where the curve equation is $y^2 = x^3 + ax + b$. The standard projective coordinates represent elliptic curve points as $(X : Y : Z)$, $Z \neq 0$, correspond to the affine point $(\frac{X}{Z}, \frac{Y}{Z})$. The projective equation of the elliptic curve is:

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

The point at infinity O corresponds to (0:1:0), where the negative of $(X : Y : Z)$ is $(X : -Y : Z)$

Jacobian Coordinates

Elliptic curve points in Jacobian coordinate are represented in the following format $(X : Y : Z)$, $Z \neq 0$, corresponds to the affine point $(\frac{X}{Z^2}, \frac{X}{Z^3})$. The projective equation of the elliptic curve is

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

The point at infinity O corresponds to (1:1:0), while the negative of $(X : Y : Z)$ is $(X : -Y : Z)$.

The field operations needed for point addition and point doubling are shown in Table 8.3. We see that Jacobian coordinates yield the fastest point doubling, while mixed Jacobian-affine coordinates yield the fastest point addition.

Table 8.3: Operation counts for point addition and doubling. A = affine, P = standard projective, J = Jacobian [99, 29]

Doubling		General addition		Mixed coordinates*	
2A $\rightarrow$ A	1I,2M,2S	A+A $\rightarrow$ A	1I,2M,1S	J+A $\rightarrow$ J	8M,3S
2P $\rightarrow$ P	7M,3S	P+P $\rightarrow$ P	12M,2S		
2J $\rightarrow$ J	4M,4S	J+J $\rightarrow$ J	12M,4S		

* Here mixed coordinates means Jacobian-Affine mixed coordinates, see below for details.

We refer the reader to [99, 29] for other detailed equations in different coordinates. Here we are only interested in point addition functions using mixed coordinates.

Point Addition using Jacobian-Affine Mixed Coordinates

Let $P = (X_1 : Y_1 : Z_1)$ be a Jacobian projective point on elliptic curve $y^2 = x^3 + ax + b$, and $Q = (X_2 : Y_2 : 1)$ be be another point on the curve, suppose that $P \neq \pm Q$, $P + Q = (X_3 : Y_3 : Z_3)$ is computed by the following equations:

$$\begin{aligned} X_3 &= (Y_2 Z_1^3 - Y_1)^2 - (X_2 Z_1^2 - X_1)^2 (X_1 + X_2 Z_1^2) \\ Y_3 &= (Y_2 Z_1^3 - Y_1)(X_1 (X_2 Z_1^2 - X_1)^2 - X_3) - Y_1 (X_2 Z_1^2 - X_1)^3 \\ Z_3 &= (X_2 Z_1^2 - X_1) Z_1 \end{aligned} \tag{8.2}$$

By storing the intermediate elements, X_3, Y_3 and Z_3 can be computed using three field squarings and eight field multiplications as follows:

$$A \leftarrow Z_1^2,\ B \leftarrow Z_1 \cdot A,\ C \leftarrow X_2 \cdot A,\ D \leftarrow Y_2 \cdot B,\ E \leftarrow C - X_1,$$

$$F \leftarrow D - Y_1,\ G \leftarrow E^2,\ H \leftarrow G \cdot E,\ I \leftarrow X_1 \cdot G,$$

$$X_3 \leftarrow F^2 - (H + 2I),\ Y_3 \leftarrow F \cdot (I - X_3) - Y_1 \cdot H,\ Z_3 \leftarrow Z_1 \cdot E.$$

secp256k1 point addition formulas In the latest version, secp256k1 point addition formulas are based on [27] which introduced a strongly unified addition formulas for standard projective coordinate. Bitcoin developers implemented mixed coordinate formula (Jacobian-Affine) version based on [27].

Let $P = (X_1 : Y_1 : Z_1)$ be a Jacobian projective point on elliptic curve $y^2 = x^3 + ax + b$, and $Q = (X_2 : Y_2 : 1)$ be be another point on the curve, suppose that $P \neq \pm Q$, $P + Q = (X_3 : Y_3 : Z_3)$ is computed by the following equations:

$$\begin{aligned} X_3 &= 4(K^2 - H) \\ Y_3 &= 4(R(3H - 2K^2) - G^2) \\ Z_3 &= 2FZ_1 \end{aligned} \tag{8.3}$$

where

$$A = Z_1^2,\ B = Z_1 \cdot A,\ C = X_2 \cdot A,\ D = Y_2 \cdot B,\ E = X_1 + C$$

$$F = Y_1 + D, G = F^2, H = E \cdot G, I = E^2, J = X_1 \cdot C, K = I - J$$

Bernstein-Lange point addition formulas In [16], Bernstein introduced the following method which takes 7M+4S; the explicit formulas are given as following [15]

$$\begin{aligned} X_3 &= r^2 - J - 2V \\ Y_3 &= r \cdot (V - X_3) - 2Y_1 \cdot J \\ Z_3 &= (Z_1 + H)^2 - Z_1^2 - H^2 \end{aligned} \tag{8.4}$$

where

$$U2 = X_2 \cdot Z_1^2,\ S2 = Y_2 \cdot Z_1^3$$

$$H = U2 - X_1,\ I = 4H^2$$

$$J = H \cdot I,\ r = 2(S2 - Y_1),\ V = X_1 \cdot I$$

Detailed Filed Operation Benchmarks

From the results of Table 8.2 we saw that Wuille's secp256k1 library [162] has a much faster field multiplication and square speed than OpenSSL and mpir library. Wuille's field implementation is optimised based on the prime used in secp256k1 curve. Secp256k1 library has 5x52 and 10x26 field implementation for 64 bits

and 32 bits integers [3]. Here we use 10x26 representation and each 256 bits value is represented as a 32 bits integer array with size of 10. We refer readers to file *field_10x26_impl.h* in secp256k1 library for more details. Secp256k1 library already implemented equation 8.3 in method *secp256k1_gej_add_ge_var*, which uses 8 multiplications, 3 squares and 12 multiply integer / addition / negation. Equation 8.2 is implemented in *secp256k1_gej_add_ge* which uses 7 multiplications, 5 squares and 21 multiply integer / addition / negation. We have implemented equation 8.4 which take 7 multiplication, 4 squares and 21 multiply integer / addition / negation.

It is important to notice the square and multiplication difference which we discussed in Table 8.2. In [76] Bernstein listed best operation counts based on different assumptions: S = 0M, S = 0.2M, S= 0.67M, S=0.8M and S=1M. In [33], the author discussed the ratio S/M is almost independent of the field of definition and of the implementation, and can be reasonably taken to equal to 0.8. Our benchmark results are very similar to S = 0.8M (see Table 8.2). In [15], other field operations are considered as 0M. In Table 8.4 our benchmark results show field addition and other operations have approximately 0.1M cost.

Table 8.4: Field operation counts and benchmark results

	#Mul 1M	#Square ≈ 0.8 M	#add/neg/*int ≈ 0.1 M	#fe_cmov ≈ 0.2 M	total time cost
secp256k1_gej _add_ge	7	5	15	6	≈ 0.681 us
secp256k1_gej _add_ge_var	8	3	12	0	≈ 0.562 us
7M + 4S code	7	4	21	0	≈ 0.594 us

The secp256k1_gej_add_ge method, which is also the default method for key generation, uses 6 secp256k1_fe_cmov operations which have a cost approximately 0.2 M. The main reason of writing code in such a way is stated in the code, and the author's comments:

"This formula has the benefit of being the same for both addition of distinct points and doubling"[162]

[3] Depends on whether compiler support 64 bits integer

The purpose of make addition and double using the same function is to prevent side channel attacks. As point doubling is much more cheaper than point addition. Our experiments are done based on the benchmark results of S/M ratio with specified machine setting (earlier in Section 8.1.1) and a specific library configuration (footnote in Section 8.1.2). Different operating systems or library configurations might have different results. One should choose between our code and secp256k1_gej_add_ge method. Detailed benchmark results are given in Table 8.5

Table 8.5: Time cost for different window width *w* for EC key generation

	w=4	w=8	w=12	w=16	w=20
d	64	32	22	16	13
# of additions	63	31	21	15	12
precomputation memory	81.92 KB	655.36 KB	7.21 MB	83.89 MB	1.09 GB
secp256k1_gej _add_ge	45.85 us	22.16 us	15.35 us	11.23 us	9.23 us
secp256k1_gej _add_ge_var	**37.37 us***	17.86 us	12.21 us	8.89 us	**7.16 us**
7M + 4S code	39.01 us	18.79 us	12.77 us	9.23 us	7.48 us
Jacobian to Affine	≈ 10 us				
Benchmark on my laptop	≈ 42 K guesses / sec (single thread) on i7-3520m 2.9 GHz CPU				
DEF CON Attack**	≈ 130 K guesses / sec on i7-2600 3.2 GHz CPU				
Improved DEF CON Attack	≈ 315 K guesses / sec				

* DEF CON attack [31] is equivalent to this results
** Results are reported by Ryan Castellucci running his DEF CON code and our improved code on 8 threads with linux gcc compiler.

DEF CON attack [31] published code on github in Aug using a faster version of secp256k1 library [4], and the results is marked as * in Table 8.5. Our best result using 1.09 GB precomputation memory gives ≈ **2.5 times speed-up** for key generation process than the current known best attack.

Some work in this chapter has been developed into teaching materials for the UCL M.Sc. Information Security course, including low level c/c++ programming

[4] Also written by Pieter Wuille one year ago, this version is performance focused and using 8M+3S

for field operation, ECDSA implementation and countermeasures against side channel attacks.

8.2 On Cracked Brain Wallets

Our work has focused on providing the first detailed benchmark of existing Bitcoin elliptic curve implementations and optimizing the speed of the state-of-art Bitcoin brain wallets attack. We did not pay attention to analysing the cracked brain wallets. We refer the reader to read this paper [159] for a detailed measurement of cracked brain wallets results. In this section we only give a brief summary of our results and discuss some interesting points which are not covered in [159].

We have collected all hash160 on blockchain data (until June 2015). 89,872,723 unique addresses have ever been used, and we have cracked 18,350 addresses in total. All of them were empty at the time we found them. It is clear that the majority of addresses had been registered by a single entity (some sort of bounty or honeypot). 17784 brain wallets were first used on August 31, 2013 and have the same transaction amount 0.00005460 bitcoin or precisely twice this amount. The largest amount ever seen in one brain wallet was 500 BTC.

8.2.1 Network Stress Test

The maximum block size of Bitcoin is one megabyte per block. On 11 November 2014, when the block was around 30% full, David Hunduson wrote a blog post [104] analyzing what will happen as the network approaches 100% full. The author did a simulation which shows that if the block is 100% full, then 10% of all transactions would still not have received a confirmation after 22800 seconds (Bitcoin transcations are normally confirmed after around 600 seconds). Later, on 4th May 2015, Gavin Andersen published his first in a series of blog posts [5] aimed at convincing the Bitcoin community to adopt a larger block size. This sparked a heated debate about the Bitcoin blocksize that broke past the technical developers of the Bitcoin protocol into the broader sphere of people that care about Bitcoin. As a result of this debate, several network stress tests were launched. We measured some aftermath of this contentious test, since the perpetrator / originator chose to

send large quantities of small amounts of bitcoin to selected brain wallets. Between 28th June 2015 and 28th August 2015, we observed 41 addresses used in the network stress test (also known as "July flood attack") with a total number of 1,554,187 transactions and total amount of 46.8 BTC.

8.2.2 Disclosure of Results

Although all the cracked brain wallets are currently empty, we still decided not to publish a full list of cracked passwords or Bitcoin addresses because users can make the same mistake again. Even statistical information about weak wallets helps attackers to steal more bitcoins. We have given some sample passwords in appendix C. One possible way for disclosure is to tag the cracked addresses on blockchain.info. See Figure 8.1. Bitcoin users can see if their brain wallet address is cracked. But the process is not automated due to captcha requirement and we do have a lot of cracked passwords.

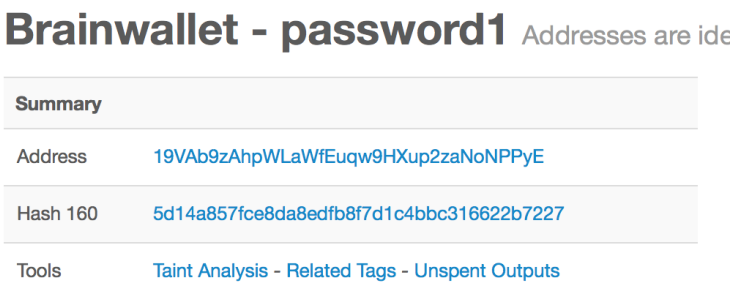


Figure 8.1: Example of tagged brain wallet address

The idea behind Bitcoin brain wallets is elegant: remembering a password or passphrase is surely easier than a private key. Our work and also Vasek's work [159] have made a clear point that it is an extremely insecure way to store bitcoin. There exist lots of other methods to keep bitcoin more secure. As a result of our research, the first and widely suggested brain wallet generation website brainwallet.org has been permanently closed.

8.3 Summary

In this chapter we provided the first detailed benchmarks on Bitcoin elliptic curve secp256k1 and proposed a new attack for craking Bitcoin brain wallets. Our attack improved the speed of the an existing attack by a factor of 2.5. We strongly encourage all Bitcoin brain wallet users to swtich to other wallet types as brain wallets are extremly insecure.

Chapter 9

Conclusion

This thesis mainly focuses on improving algebraic cryptanalysis with software and solvers. Algebraic cryptanalysis is powerful as it requires small quantities of data, but in general the complexity grows quickly as the number of rounds increases. Many mitigations to improve runtimes are studied. We explored different types of optimization processes meant to make algebraic cryptanalysis problems transition from "hard to solve" to "easier to solve" by a software solver. We applied these optimizations to concrete ciphers and demonstrated the improvements. We aim to contribute to future government standards, such as Simon and other widely used ciphers including new releases of GOST. We proposed several possible optimizations for algebraic cryptanalysis and experimentally demonstrated the attacks on GOST and Simon, which were submitted to ISO. We explore many powerful enhancements for algebraic attacks, and in one case we show a result which upper bounds can be obtained, and suggest a new method to predict the complexity of future attacks. We also propose an optimized attack for Bitcoin brain wallet attack.

In Chapter 4, we introduced a new notion of contradiction immunity and SAT immunity, which converts a first stage in cryptanalysis of GOST to an optimization problem. Then we implemented a guess-then-solve attack with a well chosen set of guessed bits. This attack later directly improved the current best attack on GOST. Incidentally GOST was rejected by ISO at that time.

In Chapters 5-6 we studied NSA block cipher Simon which was introduced in 2013 and submitted to ISO in 2015. We introduced a new method that uses well

selected P/C pairs which follow a truncated differential property for algebraic cryptanalysis, and demonstrated the improvement on basic algebraic attacks on Simon with an extremely detailed study of what happens inside the attack and a serious improvement which generates more equations directly. Our work breaks 10/48 rounds of Simon64/128 with less than 10 P/C pairs. We disagree when some researchers believe that Simon should not be studied by academics:

> "because it dignifies [the designs] and wastes the cycles the brain cycles of intelligent people, by going to look at a thing that is produced by a bad actor agency [(the NSA)]."
>
> — Jacob Appelbaum, FSE 2015 invited talk

We propose an opposite view: it is important for the research community to study Simon because it is likely to become an important industry standard in the future. We published the first algebraic cryptanalysis work on Simon in 2014. Today, it is not the best attack. But it is important for the community to notice Simon's low multipicative complexity, low non-linearity and its low security against algebraic cryptanalysis which is also shown in Section 7.5.

In our research we spent a lot of time on contemplating what happens inside ElimLin algorithm. It contains a rich variety of attacks, for example, various generalizations of cube attacks not previously studied. ElimLin is a powerful tool for algebraic cryptanalysis, but with a fundamental limitation on computational complexity. When trying to solve larger number of rounds, the converted problems get much more bigger and ElimLin can not provide an answer within a short time. With a large number of experiments using ElimLin on Simon, we show that precise prediction for ElimLin is possible. We have made progress in both understanding better and extending/enhancing the ElimLin attack. Our discovery method of Section 7.5 suggests that the same equations can yet be computed a lot more efficiently.

Finally, we also looked at the widely used cryptography application – ECDSA in Bitcoin with the secp256k1 curve in Chapter 7. Elliptic curve problems themselves are hard algebraic cryptanalysis problems with complex polynomials and sometimes equations which follow the same topology as in a block cipher. Here

nobody is yet able to propose advanced practical attacks. Another application we studied is Bitcoin. It was invented in 2008 and has grown rapidly since 2012, and it's one of the largest ECC practical applications in the world. We studied how some users manage their private keys and the security pitfalls related to this. Bitcoin uses a special elliptic curve secp256k1 which has not been widely used by any previous application, and in this thesis we provide a detailed benchmark of all the major implementations of this curve, and propose an optimized password cracking attack on Bitcoin brain wallets with a slightly unusual ECC speed optimization. Our work together with other researchers work had made the Bitcoin community aware that brain wallets are extremely insecure.

Appendix A

Full Instruction for ElimLin Prediction Experiments

Software Setup:

1. **Simon.exe**: http://www.nicolascourtois.com/software/simon.exe
   - Source code: https://github.com/GSongHashrate/SimonSpeck
   - Documentation: section Simon inside: http://www.cryptosystem.net/aes/toyciphers.html
2. **ax64.exe**: http://www.nicolascourtois.com/software/ax64.exe
   - Documentation: http://www.cryptosystem.net/aes/tools.html
3. **cryptominisat-2.9.6-win64.exe** http://www.nicolascourtois.com/software/cryptominisat-2.9.6.-win64.exe
   - Sources and documentation: http://www.msoos.org/cryptominisat2/

Command Line:

Simon.exe NR /fixkF /insK /xl0, where NR = number of rounds for the cipher, F = number of fixed/guessed bits for the key and K = number of random plaintext/ciphertext pairs used.

Command Line output might look like:

```
Simon.exe 16 /fixk0 /ins8 /xl0
...
10496+ 512+ 448+ 257+ 1+ 0
Elim[ 14208] 0.529 h 2494/3712 ...
```

In this case NR = 16, F = 0, K = 8. The output is interpreted as follows:

Simon.exe 16 /fixk0 /ins8 /xl0

...

r_0+ r_1+ r_2+ r_3+ r_4+ r_5

Elim[$Total(= r_0 + V_{start})$] 0.529 h $V_{Unbroken}/V_{start}$ ...

Table A.1: Data gathered by running ElimLin on 7 rounds of Simon 64/128

NR	K	V_{start}	$V_{unbroken}$	r_1	r_2	r_3	r_4	r_5	r_6	r_7
7	2	448	248	1472	128	64	8	0		
7	3	608	268	2208	192	128	20	0		
7	4	768	271	2944	256	192	49	0		
7	5	928	262	3680	320	256	89	1	0	
7	6	1088	244	4416	384	320	139	1	0	
7	7	1248	215	5152	448	384	200	1	0	
7	8	1408	188	5888	512	448	259	1	0	
7	9	1568	158	6624	576	512	320	2	0	
7	10	1728	126	7360	640	576	384	2	0	
7	11	1888	82	8096	704	640	448	14	0	
7	12	2048	0	8832	768	704	512	17	47	0

Table A.2: Data gathered by running ElimLin on 8 rounds of Simon 64/128

NR	K	V_{start}	$V_{unbroken}$	r_1	r_2	r_3	r_4	r_5	r_6	r_7
8	2	512	317	1600	128	64	3	0		
8	4	896	409	3200	256	192	39	0		
8	8	1664	443	6400	512	448	261	0		
8	10	2048	448	8000	640	576	384	0		
8	12	2432	448	9600	768	704	512	0		
8	16	3200	444	12800	1024	960	768	4	0	
8	20	3968	439	16000	1280	1216	1024	9	0	
8	25	4928	435	20000	1600	1536	1344	13	0	
8	30	5888	525	24000	1920	1856	1664	23	0	
8	32	6272	417	25600	2048	1984	1792	31	0	
8	40	7808	402	32000	2560	2496	2304	46	0	
8	50	7808	378	40000	3200	3136	2944	70	0	
8	64	12416	307	51200	4096	4032	3840	140	1	0
8	70	13568	249	56000	4480	4416	4224	189	10	0

Appendix B

Java Tool for Deep Inspection of ElimLin

Guangyan Song and Nicolas Courtois: "Java tool for DEEP INSPECTION of equations generated with ElimLin over GF(2) in Cryptalanalysis of Block Ciphers", available at `http://www.nicolascourtois.com/software/DeepElimlin-1.4-SNAPSHOT.jar`. Documentation can be found in the appropriate section of this web page: `http://www.cryptosystem.net/aes/tools.html`.

Appendix C

Examples of Cracked Brainwallet Passwords

We have found over 18,000 cracked Brainwallet passwords. In 2016, our open source tool was given to students for a UCL code breaking competition in module GA18. More than 100 new passwords were found by MSc students: Iason Papapanagiotakis, Jeonghyuk Park, Ellery Smith, Weixiu Tan and Wei Shao. Here we only list some interesting passwords. The full result remains confidential for security reasons.

1. say hello to my little friend
2. to be or not to be
3. Walk Into This Room
4. party like it's 1999
5. yohohoandabottleofrum
6. dudewheresmycar
7. dajiahao
8. hankou
9. {1summer2leo3phoebe

10. 0racle9i
11. andreas antonopoulos
12. Arnold Schwarzenegger
13. blablablablablablabla
14. for the longest time
15. captain spaulding

Bibliography

[1] A russian reference implementation of gost implementing russian algorithms as an extension of tls v1.0 is available as a part of openssl library. the file gost89.c contains eight different sets of s-boxes and is found in openssl 0.9.8 and later at. `http://www.openssl.org/source/`.

[2] Standard specifications for public key cryptography annex a. pages 76–172. IEEE P1363 / D9, Feb 1999.

[3] J. Alizadeh, H. A. Alkhzaimi, M. R. Aref, N. Bagheri, P. Gauravaram, A. Kumar, M. M. Lauridsen, and S. K. Sanadhya. Cryptanalysis of simon variants with connections. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 90–107. Springer, 2014.

[4] H. Alkhzaimi and M. Lauridsen. Differential and linear cryptanalysis of reduced-round simon. In *Cryptology ePrint Archive, Report 2013/543*, 2013.

[5] G. Andersen. Why increasing the max block size is urgent. `http://gavinandresen.ninja/why-increasing-the-max-block-size-is-urgent`, 2015.

[6] A. ANSI. X9. 62: 2005: Public key cryptography for the financial services industry. *The elliptic curve digital signature algorithm (ECDSA)*, 2005.

[7] A. Aysu, E. Gulcan, and P. Schaumont. Simon says, break the area records for symmetric key block ciphers on fpgas. In *Cryptology ePrint Archive, Report 2014/237*, 2014.

[8] R. Balasubramanian and N. Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the menezesokamotovanstone algorithm. *Journal of cryptology*, 11(2):141–145, 1998.

[9] G. Bard, N. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers. 2007.

[10] G. V. Bard, N. T. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf (2) via sat-solvers, 2007.

[11] G. V. Bard, N. T. Courtois, J. Nakahara Jr, P. Sepehrdad, and B. Zhang. Algebraic, aida/cube and side channel analysis of katan family of block ciphers. In *International Conference on Cryptology in India*, pages 176–196. Springer, 2010.

[12] M. Bardet, J.-C. Faugere, and B. Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[13] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management-part 1: General (revised. In *NIST special publication*. Citeseer, 2006.

[14] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck families of lightweight block ciphers. In *Cryptology ePrint Archive, Report 2013/404*, 2013.

[15] D. J. Bernstein and T. Lange. Explicit-formulas database, 2007.

[16] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Advances in cryptology–ASIACRYPT 2007*, pages 29–50. Springer, 2007.

[17] D. J. Bernstein and T. Lange. ebacs: Ecrypt benchmarking of cryptographic systems, 2009.

[18] Ē. Biham and A. Shamir. *Differential cryptanalysis of the data encryption standard*. Springer-Verlag, 1993.

[19] A. Biryukov, A. A. Roy, and V. Velichkov. Differential analysis of block ciphers simon and speck. In *21st International Workshop on Fast Software Encryption, FSE 2014*, 2014.

[20] A. Bogdanov, L. Knudsen, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and Y. Vikkelsoe. Present: An ultra-lightweight block cipher. In *In the proceedings of CHES 2007*, 2007.

[21] C. Bouillaguet, P. Derbez, O. Dunkelman, P.-A. Fouque, N. Keller, and V. Rijmen. Low-data complexity attacks on aes. *IEEE Transactions on Information Theory*, 58(11):7002–7017, 2012.

[22] J. Boyar, M. Find, and R. Peralta. Four measures of nonlinearity. In *In Algorithms and Complexity, pp. 61-72. Springer Berlin Heidelberg*, 2013.

[23] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. 2010.

[24] J. Boyar and R. Peralta. A depth-16 circuit for the aes s-box. 2011.

[25] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis. 2000.

[26] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation. In *Advances in CryptologyEUROCRYPT92*, pages 200–207. Springer, 1993.

[27] E. Brier and M. Joye. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography*, pages 335–345. Springer, 2002.

[28] L. Brown, J. Pieprzyk, and J. Seberry. Lokia cryptographic primitive for authentication and secrecy applications. In *Advances in CryptologyAUSCRYPT'90*, pages 229–236. Springer, 1990.

[29] M. Brown, D. Hankerson, J. López, and A. Menezes. *Software implementation of the NIST elliptic curves over prime fields*. Springer, 2001.

[30] C. Canniere, O. Dunkelman, and M. Knezevic. Katan and ktantan a family of small and efficient hardware-oriented block ciphers. In *In Christophe Clavier and Kris Gaj, editors,Cryptographic Hardware and Embedded Systems - CHES 2009, volume 5747 of Lecture Notes in Computer Science, Springer Berlin Heidelberg*, 2009.

[31] R. Castellucci. Cracking cryptocurrency brainwallets. `https://www.defcon.org/html/defcon-23/dc-23-index.html`.

[32] S. Certicom. Sec 2: Recommended elliptic curve domain parameters. *Proceeding of Standards for Efficient Cryptography, Version*, 1, 2000.

[33] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology ASIACRYPT98*, pages 51–65. Springer, 1998.

[34] D. Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[35] D. Coppersmith. The development of des. *Invited talk at CRYPTO*, 2000.

[36] N. Courtois. 100 years of cryptanalysis: Compositions of permutations. `http://www.nicolascourtois.com/papers/code_breakers_enigma_block_teach.pdf`. non-commutative combinations of permutations, used teaching GA18 Cryptanalysis course at University College London 2014-2016.

[37] N. Courtois. Guiding principles of effective crypto and security engineering. http://www.nicolascourtois.com/papers/sc/Pr_Crypto_eng_v12.pdf.

[38] N. Courtois. Algebraic cryptanalysis software,. http://www.cryptosystem.net/aes/tools.html, 2000-2016.

[39] N. Courtois. Ctc2 and fast algebraic attacks on block ciphers revisited. In *eprint*. eprint.iacr.org/2007/152/, 2007.

[40] N. Courtois. How fast can be algebraic attacks on block ciphers? In *Dagstuhl Seminar 07021, Symmetric Cryptography*. dagstuhl.de, 2007.

[41] N. Courtois. Basic equation solving toolbox - http://www.cryptosystem.net/aes/tools.html. 2010.

[42] N. Courtois. Algebraic complexity reduction and cryptanalysis of gost. *IACR Cryptology ePrint Archive*, 2011:626, 2011.

[43] N. Courtois. Half of all elliptic curves broken??? http://blog.bettercrypto.com/?p=1544, 2015.

[44] N. Courtois. Algebraic attacks vs. design of block and stream ciphers. In *slides used in UCL GA18 course "Cryptanalysis", University College London*. http://www.nicolascourtois.com/papers/algat_all_teach_2015.pdf, 2016.

[45] N. Courtois. Computer security foundations and principles standard version. In *slides used in UCL course "Computer Security", University College London*. http://www.nicolascourtois.com/papers/compsec/CompSec_Intro_01.ppt, 2016.

[46] N. Courtois. Software and algebraic cryptanalysis lab,. In *University College London*. http://www.nicolascourtois.com/papers/ga18/AC_Lab1_ElimLin_Simon_CTC2.pdf, 2016.

[47] N. Courtois and G. Bard. Algebraic cryptanalysis of the data encryption standard. In *In IMA Int. Conf. volume 4887, Springer*, 2007.

[48] N. Courtois and B. Debraize. Specific s-box criteria in algebraic attacks on block ciphers with several known plaintexts. In *WEWoRC 2007, pp 100-113*. Springer, 2008.

[49] N. Courtois, D. Hulme, and T. Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis. In *In electronic proceedings of 2nd IMA Conference Mathematics in Defence 2011*, 2011.

[50] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–407. Springer, 2000.

[51] N. Courtois and M. Misztal. Differential cryptanalysis of gost. *IACR Cryptology ePrint Archive*, 2011:312, 2011.

[52] N. Courtois, T. Mourouzis, and D. Hulme. Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits. In *To Appear in IARIA Journal: IntSys13v6n34*, 2013.

[53] N. Courtois, T. Mourouzis, and G. Song. Reference implementation of simon and speck and a basic generator of equations - https://github.com/gsonghashrate/simonspeck/. 2014.

[54] N. Courtois, T. Mourouzis, G. Song, P. Sepehrdad, and P. Susil. Combined algebraic and truncated differential cryptanalysis on reduced-round simon. In *SECRYPT*, pages 399–404, 2014.

[55] N. Courtois, K. Nohl, and S. O'Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.

[56] N. Courtois, P. Sepehrdad, P. Susil, and S. Vaudenay. Elimlin algorithm revisited. In *Fast Software Encryption, pp. 306-325, Springer Berlin Heidelberg*, 2012.

[57] N. T. Courtois. Feistel schemes and bi-linear cryptanalysis. In *Advances in Cryptology–CRYPTO 2004*, pages 23–40. Springer, 2004.

[58] N. T. Courtois. The dark side of security by obscurity - and cloning mifare classic rail and building passes, anywhere, anytime. In *Proceedings of the International Conference on Security and Cryptography (ICETE 2009)*, pages 331–338, 2009.

[59] N. T. Courtois. An improved differential attack on full gost. *IACR Cryptology ePrint Archive*, 2012:138, 2012.

[60] N. T. Courtois. Security evaluation of gost 28147-89 in view of international standardisation. *Cryptologia*, 36(1):2–13, 2012.

[61] N. T. Courtois. Cryptanalysis of gost in the multiple-key scenario. *Tatra Mountains Mathematical Publications*, 57(1):45–63, 2013.

[62] N. T. Courtois. Low-complexity key recovery attacks on gost block cipher. *Cryptologia*, 37(1):1–10, 2013.

[63] N. T. Courtois. High saturation complete graph approach for ec point decomposition and ecdl problem. Cryptology ePrint Archive, Report 2016/704, 2016. `http://eprint.iacr.org/2016/704`.

[64] N. T. Courtois and G. V. Bard. Algebraic cryptanalysis of the data encryption standard. In *Cryptography and Coding*, pages 152–169. Springer, 2007.

[65] N. T. Courtois, G. V. Bard, and D. Wagner. Algebraic and slide attacks on keeloq. In *Fast Software Encryption*, pages 97–115. Springer, 2008.

[66] N. T. Courtois, G. Castagnos, and L. Goubin. What do des s-boxes say to each other? *IACR Cryptology ePrint Archive*, 2003:184, 2003.

[67] N. T. Courtois and B. Debraize. Algebraic description and simultaneous linear approximations of addition in snow 2.0. In *Information and Communications Security*, pages 328–344. Springer, 2008.

[68] N. T. Courtois, J. A. Gawinecki, and G. Song. Contradiction immunity and guess-then-determine attacks on gost. *Tatra Mountains Mathematical Publications*, 53(1):65–79, 2012.

[69] N. T. Courtois and M. Misztal. First differential attack on full 32-round gost. In *Information and Communications Security*, pages 216–227. Springer, 2011.

[70] N. T. Courtois and M. Misztal. Aggregated differentials and cryptanalysis of pp-1 and gost. *Periodica Mathematica Hungarica*, 65(2):177–192, 2012.

[71] N. T. Courtois, T. Mourouzis, M. Misztal, J.-J. Quisquater, and G. Song. Can gost be made secure against differential cryptanalysis? *Cryptologia*, 39(2):145–156, 2015.

[72] N. T. Courtois, S. ONeil, and J.-J. Quisquater. Practical algebraic attacks on the hitag2 stream cipher. In *Information Security*, pages 167–176. Springer, 2009.

[73] N. T. Courtois and J. Patarin. About the xl algorithm over gf (2). In *Cryptographers Track at the RSA Conference*, pages 141–157. Springer, 2003.

[74] N. T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in CryptologyASIACRYPT 2002*, pages 267–287. Springer, 2002.

[75] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[76] T. L. Daniel J. Bernstein. Explicit-formulas database. `https://www.hyperelliptic.org/EFD/`.

[77] D. W. Davies and W. L. Price. *Security for computer networks: and introduction to data security in teleprocessing and electronic funds transfer*. John Wiley & Sons, Inc., 1989.

[78] H. Delfs, H. Knebl, and H. Knebl. *Introduction to cryptography*, volume 2. Springer, 2002.

[79] H. Deukjo, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. Hight: A new block cipher suitable for low-resource device. In *In Louis Goubin and Mitsuru Matsui, editors, Cryptographic Hardware and Embedded Systems, CHES 2006, volume 4249 of Lecture Notes in Computer Science*, 2006.

[80] C. Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147(01):75–104, 2011.

[81] I. Dinur, O. Dunkelman, and A. Shamir. Improved attacks on full gost. In *Fast Software Encryption*, pages 9–28. Springer, 2012.

[82] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 278–299. Springer, 2009.

[83] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985.

[84] J. Erickson, J. Ding, and C. Christensen. Algebraic cryptanalysis of sms4: Gröbner basis attack and sat attack compared. In *Information, Security and Cryptology–ICISC 2009*, pages 73–86. Springer, 2010.

[85] A. Farzaneh, E. List, S. Lucks, and J. Wenzel. Differential and linear cryptanalysis of reduced-round simon. In *Cryptology ePrint Archive, Report 2013/526*, 2013.

[86] J.-C. Faugere. A new efficient algorithm for computing gröbner bases (f 4). *Journal of pure and applied algebra*, 139(1):61–88, 1999.

[87] J.-C. Faugere. A new efficient algorithm for computing gröbner bases (f 4). *Journal of pure and applied algebra*, 139(1):61–88, 1999.

[88] J.-C. Faugère, P. Gaudry, L. Huot, and G. Renault. Using symmetries in the index calculus for elliptic curves discrete logarithm. *Journal of cryptology*, 27(4):595–635, 2014.

[89] J.-C. Faugère, L. Perret, C. Petit, and G. Renault. Improving the complexity of index calculus algorithms in elliptic curves over binary fields. In *Advances in Cryptology–EUROCRYPT 2012*, pages 27–44. Springer, 2012.

[90] H. Feistel. Cryptography and computer privacy. *Scientific american*, 228:15–23, 1973.

[91] P. FIPS. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 2000.

[92] A. S. Fraenkel and Y. Yesha. Complexity of solving algebraic equations. *Information Processing Letters*, 10(4):178–179, 1980.

[93] S. Galbraith. Elliptic curve discrete logarithm problem in characteristic two. `https://ellipticnews.wordpress.com/2015/04/13/elliptic-curve-discrete-logarithm-problem-in-characteristic-two` 2015.

[94] P. Gaudry. Index calculus for abelian varieties and the elliptic curve discrete logarithm problem. *IACR Cryptology ePrint Archive*, 2004:73, 2004.

[95] Z. Gong, S. Nikova, and Y. Law. Klein: A new family of lightweight block ciphers. In *In Ari Juels and Christof Paar, editors, RFID. Security and Privacy, volume 7055 of Lecture Notes in Computer Science*, 2012.

[96] G. S. GOST. 28147-89,. *Cryptographic protection for data processing systems, Government Committee of the USSR for Standards*, 1989.

[97] I. W. Group et al. Ieee 1363-2000: Standard specifications for public key cryptography. *IEEE Standard, IEEE, New York, NY*, 10017, 2000.

[98] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The led block cipher. In *In Bart Preneel and Tsuyoshi Takagi, editors, Cryptographic Hardware and Embedded Systems, CHES 2011, volume 6917 of Lecture Notes in Computer Science, Springer Berlin Heidelberg*, 2011.

[99] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[100] T. J. Hodges, S. D. Molina, and J. Schlather. On the existence of semi-regular sequences. *arXiv preprint arXiv:1412.7865*, 2014.

[101] S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. Differential cryptanalysis of tea and xtea. In *International Conference on Information Security and Cryptology*, pages 402–417. Springer, 2003.

[102] J. Huang and X. Lai. What is the effective key length for a block cipher: an attack on every practical block cipher. *Science China Information Sciences*, 57(7):1–11, 2014.

[103] Y.-J. Huang, C. Petit, N. Shinohara, and T. Takagi. Improvement of faugere et al.s method to solve ecdlp. In *Advances in Information and Computer Security*, pages 115–132. Springer, 2013.

[104] D. Hunduson. Bitcoin traffic bulletin. `http://hashingit.com/analysis/34-bitcoin-traffic-bulletin`, 2015.

[105] T. Isobe. A single-key attack on the full gost block cipher. *Journal of cryptology*, 26(1):172–189, 2013.

[106] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, 2001.

[107] D. Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster, 1996.

[108] B. S. Kaliski Jr and M. J. Robshaw. Linear cryptanalysis using multiple approximations. In *Advances in CryptologyCrypto94*, pages 26–39. Springer, 1994.

[109] O. Kara. Reflection cryptanalysis of some ciphers. In *Progress in Cryptology-INDOCRYPT 2008*, pages 294–307. Springer, 2008.

[110] J. Kilian and P. Rogaway. How to protect des against exhaustive key search. In *Advances in CryptologyCRYPTO96*, pages 252–267. Springer, 1996.

[111] L. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer Berlin Heidelberg, 1995.

[112] L. R. Knudsen. Cryptanalysis of loki. In *Advances in CryptologyASIACRYPT'91*, pages 22–35. Springer, 1993.

[113] L. R. Knudsen. Block ciphers: analysis, design and applications. *DAIMI Report Series*, 23(485), 1994.

[114] L. R. Knudsen. Block ciphersa survey. In *State of the Art in Applied Cryptography*, pages 18–48. Springer, 1998.

[115] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[116] M. Kosters and S. L. Yeo. Notes on summation polynomials. *arXiv preprint arXiv:1503.08001*, 2015.

[117] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of cryptology*, 14(4):255–293, 2001.

[118] C. Lim and T. Korkishko. mcrypton - a lightweight block cipher for security of low-cost rfid tags and sensors. In *In Joo-Seok Song, Taekyoung Kwon,*

and Moti Yung, editors, Information Security Applications, volume 3786 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006.

[119] J. L. Massey. Safer k-64: A byte-oriented block-ciphering algorithm. In *Fast Software Encryption*, pages 1–17. Springer, 1994.

[120] M. Matsui. Linear cryptanalysis method for des cipher. In *Advances in CryptologyEUROCRYPT93*, pages 386–397. Springer, 1994.

[121] M. Matsui and A. Yamagishi. A new method for known plaintext attack of feal cipher. In *Advances in CryptologyEurocrypt92*, pages 81–91. Springer, 1993.

[122] V. S. Miller. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 417–426. Springer, 1985.

[123] T. Mourouzis. *Optimizations in algebraic and differential cryptanalysis*. PhD thesis, UCL (University College London), 2015.

[124] J. Nakahara Jr, P. Sepehrdad, B. Zhang, and M. Wang. Linear (hull) and algebraic cryptanalysis of the block cipher present. In *International Conference on Cryptology and Network Security*, pages 58–75. Springer, 2009.

[125] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[126] NSA. Cnsa suite and quantum computing faq. "`https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm`, 2016.

[127] NSA. Recommendation for key management. "`http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf`, 2016.

[128] E. Pasalic. Probabilistic versus deterministic algebraic cryptanalysisa performance comparison. *IEEE Transactions on information theory*, 55(11):5233–5240, 2009.

[129] C. Petit, M. Kosters, and A. Messeng. Algebraic approaches for the elliptic curve discrete logarithm problem over prime fields. In *IACR International Workshop on Public Key Cryptography*, pages 3–18. Springer, 2016.

[130] C. Petit and J.-J. Quisquater. On polynomial systems arising from a weil descent. In *Advances in Cryptology–ASIACRYPT 2012*, pages 451–466. Springer, 2012.

[131] A. Poschmann, S. Ling, and H. Wang. 256 bit standardized crypto for 650 ge–gost revisited. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 219–233. Springer, 2010.

[132] N. F. PUB. 46-3. data encryption standard. *Federal Information Processing Standards, National Bureau of Standards, US Department of Commerce*, 1977.

[133] H. Raddum. Algebraic analysis of the simon block cipher family. In *International Conference on Cryptology and Information Security in Latin America*, pages 157–169. Springer, 2015.

[134] H. Raddum and I. Semaev. New technique for solving sparse equation systems. *IACR Cryptology ePrint Archive*, 2006:475, 2006.

[135] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[136] D. E. Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[137] B. Schneier. Applied cryptography. protocols, algorithms, and source code in c/bruce schneier, 1996.

[138] B. Schneier. *Section 14.1 GOST (2nd ed.) Applied cryptography*. john wiley & sons, 1996.

[139] B. Schneier. *Beyond fear: Thinking sensibly about security in an uncertain world*. Springer Science & Business Media, 2006.

[140] I. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

[141] I. Semaev. Sparse algebraic equations over finite fields. *SIAM Journal on Computing*, 39(2):388–409, 2009.

[142] I. Semaev. New algorithm for the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2015/310, 2015.

[143] I. Semaev and M. Mikus. Methods to solve algebraic equations in cryptanalysis. In *In Tatra Mountains Mathematic Publications, Vol. 45, pp. 107-136*, 2010.

[144] P. Sepehrdad. *Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2012.

[145] C. Shannon. Communication theory of secrecy systems. In *Bell System Technical Journal 28*, 1949.

[146] C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[147] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An ultra-lightweight blockcipher. In *In Bart Preneel and Tsuyoshi Takagi, editors, Cryptographic Hardware and Embedded Systems, CHES 2011, volume 6917 of Lecture Notes in Computer Science, Springer Berlin Heidelberg*, 2011.

[148] G. J. Simmons et al. *Contemporary cryptology*, volume 3. IEEE press New York, 1992.

[149] G. Song. Simon and speck implementation in c. `https://github.com/gsonghashrate/SimonSpeck`.

[150] M. Soos, K. Nohl, and C. Castelluccia. Extending sat solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 244–257. Springer, 2009.

[151] N. Sorensson and N. Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.

[152] N. Sörensson, N. Eén, and M. Soos. Cryptominisat 2.92, an open-source sat solver package based on earlier minisat software.

[153] F. Standaert, G. Piret, G. Rouvroy, J. Quisquater, and J. Legat. Iceberg : An involutional cipher efficient for block encryption in reconfigurable hardware. In *In Bimal Roy and Willi Meier, editors, Fast Software Encryption, volume 3017 of Lecture Notes in Computer Science*, 2004.

[154] M. Sugita, K. Kobara, and H. Imai. Security of reduced version of the block cipher camellia against truncated and impossible differential cryptanalysis. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 193–207. Springer, 2001.

[155] N. C. P. S. P. Susil and S. Vaudenay. Elimlin algorithm revisited. In *FSE 2012*. Springer, 2012.

[156] P. Susil, P. Sepehrdad, S. Vaudenay, and N. Courtois. On selection of samples in algebraic attacks and a new technique to find hidden low degree equations. *International Journal of Information Security*, 15(1):51–65, 2016.

[157] P. Sušil, P. Sepehrdad, S. Vaudenay, and N. Courtois. On selection of samples in algebraic attacks and a new technique to find hidden low degree equations. *International Journal of Information Security*, 15(1):51–65, 2016.

[158] A. Tardy-Corfdir and H. Gilbert. A known plaintext attack of feal-4 and feal-6. In *Annual International Cryptology Conference*, pages 172–182. Springer, 1991.

[159] M. Vasek, J. Bonneau, C. K. Ryan Castellucci, and T. Moore. The bitcoin brain drain: A short paper on the use and abuse of bitcoin brain wallets. *Financial Cryptography and Data Security, Lecture Notes in Computer Science. Springer*, 2016.

[160] N. Wang, X. Wang, K. Jia, and J. Zhao. Improved differential attacks on reduced simon versions. *IACR Cryptology ePrint Archive*, 2014:448, 2014.

[161] Q. Wang, Z. Liu, K. Varıcı, Y. Sasaki, V. Rijmen, and Y. Todo. Cryptanalysis of reduced-round simon32 and simon48. In *International Conference in Cryptology in India*, pages 143–160. Springer, 2014.

[162] P. Wullie. bitcoin secp256k1 library, version 2015/08/11. `https://github.com/bitcoin/secp256k1`.

[163] B.-Y. Yang, J.-M. Chen, and N. T. Courtois. On asymptotic security estimates in xl and gröbner bases-related algebraic cryptanalysis. In *International Conference on Information and Communications Security*, pages 401–413. Springer, 2004.