

# A Thesis Title

*Author Name*

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
**Doctor of Philosophy**  
of  
**University College London.**

Department of Something  
University College London

August 9, 2016

I, Author Name, 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

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

# Acknowledgements

Acknowledge all the things!

# Contents

<b>1</b>	<b>Introductory Material</b>	<b>10</b>
<b>2</b>	<b>Introduction to Cryptography</b>	<b>11</b>
2.1	Symmetric and Asymmetric Cryptography . . . . .	12
2.2	Block Ciphers . . . . .	14
2.2.1	Substitution Ciphers . . . . .	15
2.2.2	Transposition Systems . . . . .	16
2.2.3	Product Ciphers . . . . .	16
<b>3</b>	<b>Cryptanalysis of Block Ciphers</b>	<b>19</b>
3.1	Classification of Attacks . . . . .	20
3.2	Brute-force Attack . . . . .	22
3.3	Linear Cryptanalysis . . . . .	22
3.4	Differential Cryptanalysis . . . . .	23
3.5	Algebraic Attacks . . . . .	26
3.5.1	Algebraic Attacks Solving Stage . . . . .	27
<b>4</b>	<b>Cryptanalysis of GOST</b>	<b>31</b>
4.1	GOST encryption standard . . . . .	31
4.1.1	GOST And ISO Standardisation. . . . .	31
4.1.2	Cryptanalysis of GOST . . . . .	32
4.2	The Internal Structure of GOST . . . . .	32
4.3	Algebraic Cryptanalysis with Complexity Reduction . . . . .	33
4.3.1	Reductions and Black-Box Reductions . . . . .	34

4.3.2	Optimizing The Reductions: Amplification . . . . .	35
4.3.3	Working at The Low Level . . . . .	36
4.3.4	Applications of UNSAT/SAT Immunities . . . . .	37
4.4	Application to DES . . . . .	38
4.5	Application to GOST . . . . .	39
4.5.1	SAT Immunity of GOST . . . . .	40
4.6	A Mixed Attack with 4 KP . . . . .	41
4.6.1	Application to Full 32-round GOST . . . . .	42
<b>5</b>	<b>Cryptanalysis of Simon</b>	<b>43</b>
5.1	Introduction . . . . .	43
5.2	General Description of SIMON . . . . .	46
5.3	Algebraic Cryptanalysis of SIMON . . . . .	48
5.4	Algebraic Attacks experiments and results . . . . .	50
5.4.1	Experiments with 2 P/C pairs . . . . .	51
5.4.2	Experiments with more P/C pairs . . . . .	52
5.4.3	Algebraic Attacks Using ElimLin Algorithm . . . . .	53
<b>6</b>	<b>Introduction to Elliptic Curves</b>	<b>54</b>
6.1	Mathematical Foundations . . . . .	54
6.1.1	Finite Groups . . . . .	54
6.1.2	Finite Fields . . . . .	54
6.1.3	Cyclic Groups . . . . .	55
6.2	Elliptic Curves . . . . .	56
6.2.1	Elliptic Curves Over $\mathbb{F}_p$ . . . . .	57
6.2.2	Binary Elliptic Curves . . . . .	58
6.3	Point Arithmetic . . . . .	59
6.3.1	Point Addition . . . . .	59
6.3.2	Point Doubling . . . . .	59
6.3.3	Explicit Formulas . . . . .	61
6.3.4	Point Multiplication and ECDLP . . . . .	62

6.4	Open Research Questions . . . . .	63
6.4.1	Summation Polynomials . . . . .	63
6.4.2	Semaev Cipher . . . . .	65
6.4.3	Curves with Special Shortcuts . . . . .	67
6.5	Elliptic Curve Cryptography . . . . .	68
6.5.1	Domain Parameters . . . . .	69
6.5.2	Key Pair Generation . . . . .	69
6.5.3	Elliptic Curve Digital Signature Algorithm . . . . .	70
<b>7</b>	<b>Bitcoin and Brain Wallet Attacks</b>	<b>72</b>
7.1	Bitcoin Elliptic Curve . . . . .	73
7.2	Brain Wallet . . . . .	74
7.3	Bitcoin Elliptic Curve Implementation and Benchmarking . . . . .	76
7.3.1	Related Work . . . . .	76
7.3.2	Special Designed Point Multiplication Method For Attack . . . . .	77
7.3.3	Point Representation . . . . .	80
7.4	Experiment Results . . . . .	85
7.4.1	Network Stress Test . . . . .	85
7.4.2	Disclosure of results . . . . .	86
<b>8</b>	<b>General Conclusions</b>	<b>88</b>
	<b>Appendices</b>	<b>89</b>
<b>A</b>	<b>An Appendix About Stuff</b>	<b>89</b>
<b>B</b>	<b>Another Appendix About Things</b>	<b>90</b>
<b>C</b>	<b>Colophon</b>	<b>91</b>
	<b>Bibliography</b>	<b>92</b>

# List of Figures

3.1	Block cipher topology: attackers are easy to control or manipulate the inputs and outputs but very hard to analyse variables in the middle	28
4.1	One Round of GOST And Connections in The Following Round . .	33
4.2	Our best set of 78 bits for UNSAT . . . . .	40
4.3	Our best set of 68 bits for SAT . . . . .	41
5.1	The round function of SIMON . . . . .	47
5.2	Our three attack scenarios . . . . .	49
6.1	Example of elliptic curve over $\mathbb{F}_{2^4}$ . . . . .	59
6.2	Elliptic curve point addition . . . . .	60
6.3	Elliptic curve point doubling . . . . .	60
7.1	Brainwallet generated by password “password” . . . . .	75
7.2	password strength comparison between using password and passphrase . . . . .	76
7.3	Blockchain.info tag address page . . . . .	86
7.4	Example of tagged brainwallet address . . . . .	87



# List of Tables

5.1	Best results obtained by a SAT solver . . . . .	51
5.2	Best results obtained by a SAT solver . . . . .	52
5.3	Best results obtained by a SAT solver . . . . .	53
5.4	Best results obtained by a ElimLin Algorithm . . . . .	53
6.1	NIST's recommendation for key management . . . . .	62
7.1	Time cost for different window width $w$ , point addition method secp256k1 library [1] secp256k1_gej_add_ge . . . . .	80
7.2	Benchmarking openssl and MPIR library for field multiplication, square and modular inverse in affine coordinate . . . . .	81
7.3	Operation counts for point addition and doubling. A = affine, P = standard projective, J = Jacobian [2, 3] . . . . .	81
7.4	Field operation counts and benchmark results . . . . .	84
7.5	Time cost for different window width $w$ for EC key generation . . .	85

## Chapter 1

# Introductory Material

Some stuff about things.[?] Some more things.

Inline citation:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## Chapter 2

# Introduction to Cryptography

Cryptology is the science of hiding and recovering secret information. It mainly divided into two research areas: the area of cryptography and that of cryptanalysis. Traditionally, cryptography is the study and practise of techniques, in order 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 he 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. More formally, this secret information is called key (usually a short string of bits, which is needed to decrypt the ciphertext) while the transformations are called the encryption and decryption algorithms.

Cryptanalysis is a 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 his 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 briefly introduction about two cryptosystems, symmetric cryptography and asymmetric cryptography. We will practically discuss block ciphers in symmetric cryptography, which is widely used and well implemented in real world.

## 2.1 Symmetric and Asymmetric Cryptography

Cryptographic algorithms are classified based on how key material is used and managed. Normally they are classified as three groups: There are keyless algorithms which do not use any key and do not need to trust anyone; Another type of algorithms use shared key, which need to trust everyone that has the key; And the third type are private-public key algorithms, which private key is only holding by one person [4].

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

A cryptosystem often contains an encryption function  $\epsilon$ , which takes a plaintext  $p$  and a secret key  $k$  which is random bits and outputs a ciphertext  $c = \epsilon_k(p)$ , and the decryption function  $D$  (inverse of  $\epsilon$ ), 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 the adversaries can get ciphertext, they can not gain any information regarding the secret key or the plaintext.

In a cryptosystem, if  $k = k'$  which means the same secret key is used in both encryption and decryption, then the cryptosystem is called symmetric cryptosystem.

**Definition 1** (Symmetric Cryptosystem). *A symmetric encryption scheme is a five-tuple  $(P, C, K, \epsilon, D)$ , where  $P$  is the finite set of plaintext,  $C$  is the finite set of ciphertext and  $K$  is the key space such that  $\forall k \in K$  there is an efficiently computable*

encryption function  $E_k \in \mathcal{E}$  which respect to random bits  $k$ ,

$$E_k : P \rightarrow C$$

and a corresponding efficiently computable decryption function  $D_k \in \mathcal{D}$ ,

$$D_k : C \rightarrow P$$

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 so that decryption of a given ciphertext  $c$  results in plaintext  $p$ , then the cryptosystem is called asymmetric cryptosystem.

**Definition 2** (Asymmetric Cryptosystem). *An Asymmetric encryption scheme is a five-tuple  $(P, C, K, \mathcal{E}, \mathcal{D})$  where  $P$  is the finite set of plaintexts,  $C$  is the finite set of ciphertexts and  $K$  is the key space, an efficiently computable key generation algorithm  $\text{keyGen}()$  randomly generate a pair of public key  $p_k$  and secret key  $s_k$ , such that  $\forall p_k \in K$  there is an efficiently computable encryption function*

$$E_k \in \mathcal{E}$$

with respect to  $p_k$ ,  $E_{p_k} : P \rightarrow C$  and an  $s_k \in K$  corresponding to an efficiently computable decryption function  $D_{s_k} \in \mathcal{D}$ ,

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

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 a full control over the system. Asymmetric cryptosystems solve

this problem by using different but corresponding keys for encryption and decryption. However, in modern cryptography, a network requires a great number of keys which must be distributed securely.

Researchers start to solve the problem by combining both types of cryptosystem, called *hybird* encryption schema. This cryptosystem combines the convenience of asymmetric with the efficiency of a symmetric cryptosystem [5]. In Hybrid encryption, symmetric cryptosystem is used for encryption while the secret key shared using a protocol based on public-key cryptography. There are two main components of hybird encryption schema, Key Encapsulation Mechanism (KEM) and Data Encapsulation Mechanism (DEM). Key feature is that the two parts are independent of one another. The framework was first formalised by Cramer and Shoup in 2003 and we refer reader to [6] for more details.

## 2.2 Block Ciphers

There are two kinds of symmetric cryptosystem, stream ciphers and block ciphers. In stream ciphers a long sequence of bits is generated from a short string of key bits, and is then added bitwise modulo 2 to the plaintext to produce the ciphertext. 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, means that the same inputs result to the same outputs. Block ciphers are considered as the hardest part of cryptography. Normally block ciphers are designed to be used for 50 years while asymmetric cryptosystems are normally designed for 10 years.

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

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

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

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  $\{0, 1\}^n \rightarrow \{0, 1\}^n$  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 give  $2^k$  distinct permutations on n-bits. A good design of a block cipher aims to choose the  $2^k$  permutations uniformly at random<sup>1</sup> from the set of all  $(2^n!)$  permutations.

Block 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 kinds of substitution ciphers: simple substitution, polyalphabetic substitution, homophonic substitution and polygram substitution [7]. Here we only discuss the first two kinds:

**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  $P = p_0, \dots, p_{n-1}$  be an n-character plaintext,  $C = c_0, \dots, c_{n-1}$  be a ciphertext,  $\forall i : 0 \leq i < n$

$$E : P \rightarrow C$$

$$c_i = f(p_i)$$

Around 50 B.C., Julius Caesar wrote to Marcus Cicero, using a cipher which encrypted messages by shifting every letter in the plaintext three positions to the right in the alphabet. This cipher is based *shifted alphabets*. For Caesar cipher, the secret key  $k$  is  $+3 \pmod{26}$ . 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  $K = k_0, k_1, \dots, k_{j-1}$ , which defines j distinct encryption functions  $E_{k_0}, E_{k_1}, \dots, E_{k_{j-1}}$ . More formally,

---

<sup>1</sup>or it is impossible to see if it was otherwise

$$\forall i : 0 \leq i < n$$

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

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

The Vigenère cipher [8], first published in 1586 which uses polyalphabetic substitution 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  $\forall i : 0 \leq i < n$

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

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

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

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

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

The Vigenère and in general substitution ciphers can be broken when enough ciphertext is available to the cryptanalyst by the index of coincidence, Kasiski's method, etc. [9, 8, 10]. Transposition ciphers can be broken by using the frequency distributions for bigrams, trigrams and N-grams [9, 8, 10]. This knowledge about natural language is also very useful to our later work in password cracking.

### 2.2.3 Product Ciphers

To produce more stronger ciphers than the ones we have seen so far is to combine substitution and transposition ciphers. These ciphers are called *product ciphers*.



Many product ciphers have been developed, including Rotor machines [9]. Most block ciphers which still been using today are product ciphers. An iterated cipher is one of the 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*

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

*where  $C_0$  is the plaintext,  $K_i$  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 which makes the encryption and decryption process identical. One method was created by German physicist and cryptographer Horst Feistel, who was the pioneer in this area while working for American IBM. Feistel together with Don Coppersmith introduced the concept of Feistel networks while working in IBMs Lucifer cipher in 1973 [11]. Their work gained the respect of the U.S Federal Government who adopted it to Data Encryption standard (DES), which is based on Lucifer project with some changes done by the NSA [12].

**Definition 4.** *(Feistel Network) A Feistel cipher is an iterated cipher which 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  $i$ -th subkey derived from the secret key  $K$  and  $f$  the one-round function which takes a subkey and a  $t$ -block to 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 Feistel network is the same as the encryption process. Only require a reversal of the key schedule. The final output is given by the following:

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

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

As the encryption and decryption process are identical, the software and hardware implementation of Feistel ciphers is much easier.

## Chapter 3

# Cryptanalysis of Block Ciphers

The history of cryptanalysis is long and at least as fascinating as the history of cryptography. As an example, in 1917 an article in “Scientific American” claimed the Vigenère cipher is “impossible of translation” [7]. Today, most cryptography classes at university use it as an exercise to illustrate that this claim is not true. When discussing the security of a cryptosystem, one needs to define a model works in the real world which we will use the model of Shannon [13].

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 decrypt  $C$  to  $P$  using  $K$ . The attacker has access to the insecure channel and can intercept 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  $K$ . To evaluate the security we assume:

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

Also we will assume that the attacker knows all details about the cryptographic algorithm used by the sender and receiver, except the secret key. This assumption is known as Kerckhofs’s Assumption [10].

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

As an extension of the Assumption 2, we discuss the following two cases.

**Assumption 3.** *The enemy cryptanalyst knows all the details of the enciphering process and deciphering process except the value of the key and the S-Boxes (Substitution-box) which is a basic component of symmetric key algorithms which performs substitution.*

Under this assumption, the S-Boxes are like the master key or a high level key, which is kept as security. Similar to the rotors using in German Enigma during World War II. The attacker has to first recover the S-boxes through silicon reverse engineering or try for different sets of known S-boxes, then perform normal cryptanalysis as Assumption 2.

**Assumption 4.** *The enemy cryptanalyst knows all the details of the enciphering process and deciphering process except that the cipher has been tweaked, for example 90% of the cipher is what we know*

attacker has again two steps under this assumption. First recover the cipher through reverse engineering and try different sets of known S-boxes, then perform normal cryptanalysis as Assumption 2.

### 3.1 Classification of Attacks

Based on these assumptions, we classify the possible attacks an attacker can do [14]

1. **Ciphertext only attack:** The attacker processes a set of intercepted ciphertexts
2. **Known plaintext attack:** The attacker obtains  $P_1, P_2, \dots, P_s$  a set of  $s$  plaintexts and the corresponding ciphertexts  $C_1, C_2, \dots, C_s$
3. **Chosen plaintext attack:** the attacker chooses a priori set of  $s$  plaintexts  $P_1, P_2, \dots, P_s$  and obtains in some way the corresponding ciphertexts  $C_1, C_2, \dots, C_s$
4. **Adaptively chosen plaintext attack:** The attacker chooses a set of plaintext  $P_1, P_2, \dots, P_s$  interactively as he obtains the corresponding ciphertext

$C_1, C_2, \dots, C_s$ . That is the attacker chooses  $P_1$ , obtains  $C_1$ , **then** chooses  $P_2$  etc.

5. **Chosen ciphertext attacks:** For symmetric ciphers these are similar to those of chosen plaintext attack and adaptively chosen plaintext attack

The chosen text attacks are the most powerful attacks. However, they are also unrealistic in many applications. If there exist redundancy in plaintext space, it will be very hard for an attacker to find an encrypted non-meaningful plaintexts send by the sender, and similarly hard to get ciphertexts decrypted. But if a system is secure against an adaptively chosen text attack then it is also secure against all other attacks.

Modern cryptanalysis on block ciphers has been very focused on finding the secret key  $K$ . However, there are other serious attacks for public key cryptography which do not find the secret key. [15] classified the types of breaks in his paper as follows:

1. **Total break:** An attacker finds the secret key  $K$
2. **Global deduction:** An attacker finds an algorithm  $A$ , 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 he did not obtain from the legitimate sender
4. **Information deduction:** An attacker gains some information about the key, plaintexts or ciphertexts, which he did not get directly from sender and which he did not have before the attack.

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

**Data Requirement** Attacks can also be characterised by resources they require. Those resources include time complexity, memory usage and data requirements.

The first two types of resources are very obvious, however it is worth to point out data requirement is also a key component that makes an attack can work in practices. Full plaintext and ciphertext pairs are not easy to obtain in real world. Some of attacks works only if all possible plaintext and ciphertext pairs for a single key is known, and some of the attacks can work with multiple key scenarios.

## 3.2 Brute-force Attack

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. This attack requires the computation of about  $2^k$  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. For modern block ciphers, brute force is always possible in theory but computationally infeasible in practise.

## 3.3 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack on block ciphers. It was popularised by Matsui in 1993 [16]. A preliminary version of the attack on FEAL was described in 1992 [17]. Although the published attack on DES requires  $2^{43}$  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 hold 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 probability  $p_L \neq \frac{1}{2}$  over all keys [16], such that  $\epsilon = |p_L - \frac{1}{2}|$ , call the bias, is maximal.

$$(P \cdot \alpha) \oplus (C \cdot \beta) = (K \cdot \gamma) \quad (3.1)$$

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[16].

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, the above linear attack is not very efficient, as it only finds one bit of information about the key. In [16], 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 repeat the attack for all values of the relevant key bits in those two rounds by using the following approximation equation (equation 3.2).

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

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  $r$ th rounds. We refer reader to [16] for more details.

Kaliski and Robshaw showed an improved linear attack using multiple linear approximations in [18]. In [19] Knudsen and Robshaw introduced a linear attack using non-linear approximations in the outer rounds of a block cipher. Both of these are not been able to show an significant improvement compared to Matsui’s linear attack. The attacks seem best suited for ciphers with large S-boxes, such as LOKI [20] [19]. In 2004, new attacks were introduced to attack Feistel schemes in [21] which have a small improvements than 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 analysed by Biham and Shamir and applied to DES algorithm in the early 1990s [22]. However, Coppersmith, a member of the IBM team which designed DES [23, 24, 25], has 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 specific original design criteria of DES can be found in [25]. 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. [23, 24]

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 [26, 27]. Recent research work also showed a great success using differential cryptanalysis to break Russian standard GOST [28, 29] and NSA lightweight cipher SIMON and SPECK [30, 31]

The data requirements for differential cryptanalysis works with multiple key scenario, where linear cryptanalysis requires data on single key. This is the main reason why differential cryptanalysis is considered more stronger than linear cryptanalysis [32].

The main task of Differential cryptanalysis is to study the propagation of input differences from round to round inside the encryption system, and find specific differences which propagate with high probability. Such pairs of input-output 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.

**Definition 5.** *a difference between two bit strings,  $X$  and  $X'$  of equal length as*

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



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 where  $(X)^{-1}$  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 in many ciphers use XOR for the key application in round function, the operator in definition 5 usually is 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 aim to find one particular ciphertext difference which is especially frequent. In this way, the cipher can be distinguished from random.

### Truncated Differentials

Truncated Differential Cryptanalysis is a generalization of differential cryptanalysis developed by Lars Knudsen [33]. Unlike differential cryptanalysis which studies the propagation of the full difference between two plaintexts, truncated differential cryptanalysis consider differences that are partially determined, as in some ciphers, it is possible and advantageous to predict the values of parts of the differences after each round. This technique have been successfully applied to many block ciphers such as Russian standard GOST [34, 35, 36]. Truncated differential is defined as follow: [33].

**Definition 6.** *A differential that predicts only parts of an  $n$ -bit value is called a truncated differential. More formally, let  $(a,b)$  be an  $i$ -round differential. If  $a'$  is a subsequence of  $a$  and  $b'$ , then  $(a',b')$  is called an  $i$ -round truncated differential.*

A truncated differential can be considered as a collection of differentials. For example, let  $(a',b)$  be the truncated differential of an  $n$ -bit block cipher, where  $a'$  specifies the least  $n' < n$  significant bits of the plaintext difference and  $b$  specifies the ciphertext difference of length  $n$ . This differential is a collection of all  $2^{n-n'}$  differentials  $(a,b)$ , where  $a$  is any value that truncated to the  $n'$  least significant bits is  $a'$ .

### 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 which are needed in a linear and differential cryptanalytic attack. However, in most cases, only a few plaintext-ciphertext pairs are available for cryptanalysis, linear or differential attacks are not expected to succeed. Thus, a new method needs to be invented which will be able to recover the secret key with only a few information is available.

Claude Shannon, has 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” [37].*

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  $GF(2)$  ) or logical constraints in terms of the secret key  $K$ , the plaintext  $P$  and the ciphertext  $C$ .

$$f_1(K, P, C) = 0, \dots, f_r(K, P, C) = 0 \iff E(K, P) = C$$

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 to the system all known plaintext/ciphertext pairs. The more the pairs the more the equations we obtain involving the key bits. For many known attack this can make the problem easier to solve.

Generally, each module of a block cipher can be described with a set of algebraic equations. Put these algebraic equation together, we can get a large precise system describing the whole cipher. The main idea of 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  $GF(2)$ . Each variable of such equation system represents a state-bit of the encryption algorithm. Then the variables which representing state-bits from the initial round are set according to values of the plaintext, and similarly variables which representing state-bits from the last round are set according to values to the ciphertext. Therefore, the security of a block cipher depends on whether there exist an efficient algorithm to solve such large and sparse multivariate polynomial equations. If we can solve the equation system faster than exhaustive key search attack, the cipher will be broken (in academic sense).

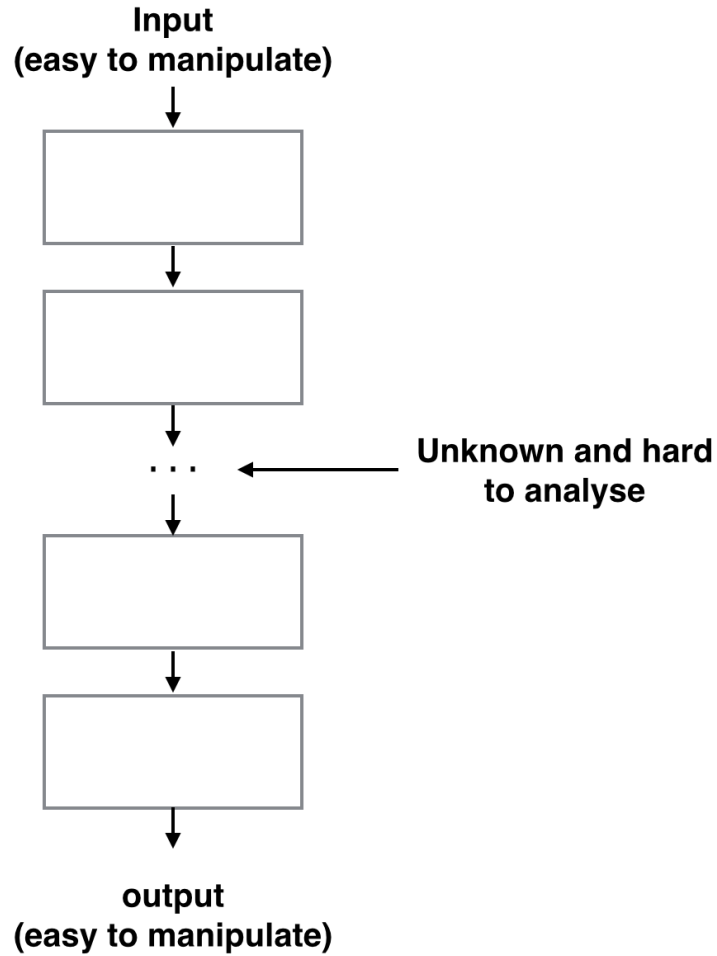
In addition, some high profile cryptanalysis problems in public key cryptography can be written in such a form which contains a “block cipher topology”<sup>1</sup>. Such equations have similar structure as a block cipher, where the beginning inputs and final outputs variables are easy to get and the middle part of the equation system are very hard to analyse (see figure 3.1). One example is Semaev’s summation polynomial equations for elliptic curve [41, 40] see section 6.4.2.

### 3.5.1 Algebraic Attacks Solving Stage

Solving a random system of multivariate non-linear boolean equations is an NP-hard problem [42]. As many cryptographic primitives can be described by a sparse multivariate non-linear system of equations over  $GF(2)$  or any other algebraic systems. Several techniques were develop in order to tackle the problem of solving such equations. A classic approach is to use techniques from algebraic geometry and especially Gröbner bases algorithms to solve the system of equations [43]. But most of the times they do not lead to solutions in practise due to the extremely high memory requirements. Then, some heuristic techniques were developed called *lin-*

---

<sup>1</sup>which is how we / Dr. Courtois classify the equations proposed by Semaev in 2015 [38, 39, 40]



**Figure 3.1:** Block cipher topology: attackers are easy to control or manipulate the inputs and outputs but very hard to analyse variables in the middle

*earisation* [44], where all the non-linear terms are replaced by an independent linear variable and the resulting linear system can be solved using Gaussian elimination [45]. However, it requires that there are enough linearly independent equations and the initial system is highly over-defined and sparse. Then, XL algorithm [44, 38] was developed to make the system over-defined, by addition of new equations to the current system. The proposal of XL made it possible to solve the multi-order nonlinear equations within polynomial time, which accelerated the development of the algebraic attacks. Then researchers focus mainly on the fast solutions to the algebraic equation systems.

In the past few years, researchers try to solve the problem by using tools and software, such as SAT solvers. Nicolas Courtois is the pioneer of bringing SAT solvers into action in the area of symmetric cryptanalysis. His paper [46, 47] described how to covert such equations to a format which can be solved automatically by SAT solvers. The advantage of such technique is that SAT solvers can perform reasonably well and do not require a lot of memory as in case of Gröbner basis-based techniques [48]. The only disadvantage is the unpredictability of its complexity. The first algebraic attack on reduce round block cipher DES was done by Courtois and Bard in [49]. Later in 2008, first algebraic attack on full block cipher KeeLoq [50]. In the past 10 years, SAT solvers were used for attacks on block ciphers such as GOST[51, 52] and Chinese block cipher SMS4 [53], stream ciphers such as Crypto-1, HiTag2 and Bivium [54, 55], and also MiFare Classic smart cards [56].

Another method, is to use ElimLin algorithm [57]. ElimLin stands for **E**liminate **L**inear and it is a simple algorithm for solving polynomial systems of multivariate equations over small finite fields and was initially proposed as a single tool by Courtois to attack DES and CTC/CTC2 ciphers [49]. It is also known as “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, as follows:

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

Given an initial multivariate system of equations over  $S^0$  in  $\mathbb{F}_2[x_1, x_2, \dots, x_n]$ , then the ElimLin is formally described as below.

This method is iterated until no linear equation is obtained in the linear span of the system. Intuitively, ElimLin seems to work better in cases where there is

**Algorithm 1** ElimLin Algorithm

---

**Input:**  $S^0 = \{f_1, f_2, \dots, f_m\} \in \mathbb{F}_2[x_1, x_2, \dots, x_n]$ **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 monomialsSet  $S_{L'} \leftarrow$  Linear Equations from  $Gauss(S^T)$ Set  $S^T \leftarrow Gauss(S^T) \setminus S_{L'}$ Let  $l \in S_{L'}$ ,  $l$  non-trivial (if unsolvable then *terminate*)Let  $x_{i_k}$  a monomial in  $l$ Substitute  $x_{i_k}$  in  $S^T$  and  $S_{L'}$  using  $l$ Insert  $l$  in  $S_L$  $k \leftarrow k + 1$ 

---

low non-linearity since this implies the existence of more linear equations. MC is another notion of non-linearity [58, 59] and possibly such method may work sufficiently well in cryptographic primitives of low MC.

## Chapter 4

# Cryptanalysis of GOST

### 4.1 GOST encryption standard

The Russian encryption standard GOST 28147-89 is an important government standard [60]. Its large key size of 256 bits make GOST a plausible alternative for AES-256 and 3-key triple DES. Clearly GOST is 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. [61, 62].

The most complete current reference implementation of GOST in OpenSSL library contains eight standard sets of S-boxes [62]. The attacks we consider in this Chapter, work with a very similar complexity for any S-boxes.

#### 4.1.1 GOST And ISO Standardisation.

The cost of cryptography is still an important problem for the industry, for example only around 2010 Intel implemented an encryption algorithm in some of its CPUs, and not yet in all of its CPUs. 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, while AES-128 requires 3100 GE, see [63]. Thus it is not surprising that GOST became an Internet standard, it is part of many crypto libraries such as OpenSSL [62], and is increasingly popular also outside its country of origin [63]. Hard to think about a better algorithm for the industry with its ultra-low implementation cost and 20 years of cryptanalysis efforts behind it

[63]. 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 have been found [64, 65, 52, 28, 34].

### 4.1.2 Cryptanalysis of GOST

The turning point in the security of GOST was the discovery of the so called “Reflection” property [66]. Initially at Indocrypt 2008 only a weak-key attack with time complexity of  $2^{192}$  is proposed, with large proportion of  $2^{-32}$  of weak keys. Then in 2011 several attacks on regular GOST keys have been discovered, and more than half of these new attacks use this reflection property [64, 67], sometimes twice, three or four times [52]. Most these attacks can be described as attacks with a “Complexity Reduction” [65, 52] 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 cipher. In this paper we look precisely at questions pertaining to cryptanalysing 8 rounds GOST with 2,3,4 KP needed as a last step in numerous already known attacks on GOST [65, 52, 68].

Many attacks which do not use any reflections have also been proposed [52, 65, 67] and also differential attacks which do not fall into the “Complexity Reduction” category. The most recent advanced differential attack on GOST has time complexity of  $2^{178}$ , see [28, 34] which also the best single-key attack known.

## 4.2 The Internal Structure of GOST

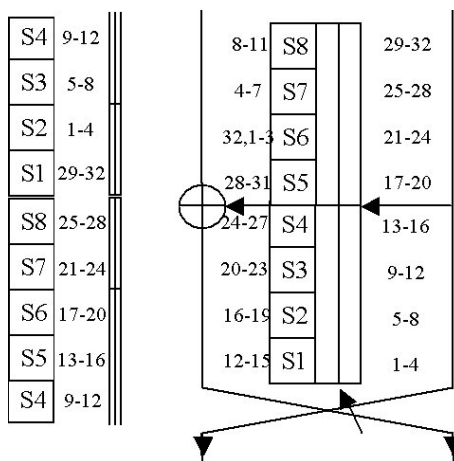
GOST is a block cipher with a simple Feistel structure, 64-bit block size, 256-bit keys and 32 rounds. Each round contains a key addition modulo  $2^{32}$ , a set of 8 bijective S-boxes on 4 bits, and a simple rotation by 11 positions.

GOST has 32 identical rounds such as the one described on Fig. 4.1 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 [65, 52, 64, 69, 28, 29, 34, 67]. In this paper however we only look at up to 8 rounds of GOST which have independent 32-bit keys which don’t repeat, therefore



we can ignore the GOST key scheduling totally.

We number the inputs of the S-box  $S_i$  for  $i = 1, 2, \dots, 8$  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  $S_6$  are 20, 21, 22, 23 and the outputs are 32, 1, 2, 3.



**Figure 4.1:** One Round of GOST And Connections in The Following Round

On our picture Fig. 4.1 the  $\boxplus$  denotes the addition modulo  $2^{32}$ . At the left margin on Fig. 4.1 we also show S-box numbers in the next round, which is very helpful, 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.

### 4.3 Algebraic Cryptanalysis with Complexity Reduction

Following [65, 52] the work of cryptanalyst who wants to cryptanalyse GOST can be split into two independent tasks. First task is how to achieve a software algebraic attack on a reduced-round version as discussed in the previous section. The second question is if and **how** the complexity major variants of full GOST with 32 rounds can ever be **reduced** to a problem of breaking a cipher with much less rounds. In fact only in the recent 5 years it became possible to design and implement an

appropriate last step for many such attacks which is a software algebraic attack, a Meet-In-the-Middle (MITM) attack, or other low-data complexity attack. This last step is the main focus in this paper.

### 4.3.1 Reductions and Black-Box Reductions

The main idea is as follows [65, 52]: In order to reduce the attack complexity, we exploit the self-similarity of the cipher (due for example to a weak key schedule) and add some well-chosen assumptions which produce interesting and sometimes quite non-trivial consequences due to the high-level structural properties of the cipher, which makes cryptanalysis problems smaller, simpler and easier to solve. In this process we need to minimise the costs (in terms of probability that our assumptions hold) and to maximise the benefits (in terms of the number and the complexity of interesting relations which hold under these assumptions).

This process is called **Algebraic Complexity Reduction**, see [65, 52]. In most cases what we get is to compute (guess or determine) many internal values inside one or several decryptions, and literally break the cipher apart into smaller pieces. In particular we have **Black-Box Algebraic Complexity Reductions** where we obtain real black-box reductions, to for example the same cipher with strictly less rounds (and less data) again at the cost of some well-chosen assumptions. Most but not all reductions we are aware of are real “black box” reductions, see [52] for a detailed discussion. The notion of Algebraic Complexity Reduction creates new important **optimisation problems** in symmetric cryptanalysis: which deals with the fundamental question of how we can reduce the complexity of a cipher in cryptanalysis to a simpler problem, with a limited quantity of data, and with greatly reduced complexity, and this in the best possible (optimal) way while many interesting and non-trivial solutions will exist. One example of a Black-Box Algebraic Complexity Reduction from  $2^{64}$  KP for 32 rounds of GOST, to 4 KP for 8 rounds of GOST, can be found in [65] and many more in [52].

Algebraic Complexity Reduction is a sort of umbrella paradigm which generalizes many already known fixed point, sliding, reflection and involution attacks. One is able to exploit similarities of individual sub-blocks and their inverses. We have

reflection attacks [66] but also have new attacks with double triple and quadruple reflection [52]. We are able to relax the conditions necessary in slide attacks [70] in nearly arbitrary ways. We have many new sort of self-similarity attacks, cf. [52]. though reflection attacks [66, 52] are related to fixed point attacks which are an important attack for ciphers with block size being smaller than the key size, cf. [50, 71].

### 4.3.2 Optimizing The Reductions: Amplification

Reductions can be compared in terms of the number of pairs obtained, the resulting reduced number of rounds, success probability, and in terms of plaintext complexity, see [52, 65]. A key property of these reductions is the process of so called **Amplification** which was introduced in [72].

**Definition 7** (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 amplification is an important question in algebraic cryptanalysis which was previously discussed in [72]. We should note that there are some difficulties in defining this ratio formally:

1. We claim that we need specific definitions for each individual cipher and for each specific attack method. For example  $Y$  can be the total number of linear equations obtained with the ElimLin algorithm [72, 49, 57] after adding a well-chosen set of  $X$  linear equations on the internal bits inside the cipher.
2. Intuitively, the higher, this amplification coefficient  $A$  is, while  $X$  and  $Z$  remain below a certain threshold, the stronger and more surprising is the attack obtained.

3. With higher values of  $X$ , the amplification can also be higher, however the attacker must limit the size of  $X$  for the whole the attack to remain fast enough overall.
4. It is very difficult to know if an attack with given parameters may exist.

**Examples.** In the most basic slide attack based on periodicity in the key scheduling [70] the amplification is unlimited. From one “slid pair”, we can obtain another “slid pair”, then another, etc.. However in more advanced sliding attacks which operate with imperfect periodicity, the amplification can be limited, or occur only after the attacker guesses a few key bits, see [50, 71].

Another example where the amplification is exceptionally high is the Weak Key Family 3 in [52]. In this case the amplification is very very high:  $X = 1$  and  $Y = 256$  while  $Z = 64$ , see [52]. This can only be obtained for some weak keys in GOST with  $Z = 64$ . In spite of the additional of factor of  $2^{64}$  implied by this value of  $Z$  the amplification is very high and overall it leads to very efficient attacks on certain GOST keys, see [68, 52]. The work described in this chapter is mostly motivated by the question how one can identify the suitable sets of key bits for such attacks.

### 4.3.3 Working at The Low Level

The amplification is easy to define as in many initial steps in the cryptanalysis of GOST [52], we deal with black boxes. It is harder but possible to define at the low level, when we look at a complete functional description of a cipher. Here the question is what is the best possible software attack with tools such as the ElimLin algorithm [72, 49, 57] SAT solvers [73, 47], Gröbner bases [48] and other [74]. 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 [72, 49, 57]. 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 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.

Our work focus more specifically on cryptographic attacks with SAT solvers, and on GOST, which is a nice example of a weak government standard cipher with relatively poor diffusion. There are two main approaches in SAT cryptanalysis or two main algorithms 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 than 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 method: where the final complexity will be a sum of two terms none of which can be neglected. We will see a very nice example how a combined attack can be better than any of SAT and UNSAT methods in isolation in Section 4.6.

#### 4.3.4 Applications of UNSAT/SAT Immunities

**Cryptanalysis.** The main idea is that these two numbers will allow to evaluate the security of the cipher against cryptanalytic attacks with a SAT solver. Upper bounds we obtain do translate, more or less, as we will see, into concrete attacks

with 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 than designing a cipher which is provably resistant to linear and differential cryptanalysis. Interestingly, Schneier once claimed that “Against differential and linear cryptanalysis, GOST is probably stronger than DES” [61]. 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 [49] and GOST [73]. 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 KP 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 KP and 56-bit keys? It is very hard to say.

## 4.4 Application to DES

In this section we give some basic results for DES obtained by the methods of [49] with however a better SAT solver CryptoMiniSat 2.92 [75]. Unhappily 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 and for now we just report upper bounds obtained when choosing the key bits at random, and letting the SAT solver to do the job.

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

**Fact 2.** *The SAT Immunity is at most 34 for 8 rounds of DES and 1 KP.*

These two results were obtained with the gate-efficient encoding described in [49], and with CryptoMiniSat 2.92 [75]. We found no attack on 8 rounds of DES

and 1 KP which would be faster than brute force. For ultra low-data complexity attacks, 8 rounds of DES seem already secure or secure enough.

## 4.5 Application to GOST

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

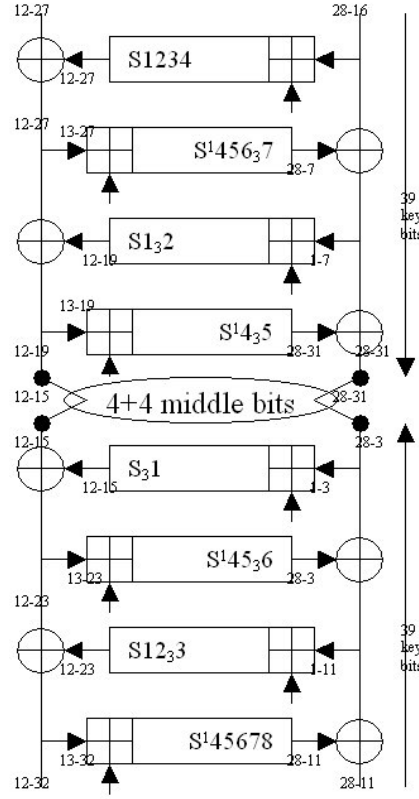
For a long time we thought that the Contradiction Immunity of 8 rounds of GOST was about 128. The reason for that can be found on Fig. 4 in [34]: it is possible to see that GOST splits very neatly into two nearly independent ciphers with 128-bit key each, which are only loosely connected. With this idea it is easy to understand why a software/algebraic attack on 8 rounds of GOST with complexity of  $2^{120}$  seems plausible and natural. However recently we found that it is much lower than 128, much closer to 64. This was the main motivation for writing this paper.

**Notation, cf. Fig. 4.2:** We denote by  $S^1_3$  just 1 higher ranking bit at S-box 1 in a given round. Similarly we denote by  $S_3$  the 3 lower ranking bits of S3.

**Fact 3.** *The Contradiction Immunity for 8 rounds of GOST is at most 78.*

*Justification:* We have constructed and tried many different sets aiming at a contradiction in the middle. Our best set is as follows (cf. Fig. 4.2): 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 [75] with 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.

**Remark.** In fact we come remarkably close to 68 bits. If we consider the set of 68 bits later shown on Fig 4.3 and also used in [68], we achieve about 39 % UNSAT with CryptoMiniSat 2.92. It is remarkably close but it does not achieve 50 % required. This may seem strange, but in order to achieve 50 %, many more key bits are needed, cf. Fact 3 above. This is because in GOST it makes a lot of sense to guess key bits for full S-boxes, and S-boxes active in one round call for other S-boxes being also active.



**Figure 4.2:** Our best set of 78 bits for UNSAT

### 4.5.1 SAT Immunity of GOST

Unhappily, 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 Fig. 4.2 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:

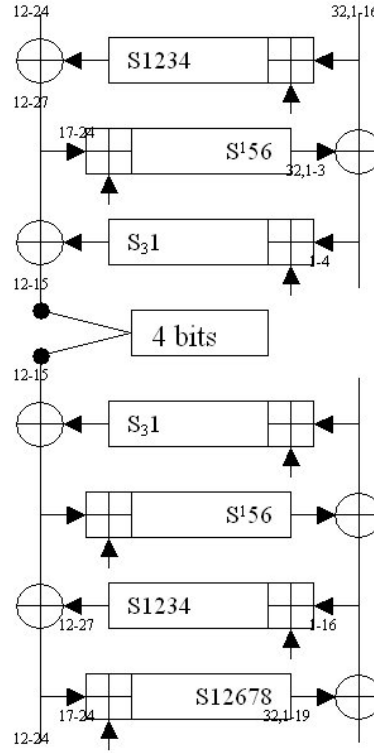
**Fact 4.** *The SAT Immunity for 8 rounds of GOST and 4 KP is at most 68.*

*Justification:* We use the following set of bits depicted on Fig 4.3 0-15,51-55,64-66,128-130,179-183,192-207,224-231,244-255 which is also used in [68]. All the remaining 256-68 bits can be determined in time of about 400 seconds using GOST encodings described in [?] and with CryptoMiniSat 2.92 [75].

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.

**Further Improvement.** In order to improve this attack, the interesting question is if we can obtain contradictions when one of the  $2^{68}$  cases is incorrect, and





**Figure 4.3:** Our best set of 68 bits for SAT

for what proportion of cases, and how long would it take. In other terms, we would like our set of 68 bits or possibly a smaller subset, to be good at UNSAT, which is the case as we have previously seen. In our attack with 4 KP we want to find a contradiction for all the 4 pairs simultaneously. This will be easier than contradiction with 1 KP we studied previously. This leads to the following improved attack which mixes the SAT and UNSAT strategies and which is also described in [68].

## 4.6 A Mixed Attack with 4 KP

**Fact 5.** *Given 4 KP for 8 rounds of GOST the full 256-bit key can be found in time of about  $2^{94}$  GOST computations and negligible memory.*

*Justification:* As in [68] we proceed as follows.

1. We use our set of 68 bits as on Fig 4.3.
2. We run the software  $2^{68}$  times for all possible assignments of the 68 bits.

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.

#### 4.6.1 Application to Full 32-round GOST

Following [68, 52], this 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, then one can find one valid 256-bit key, in time of about  $2^{94}$  GOST encryptions for one (weaker) key. This can be converted into attacks on full GOST not in just one way, on the contrary there are tens of ways of doing it which is outside of the scope of this paper, we refer to [65, 68, 52].

## Chapter 5

# Cryptanalysis of Simon

### 5.1 Introduction

Nowadays lightweight cryptography is rapidly evolving and becoming more and more important due to the increasing demand from mobile phones and 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, efficient software and hardware implementation and very low overall cost with respect to several meaningful metrics (power consumption, energy consumption, size of the circuit [73, 76, 77]).

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 the cryptographic literature we find lots of such lightweight cryptographic primitives such as KATAN [78], KLEIN [79], ICEBERG [80], HIGHT [81], LED [82], mCrypton [83], PRESENT [84], Piccolo [85] and many others.

In July 2013, a team from the NSA has proposed two new families of particularly lightweight block ciphers, SIMON and SPECK, both coming in a variety of widths and key sizes [86]. We use a basic reference implementation of both ciphers

which can be found in [87], as well as the generator of algebraic equations to be used in algebraic attacks.

The designers 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 [88], 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 LUTs, 30 registers). Recent results about hardware implementation of block ciphers emphasize reducing the size and/or MC further possibly lead to optimal implementations [89, 73].

However in [86], no advanced analysis of the security of the ciphers was discussed. In the same paper [86], they briefly said that SIMON and SPECK were designed to provide security against traditional adversaries who can adaptively encrypt and decrypt large amount of data and some attention was given so that there are no related-key attacks. Except of 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 of cryptanalysis against differential, linear and rotational cryptanalysis [90, 91]. However, since SIMON is a cipher of exceptionally low MC and as a result of low non-linearity it is an ideal candidate for algebraic attacks and combinations of algebraic attacks with truncated differential cryptanalysis. Combining an algebraic attack with a truncated differential property is equivalent to adding extra linear equations to the system involving input and output bits. This can speed up

the solving stage when software solvers which exploit the linearity are used like ElimLin [57].

**Contribution and Outline:** In this part of the report, we will study SIMON-64/128 against algebraic attacks and combinations of algebraic attacks with truncated differential cryptanalysis. Due to its very simple mathematical structure, SIMON has a compact algebraic representation and because of the low non-linearity and MC, it seems an ideal for algebraic cryptanalysis and its variants. 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 [47]) or by ElimLin algorithm [58, 73, 59]. We are able to break up to 10 (/44) rounds of the cipher using a SAT solver and 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 results since one of the biggest difficulties in software algebraic cryptanalysis is to know how do experimental attacks scale up for larger numbers of rounds. This is possibly due to the very low non-linearity of the cipher and suggests that it worths studying a specific strategy for P-C pairs which have 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 more number of rounds. We discuss in details our results in Section 4.

In our attacks, we study the following three scenarios:

1. **RP/RC** (Random Plaintexts/Random Ciphertexts): We assume that random plaintext-ciphertext (P-C) pairs are available
2. **SP/RC** (Similar Plaintexts/Random Ciphertexts): We assume that random P-C pairs such that the plaintexts differ by very few bits are available (low Hamming distance)
3. **SP/SC** (Similar Plaintexts/Similar Ciphertexts): We assume that random P-

C pairs which follow a highly probably truncated differential property are available. In this setting we use just two such pair and then generate more random P-C pairs.

## 5.2 General Description of SIMON

SIMON is a family of lightweight block ciphers with the aim to have optimal hardware performance [86]. It follows the classical Feistel design paradigm, 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).

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

$$F : GF(2)^n \rightarrow GF(2)^n \quad (5.1)$$

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.

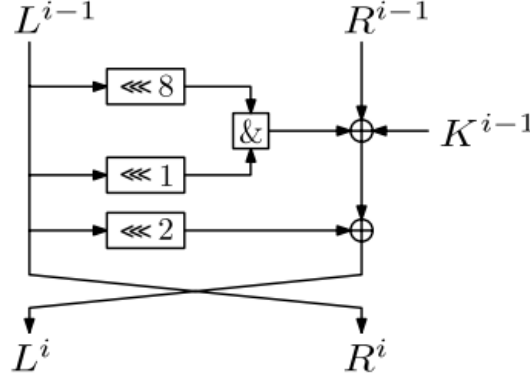
We denote the input to the  $i$ -th round by  $L^{i-1} || R^{i-1}$  and in each round the left word  $L^{i-1}$  is used as input to the round function  $F$  defined by,

$$F(L^{i-1}) = (L^{i-1} \lll 1) \wedge (L^{i-1} \lll 8) \oplus (L^{i-1} \lll 2) \quad (5.2)$$

Then, the next state  $L^i || R^i$  is computed as follows (cf. Fig. 5.1),

$$L^i = R^{i-1} \oplus F(L^{i-1}) \oplus K^{i-1} \quad (5.3)$$

$$R^i = L^{i-1} \quad (5.4)$$



**Figure 5.1:** The round function of SIMON

The output of the last round is the ciphertext.

The key schedule of SIMON is based on an LSFR-like procedure, where the  $nm$ -bits of the key are used to generate the keys  $K_0, K_1, \dots, K_{r-1}$  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  $K^0, K^1, \dots, K^{m-1}$  are initialized with the secret key, while the remaining are generated by the LSFR-like construction. For the variant of our interest, where  $m = 4$ , the remaining keys are generated in the following way:

$$Y = K^{i+1} \oplus (K^{i+3} \ggg 3) \quad (5.5)$$

$$K^{i+4} = K^i \oplus Y \oplus (Y \ggg 1) \oplus c \oplus (z_j)_i \quad (5.6)$$

The constant  $c = 0xff...fc$  is used for preventing slide attacks and attacks exploiting rotational symmetries [86]. 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  $z_0, \dots, z_4$ . These sequences are defined in [86] and for our variant we use  $z_3$ . In [87] there is a basic reference implementation of SIMON and SPECK ciphers and

a basic generator of equations that are used in algebraic attacks.

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 [88]. Additionally, encryption and decryption can be done using the same hardware.

### 5.3 Algebraic Cryptanalysis of SIMON

We evaluated the security of SIMON against algebraic attacks under the following three settings (cf. Fig. 5.2), where S=Similar and R=Random as explained in the introduction.

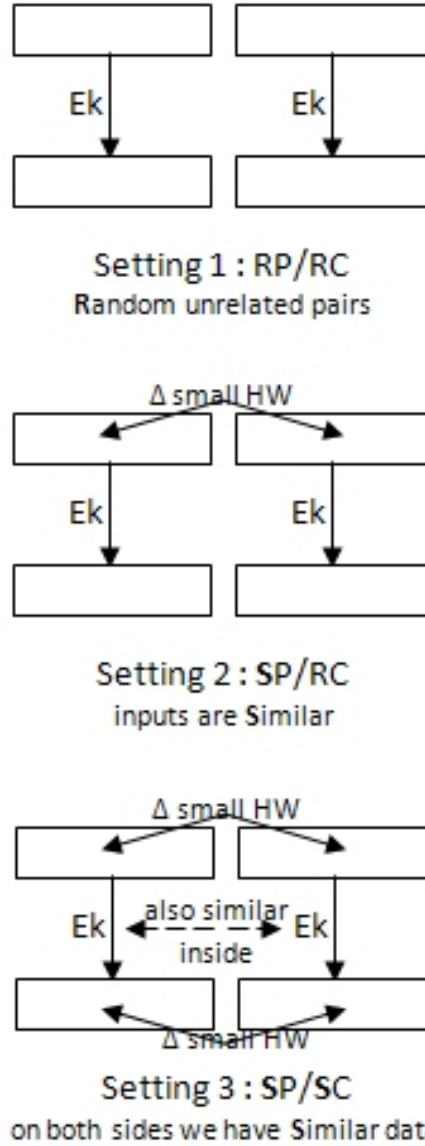
- **Setting 1:**(RP/RC) Random P-C pairs are available
- **Setting 2:**(SP/RC) Random P-C pairs are available with plaintexts of low Hamming distance
- **Setting 3:**(SP/SC) Random P-C pairs which satisfy a truncated differential property in the input and output of the reduced-version of the cipher we study. More random P-C pairs are generated and used.

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



**Figure 5.2:** Our three attack scenarios

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

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 more easier to solve. In this attack, we need assume 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  $\Delta = [0000022200000080]$  with 4 active bits found by our discovery method which is the following

$$Prob(\Delta \rightarrow \Delta) \simeq 2^{-20.51} \quad (5.7)$$

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.

## 5.4 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 implementation of Simon can be found at [87] which also include an equation generator which generate an algebraic equitation system for n round Simon encryption. Once the equation is generated, we use Nicolas Courtois' software [92] to either solve the system by ELimlin or convert to CNF file and then solved by a SAT solver. For reader to check and verify our results, here are the command line we used to get our experiments results:

- ELimlin: `xl0.exe /deg2 filePath`
- SAT Solver: `xl0.exe /deg2 /sat /bard /timeout600 /cryptominisat64296 filePath`

### 5.4.1 Experiments with 2 P/C pairs

The initial benchmark experiments was done with only 2 P/C pairs and solved by SAT solver using 8 round Truncated differential mask  $[0000022200000080] \Rightarrow [0000022200000080]$ . Table 5.1 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,

$$C_T = 2^k \times 2^{\log_2(T_{average} * N_{8REnc})}, \quad (5.8)$$

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 seconds.

**Table 5.1:** Best results obtained by a SAT solver

#(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

We start from setting 1 using random Plaintext and random Ciphertext pair (RP/RC), until we can not 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 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 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) is better than RP/RC. However, the results in table 5.1 show that SP/SC is not perform 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 can not 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.

### 5.4.2 Experiments with more P/C pairs

We start to using more P/C pairs for 8 rounds Simon and we show our results in talbe 5.2. Here we start to record some features of the converted CNF files. In table 5.2 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 [93])

**Table 5.2:** Best results obtained by a SAT solver

#(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

From Table 5.2 we can see that when using more P/C pairs, all the three settings get better results than using 2 P/C pairs in Table 5.1. The improvements for setting 2 and setting 3 are much more larger than setting 1. Setting 2 and setting 3 can break 8R Simon without guessing any key bits.

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:  $[0000002200000080] \rightarrow [002eff9a00022e4c]$

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 5.3

Moreover, assuming Setting 3 we can break 10 rounds by guessing 70 bits of the key initially with time complexity  $2^{98.79}$  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 MC, there exists

**Table 5.3:** Best results obtained by a SAT solver

#(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

a certain amount of pairs which depends on the MC which can be used to break any round.

### 5.4.3 Algebraic Attacks Using ElimLin Algorithm

Table 5.4 presents the results using the ElimLin algorithm for solving the underlying system of equations.

**Table 5.4:** 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 eliminate also variables in intermediate rounds and introduce more linear equations in the intermediate rounds. We conjecture that ElimLin is more powerful in case where a strong characteristic is found by our discovery method (intermediate difference states).

## Chapter 6

# Introduction to Elliptic Curves

## 6.1 Mathematical Foundations

### 6.1.1 Finite Groups

A Finite Group is a set  $G$  with a finite number of  $q$  elements, which has a binary operation  $*$  :  $G \times G \rightarrow G$  and satisfies the following properties [2]:

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 identity 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 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 identity 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  $a^{-1}$ .

### 6.1.2 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 [2]

1.  $(\mathbb{F}, +)$  is an abelian group with (additive) identity denoted by 0.
2.  $(\mathbb{F}, \cdot)$  is an abelian group with (multiplicative) identity 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 integers modulo  $p$ , consisting of the integers  $\{0, 1, 2, \dots, p-1\}$  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  $\mathbb{F} = 0, 1$ ) of degree at most  $m-1$ :

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \dots + a_2z^2 + a_1z^1 + a_0 : a_i \in \{0, 1\}\}$$

.

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 performed modulo 2. Multiplication of field elements is performed modulo the reduction polynomial  $f(z)$ .

### 6.1.3 Cyclic Groups

Let  $G$  be a finite group of order  $n$  with multiplication  $(\cdot)$  as binary operation, then  $G$  is called cyclic if there exist a  $g$  be an element of  $G$  such that  $\langle g \rangle = \{g^i : 0 \leq i \leq r-1\}$  is the subgroup of  $G$  generated by  $g$ , where  $r$  is the order of the element  $g$ , that is,  $r$  is the smallest positive integer for which  $g^r = 1$ . If  $r = n$  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  $g$ .

## 6.2 Elliptic Curves

An elliptic curve over a field  $K$  is defined by the following equation (also known as Weierstrass form)

$$y^3 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (6.1)$$

where:  $a_1, a_2, a_3, a_4, a_6 \in K$ , and the discriminant

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

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, we can assume that  $a_1 = a_2 = a_3 = 0$ , we have  $d_2 = 0$ ,  $d_4 = 2a_4$ ,  $d_6 = 4a_6$  and  $d_8 = -a_4^2$ . According to equation, the condition  $\Delta = -16(4a_4^3 + 27a_6^2) \neq 0$  can be simplified to  $4a_4^3 + 27a_6^2 \neq 0$ . The curve equation is normally written in the following form:

$$y^3 = x^3 + ax + b \quad (6.2)$$

where  $a = a_4$  and  $b = a_6$ .

For an elliptic curve over binary fields  $\mathbb{F}_2^m$ , without loss of generality, we can



assume that  $a_1 = 1, a_3 = a_4 = 0$ . Then we have  $d_2 = 1, d_4 = 0, d_6 = 0$  and  $d_8 = a_6$ <sup>1</sup>, and  $\Delta = -a_6 \neq 0$ . The curve equation is normally written as following:

$$y^2 + xy = x^3 + ax^2 + b \quad (6.3)$$

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

### 6.2.1 Elliptic Curves Over $\mathbb{F}_p$

The finite field  $\mathbb{F}_p$  uses the numbers from 0 to  $p - 1$ , and computations are done in modulo  $p$ . An Elliptic Curve over finite field  $\mathbb{F}_p$  where  $p$  is a large prime, can be formed by choosing the variables  $a$  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's typically defined in the short Weierstrass form:

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

where  $a, b \in F_p$  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 can be used to form a group. The elliptic curve contains all points  $P = (x, y)$  for  $x, y \in F_p$  that satisfy the elliptic curve equation and together with a special point  $O$  call the point at infinity<sup>2</sup>.

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 [94]. 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),  
 (14, 16), (3, 13), (2, 15), (4, 13), (0, 14),  $O$ ,

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

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

---

<sup>1</sup>In binary field anything multiply by 2 equals to 0.

<sup>2</sup>In code implementation,  $O$  is normally be represented as point (0,0), but not always, as (0,0) might be on the curve.

$$6^2 \bmod 19 = 4^3 + 4 + 6 \bmod 19$$

$$36 \bmod 19 = 74 \bmod 19$$

$$17 = 17$$

### 6.2.2 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 optimal normal basis[95]. 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,  $O$ .

To give 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:

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

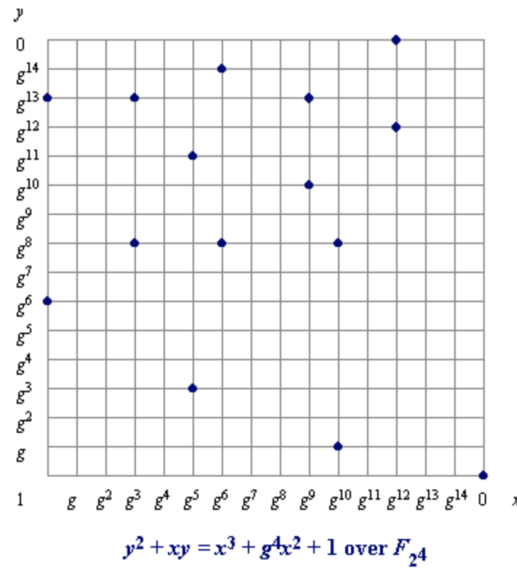
$$g^6 = (1100), g^7 = (1011), g^8 = (0101), g^9 = (1010), g^{10} = (0111), g^{11} = (1110)$$

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

Consider the elliptic curve  $y^2 + xy = x^3 + g^4x^2 + 1$ . The points on  $E$  are the following and shown in figure 6.1.

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

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

Figure 6.1: Example of elliptic curve over  $\mathbb{F}_{2^4}$ 

## 6.3 Point Arithmetic

### 6.3.1 Point Addition

Let  $P$  and  $Q$  be two distinct points on an elliptic curve, and the  $P$  is not  $-Q$ . To add the points  $P$  and  $Q$ , a line<sup>3</sup> is drawn through the two points. This line will have exactly one additional intersection points 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  $P + Q = R$ . Given an example of geometrical graph in figure 6.2

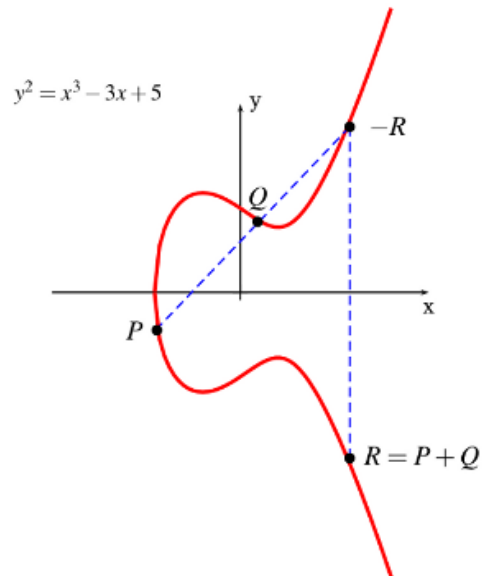
For point addition of  $P$  and  $-P$ , the line through  $P$  and  $-P$  is a vertical line which does not intersect the elliptic curve at a third point. Thus the point  $P$  and  $-P$  can not be added using the above method. It's for this reason that the elliptic curve group includes point  $O$ , the point at infinity and it is defined as  $P + (-P) = O$ .

### 6.3.2 Point Doubling

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

---

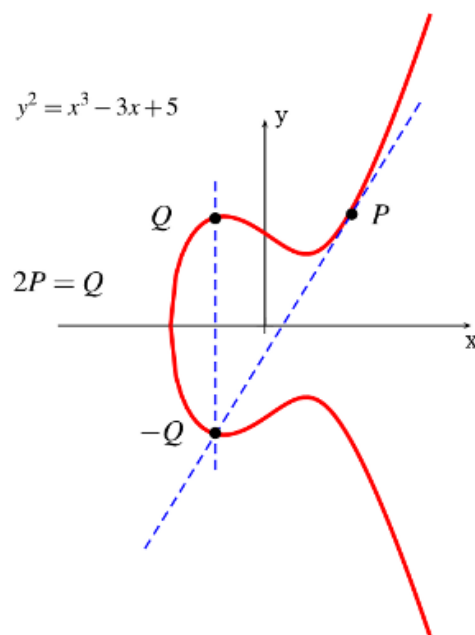
<sup>3</sup>line is set of points which satisfy the equation  $Ax + By + C = 0$  where  $A, B, C \in \mathbb{F}_p$



**Figure 6.2:** Elliptic curve point addition

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 and also shown in figure 6.3:

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



**Figure 6.3:** Elliptic curve point doubling

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

### 6.3.3 Explicit Formulas

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:

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

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

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

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

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

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

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

For elliptic curves over  $\mathbb{F}_{2^m}$ , the curve equation  $E$  is

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

where  $b \neq 0$ . Let  $P = (x_1, y_1) \in E$ ; then  $-P = (x_1, y_1 + x_1)$ . If  $Q = (x_2, y_2) \in E$  and  $Q \neq \pm P$ , then  $P + Q = (x_3, y_3)$  is given by:

$$x_3 = \left( \frac{y_1 + y_2}{x_1 + x_2} \right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a$$

$$y_3 = \left( \frac{y_1 + y_2}{x_1 + x_2} \right)(x_1 + x_3) + x_3 + y_1$$

and point doubling for  $P = Q$  is given by:

$$x_3 = x_1^2 + \frac{b}{x_1^2}$$

$$y_3 = x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3$$

### 6.3.4 Point Multiplication and ECDLP

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  $O, P, 2P, \dots, (r-1)P$ . Then if we define the scalar  $k$  as an integer within the range  $[1, r-1]$ , we can represent point multiplication, also known as scalar multiplication as the following:  $Q = kP$ , where  $Q$  is also a point in the subgroup generated by  $P$ .

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 8.** *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 assumed 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 [96] and the ElGamal [97] cryptosystems. “Harder to solve” implies shorter keys are needed to provide same level of security as recommended by [98]. Table 6.1 shows the comparison for elliptic curves and RSA.

**Table 6.1:** NIST’s recommendation for key management

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

Common methods for solving ECDLP are Pollards rho algorithm and index-calculus method, we refer readers to [2] for more details.

## 6.4 Open Research Questions

### 6.4.1 Summation Polynomials

Let  $E$  be a general elliptic curve over field  $F$  in Weierstrass form given by the equation 6.1 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 [99]:

$$S_3(X_1, X_2, X_3) = (X_1^2 X_2^2 + X_1^2 X_3^2 + X_2^2 X_3^2) - 2(X_1^2 X_2 X_3 + X_1 X_2^2 X_3 + X_1 X_2 X_3^2) \quad (6.4)$$

$$- d_2(X_1 X_2 X_3) - d_4(X_1 X_2 + X_1 X_3 + X_2 X_3) - d_6(X_1 + X_2 + X_3) -$$

$d_8$

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

$$S_m(X_1, \dots, X_m) = \text{Res}_X(S_{m-r}(X_1, \dots, X_{m-r-1}, X), S_{r+2}(X_{m-r}(X_{m-r}, \dots, X_m, X))$$

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

Summation Polynomials were first introduced by Semaev in 2004 [41]. Semaev's summation polynomials have the property that if  $S_m(a_1, \dots, a_m) = 0$  for some field elements  $a_1, \dots, a_m \in F$  if and only if there are elliptic curve points  $(a_1, b_1), \dots, (a_m, b_m)$  on  $E$  such that  $(a_1, b_1) + \dots + (a_m, b_m) = 0$ . The idea is to represent point addition in elliptic curves using a multivariate equation system and trying to solve the equation system. This topic has been studied by a lot of researchers trying to solve the ECDLP problem [100, 101, 102, 103, 104, 105]. In 2015 a new method was introduced by Semaev [40] 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 polyno-

mials  $S_3(X_1, X_2, X_3)$  for curves over  $\mathbb{F}_p$  and  $\mathbb{F}_2^m$ , then introduce some open research questions for future work.

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

For an elliptic curve over a field  $K$  of characteristic  $> 3$  we recall the curve equation 6.2  $y^3 = x^3 + a_4x + a_6$  and  $S_3(X_1, X_2, X_3)$  equation can be simplified to: [99]

$$\begin{aligned} S_3(X_1, X_2, X_3) &= (X_1^2X_2^2 + X_1^2X_3^2 + X_2^2X_3^2) - 2(X_1^2X_2X_3 + X_1X_2^2X_3 + X_1X_2X_3^2) \\ &\quad - 2a_4(X_1X_2 + X_1X_3 + X_2X_3) - 4a_6(X_1 + X_2 + X_3) + a_4^2 \\ &= (X_1 - X_2)^2X_3^2 - 2((X_1X_2 + a_4)(X_1 + X_2) + 2a_6)X_3 + (X_1X_2 - a_4)^2 \\ &\quad - 4a_6(X_1 + X_2) \end{aligned} \tag{6.5}$$

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

$$-2(X_1^3 + 2a_4X_1 + 2a_6)X_3 + (X_1^2 - a_4)^2 - 8a_6X_1 = 0 \tag{6.6}$$

### Special Curve secp256k1

For curve secp256k1<sup>4</sup> where  $a_4 = 0$  and  $a_6 = 7$ , equation 6.5 and equation 6.6 can be written as the following:

$$\begin{aligned} S_3(X_1, X_2, X_3) &= (X_1 - X_2)^2X_3^2 - 2((X_1X_2)(X_1 + X_2) + 14)X_3 + X_1^2X_2^2 \\ &\quad - 28(X_1 + X_2) \end{aligned} \tag{6.7}$$

and for point doubling when  $X_1 = X_2$

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

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

---

<sup>4</sup>Elliptic curve used in bitcoin, we will give more details in section 7.1



**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 have  $d_2 = 1$ ,  $d_4 = 0$ ,  $d_6 = 0$  and  $d_8 = a_6$  (see section 6.2). In binary field anything multiply by 2 equal to 0,  $A - B = A + B$ , thus equation 6.4 can be simplified to:

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

and when  $X_1 = X_2$  we have:

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

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

**6.4.2 Semaev Cipher**

In Semaev's recent paper [40] he introduced a new algorithm trying to solving ECDLP using summation polynomial for curves over  $F_{2^m}$ . By recursively compute summation polynomials, instead of trying to write  $R = P_1 + \dots + P_m$  for point  $P_i$ , we can write  $Q_1 = P_1 + P_2, Q_2 = Q_1 + P_3, \dots, R = Q_{m-2} + P_m$ , where the  $Q_i$  are completely arbitrary points (see equation 6.8). Then solve the multivariate equation

system under some assumptions.

$$\left\{ \begin{array}{l} S_3(Q_1, P_1, P_2) \\ S_3(Q_1, Q_2, P_3) \\ S_3(Q_2, Q_3, P_4) \\ \vdots \\ S_3(Q_i, Q_{i+1}, P_{i+2}) \\ \vdots \\ S_3(Q_{m-2}, P_m, R). \end{array} \right. \quad (6.8)$$

However the final result of Semaev's new work is still uncertain, some researchers believes the assumption of Semaev's paper is wrong, cf. later work by Kisters and Yeo [99] and blog posts [106, 107]. This leads to some open research questions.

The new equation system introduced in Semaev's new paper have clear block cipher topology (see figure 3.1 in section 3.5) [38]. It is very similar to the block cipher equations we have tried to solve for GOST and SIMON [51, 108]. It is potentially to be solved by methods used in algebraic cryptanalysis [38]. We call Semaev's new summation polynomial equations **Semaev cipher** and plan to explore the following research questions:

1. Can we solve Semaev cipher using SAT solvers with some clever guessing (similar as our previous work for guess-then-determine attack on GOST)?
2. What about the equations when  $X_1 = X_2$  ? Can we solve those equations efficiently which will lead to speed improvements for the bitcoin elliptic curve implementation?

These questions are not directly related to the password guessing work we are discussing in this part. We will leave it as future work at this moment.

### 6.4.3 Curves with Special Shortcuts

Let  $E$  be an elliptic curve over  $\mathbb{F}_p$  with prime number  $p$  and curve order  $n$ . For any point  $Q = (x, y)$  on curve  $E$ , there exist  $\lambda$  and  $\zeta$  such that

$$\lambda \cdot Q = (\zeta \cdot x, y)$$

So that for computing point multiplication  $k \cdot Q$ , one can use the above shortcut compute  $Q' = \lambda \cdot Q$ . Then decompose  $k$  to  $k = k_1 + k_2 \cdot \lambda \mod n$ . Then

$$k \cdot Q = (k_1 + k_2 \cdot \lambda) \cdot Q = k_1 \cdot Q + k_2 \cdot \lambda \cdot Q = k_1 \cdot Q + k_2 \cdot Q'$$

Since  $k_2 \cdot Q'$  can be computed efficiently, total number of point doubling for computing  $k \cdot Q$  is reduced. An example have been given for curve secp160k1 in [2].

We looked at elliptic curve secp256k1 (used in Bitcoin, detail parameters introduced in section 7.1) where prime

$$p = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC2F}$$

and curve order

$$n = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF BAAEDCE6 AF48A03B BFD25E8C D0364141}$$

For secp256k1, we have  $p \equiv 1 \mod 6$ , there exists a primitive 6th root of unity  $\zeta \in \mathbb{F}_p$  and  $\zeta^6 - 1 = 0 \mod n$ . Since

$$\zeta^6 - 1 = (\zeta^2 - 1) \cdot (\zeta^2 + \zeta - 1) \cdot (\zeta^2 - \zeta - 1)$$

any root of  $\zeta^6 - 1$  is root of one of the above three polynomials.

We found special multiples for secp256k1 as following:

Let  $\lambda_6 =$

$$\text{AC9C52B33FA3CF1F5AD9E3FD77ED9BA4A880B9FC8EC739C2E0CFC810B51283CF}$$

a primitive 6th root of unity mod  $n$ . We have  $\lambda_6 - \lambda_6^2 - 1 = 0 \mod n$ .

Let  $\zeta_6 =$

851695D49A83F8EF919BB86153CBCB16630FB68AED0A766A3EC693D68E6AFA40

be a (not primitive) 6-th root of unity mod  $P$ . We have  $\zeta_6 + \zeta_6^2 + 1 \mod P$ .

We have for ANY point:

$$\lambda \cdot (x, y) = (\zeta \cdot x, y)$$

This property is not common for every elliptic curve. We have discovered more special shortcuts like this, and we will investigate how they can be used to speed up the point multiplication process. This should lead to future improvements in the speed of bitcoin ECDSA signature and verification process.

## 6.5 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) was independently proposed by Neal Koblitz[109] and Victor Miller 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 practise since its inclusion in standards by organisations such as ISO, IEEE, NIST, etc. Elliptic curves are more efficient [110] and offer smaller key sizes [111] at the same security as other widely adopted public key cryptography schemes such as RSA [96].

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 part of PhD work we only focus on ECDSA [112] (key generation part in particular) which is used in Bitcoin and we refer the readers to [2] for details of other schemes.

In section 6.5.1 we will describe elliptic curve domain parameters, in section 6.5.2 we will discuss elliptic curve key pair generation process, and briefly introduce ECDSA in section 6.5.3.

### 6.5.1 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. Different countries might have their own specified standards for choice of elliptic curves, which we will give a list in section ???. The specification of a chosen elliptic curve is defined using domain parameters.

Domain parameters for an elliptic curve scheme describes 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  $D = (q, \text{FR}, S, a, b, G, n, h)$  where

1.  $q$  is the field order
2. FR (field representation) is an indication of the representation used for the elements of  $\mathbb{F}_q$
3. If the curve is randomly generated,  $S$  is the seed used to generate the curve
4.  $a, b \in \mathbb{F}_q$  that define the curve equation over field  $\mathbb{F}_q$
5.  $G$  is the base point where  $G = (G_x, G_y) \in E(\mathbb{F}_q)$
6. The order  $n$  of  $G$
7. The cofactor  $h = \frac{\#E(\mathbb{F}_q)}{n}$

### 6.5.2 Key Pair Generation

An elliptic curve key pair is associated on a particular set of valid domain parameters (cf. [2] 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  $d = \log_G Q$ . Key pair generation algorithm is the following:

**Algorithm 2** Key pair generation [2]Input: Domain parameters  $D = (q, FR, S, a, b, G, n, h)$ Output: Public key  $Q$ , private key  $d$ 

- 1: Select  $d \in_R [1, n - 1]$ .
- 2: Compute  $Q = dP$ .
- 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 numbers  $d$  should be **random** in the sense that the probability of any particular value being selected must be sufficiently small to avoid an adversary from gaining advantage through optimising a search strategy based on such 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.  $Q \neq O$  ( $O$  is the point at infinity)
2.  $Q_x$  and  $Q_y$  are properly represented elements of  $\mathbb{F}_q$  (i.e., integers in  $[0, q - 1]$  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.  $nQ = O$

### 6.5.3 Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is a mathematical scheme based on elliptic curve cryptography that authenticates the identity of a message signer, and checks that the content of the message is not modified. ECDSA is the most widely standardised elliptic curve based signature scheme, appearing in the FIPS 186-2[113], IEEE 1363-2000 [114], ANSI X9.62 [115] etc. Typically, ECDSA is consists of three parts: key generation, signing and verification. We have discussed the key generation part in previous section, see algorithm 2. Now we look at sign 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  $D = (q, FR, S, a, b, G, n, h)$ . The process for signature generation are described in algorithm 3. In the following algorithms,  $H$  denotes a cryptographic hash function whose outputs have length no more than  $n$  (otherwise, the outputs of  $H$  can be truncated).

---

**Algorithm 3** ECDSA signature generation[2]

---

Input: Domain parameters  $D = (q, FR, S, a, b, G, n, h)$ , private key  $d$ , message  $m$

Output: Signature  $(r, s)$

- 1: Select  $k \in_R [1, n - 1]$ .
  - 2: Compute  $kP = (x_1, y_1)$  and covert  $x_1$  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 process is described as following:

---

**Algorithm 4** ECDSA signature verification[2]

---

Input: Domain parameters  $D = (q, FR, S, a, b, G, n, h)$ , 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  $X = u_1P + u_2Q$ .
  - 6: If  $X$  is point at infinity  $O$  then reject the signature
  - 7: Covert 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.
-

## Chapter 7

# Bitcoin and Brain Wallet Attacks

Bitcoin is a virtual currency, an electronic payment system based on cryptography. It was invented by Satoshi Nakamoto<sup>1</sup> in 2008 [116]. In 2009, Bitcoin was launched as an open-source software. Bitcoin is designed to be fully decentralised based on peer-to-peer network, which means self-governing without support from trust entities, banks or government. Bitcoin transactions are like checks but signed cryptographically instead of using ink. Transactions are broadcast to the peer-to-peer network and each node in the peer-to-peer network can verify them. Transactions are recorded in a public distributed ledger called the block chain. All bitcoin transaction history is public and pseudonymous.

Creation of new bitcoins is through a process called mining and participants are called miners. Miners offer their computation power to solve a hard mathematics problem, winners will be rewarded the newly created bitcoin. Chance of winning a reward is directly related to the miner's computation power. During the process of mining, transactions have been processed and recorded into the block chain.

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 must digitally sign the transaction using their corresponding private key. Then broadcast the signed transaction to the peer-to-peer network. Everyone else in 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 cosponsoring private key.

---

<sup>1</sup>It's not known whether Satoshi Nakamoto is a real or pseudonym name or if it represents one or a group



Once the private is lost, the bitcoin network will not recognize any other evidence of ownership.

Bitcoin uses digital signature protect the ownership bitcoin and private key is the only evidence of owning bitcoin. Thus it is very important to look at the technical details of the digital signature scheme used in bitcoin. In the next section we look at bitcoin brain wallet, our main attack target in this part of research work. We discuss the technical details of how bitcoin address are generated, using which elliptic curve, and how the curve is defined.

## 7.1 Bitcoin Elliptic Curve

Bitcoin uses elliptic curve digital signature algorithm (see section 6.5.3). The elliptic curve used in bitcoin is called secp256k1. In the FIPS 186-2 standard [113] NIST recommends five elliptic curves for use in the curve digital signature algorithm targeting five different security levels (192,224,256,384,521). In the standard, these curves are name P-192, P-224, P-256, P-384, and P521 <sup>2</sup>. The bitcoin elliptic curve is proposed in Certicom [117] 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:

$$p = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFFE FFFFFFFC2F}$$

$$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

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:

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

$$G_y = \text{483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8}$$

---

<sup>2</sup>in practice also appear as nistp192, nistp224 etc, in Certicom recommended curves they are named as secp\*\*\*r1, and in openssl they are called prime\*\*\*v1

and the order  $n$  of  $G$  and the cofactor  $h$  are:

$n = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141}$

$h = 1$

## 7.2 Brain Wallet

Bitcoin wallet is a collection of your bitcoin addresses and stores the corresponding key pairs for those addresses. Bitcoin wallets comes in different form, desktop software, mobile apps, online service, hardware, smart card or paper. The idea is to keep your private keys safe.

As we discussed earlier in section 6.5.2, the private key is a number which is supposed to be totally random. Normally it will be a long hex string which is very hard for a person to remember and stored in your wallet. No matter what form of wallet you are using, there always exists a chance that you might lose your wallet. Even till now, many companies have already provided smart ideas of protecting your private key from hackers, malwares, etc. Users are still required to keep something (maybe not your private key in plain text) safe and hope it won't be stolen in order to recover their private keys once they lost their wallet.

Brain wallet is another solution, which does not need you to keep anything in safe and still be able to recover your private key. Instead of storing your private key and protecting it, you can store it in your mind. Brain wallet creates private key by a human chosen password or a passphrase, then using SHA-256 hash algorithm to obtain an encrypted hash string. As SHA-256 is a deterministic method, you can always use your password remembered in your brain to recover the private key. Note that brain wallet uses the hash directly as private key, the security of your private keys now depends on how unpredictable your passwords are.

Now we give an example to show how bitcoin brain wallets are generated by using password "password"

1. Private key: SHA256("password")

5E884898DA28047151D0E56F8DC6292773603D0D6AABBDD62A11EF721D1542D8

2. Public key (uncompressed) : Elliptic curve secp256k1 key pair generation

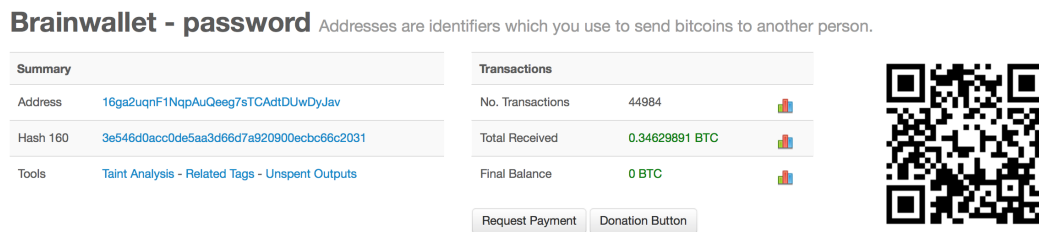
04B568858A407A8721923B89DF9963D30013639AC690CCE5F555529B77B83CBFC7  
6950F90BE717E38A3ECE1F5558F40179F8C9502DECA11183BB3A3AEA797736A6

3. SHA256 (Public key)

1D8ED6551EE910136EB0EA735106E137565E8F5EBF8DF73A6A877C92C049F922

4. Hash160 : RIPEMD160

3E546D0ACC0DE5AA3D66D7A920900ECBC66C2031  
(used for transaction)



**Figure 7.1:** Brainwallet generated by password “password”

Brainwallet users are normally suggested to use strong password or passphrase. Websites provide brainwallet generation service normally use figure 7.2 to tell user what is a strong password. However this figure is a little bit misleading which makes user feels it is safe to use brainwallet with a passphrase. In order to crack brainwallet, there are two directions for optimisation: **speed** of password guessing and **efficiency** of password guessing.

In the next section we will discuss about existing method of bitcoin elliptic curve implementation, benchmarking the state-of-art attack and show an improvement method running the attack with much more faster speed on a laptop. It's also important to note that attacks can use much more CPUs with a cheap price to speed up the password cracking process way more faster.

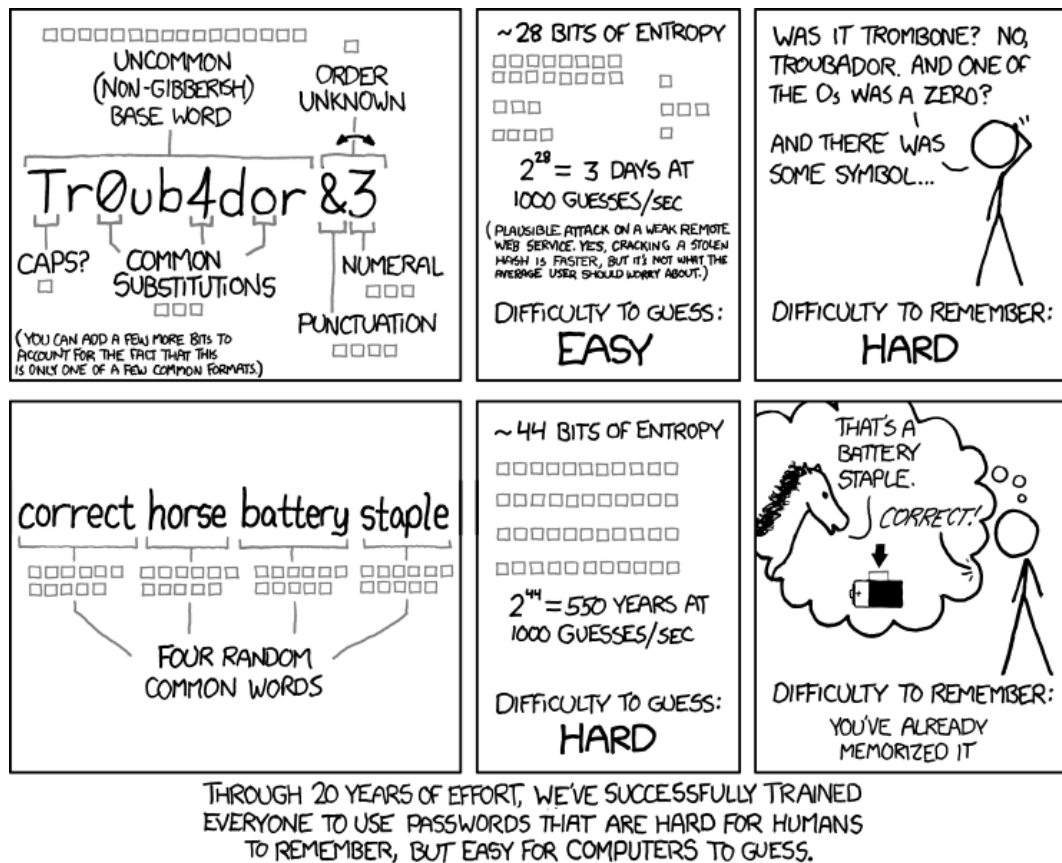


Figure 7.2: password strength comparison between using password and passphrase

## 7.3 Bitcoin Elliptic Curve Implementation and Benchmarking

### 7.3.1 Related Work

We are not the first one try to crack bitcoin brainwallet, a lot of hackers are doing it. Victims also found their money been stolen and post it in forums. The first brainwallet attack was announced publicly in a recent hacking conference DefCon 23 (Aug 2015). Ryan Castellucci, a white hat hacker presented his attack on cracking brainwallet, and also published his hacking software[118]. Ryan's attack was done on an intel i7 PC with 8 cores. The attack speed can reach approximately 16,250 password per second on each core and he had cracked more than 18,000 brainwallet addresses<sup>3</sup>.

<sup>3</sup>We did contact Ryan in private after his talk and got his detailed results, these numbers are not in his online presentation.

The software Ryan has published is using an existing opensource secp256k1 bitcoin elliptic curve implementation mainly written by Pieter Wullie, one of bitcoin core developers. This is considered the current best bitcoin elliptic curve implementation in terms of code level optimisation. Pieter's implementation is written in C based on the Multiple Precision Integers and Rationals (MPIR) library.

The rest of this section we will look at what has been implemented in Pieter's secp256k1 library, how did the current best attack uses the implementation, show the benchmarking result on our machine, and discuss how did we improved the attack.

### 7.3.2 Special Designed Point Multiplication Method For Attack

The process of cracking bitcoin brain wallet is to repeatedly generated public keys in bitcoin elliptic curve using generated password. Key generation method as we described in 6.5.2, 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 7.1). We first benchmark the current best implementation secp256k1 library. All benchmark results are running on a laptop with the following

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

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

**47.2 us.**

#### 7.3.2.1 Fixed Point Multiplication Methods

The most basic and naive method for point multiplication  $Q = kP$  with a unknown point  $P$  is double-and-add method [2]. The idea is to use binary representation for  $k$ :

$$k = k_0 + 2k_1 + 2^2k_2 + \cdots + 2^mk_m$$

where  $[k_0 \dots k_m] \in \{0, 1\}$  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[2]

---

```

1:  $Q := \text{infinity}$ 
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 double-and-add method will need  $\frac{m}{2} + mD$  computations 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 precomputing some data that depends only on  $P$ . For example if the points  $2P, 2^2P, \dots, 2^{m-1}P$  are precomputed, then the double-and-add method (algorithm 5) has expected running time  $(\frac{m}{2})A$ , and all doublings are eliminated.

In [119] the authors introduced a new method for fix point multiplication. The precomputing step store every multiple  $2^iP$ . Let  $(K_{d-1}, \dots, K_1, K_0)_{2^w}$  be the base- $2^w$  representation of  $k$ , where  $d = \lceil m/w \rceil$ , and let  $Q_j = \sum_{i:K_i=j} 2^{wi}P$  for each  $j$ ,  $1 \leq j \leq 2^w - 1$ , Then

$$\begin{aligned}
 kP &= \sum_{i=0}^{d_1} K_i(2^{wi}P) = \sum_{j=1}^{2^w-1} (j \sum_{i:K_i=j} 2^{wi}P) = \sum_{j=1}^{2^w-1} jQ_j \\
 &= Q_{2^w-1} + (Q_{2^w-1} + Q_{2^w-2}) + \dots + (Q_{2^w-1} + Q_{2^w-2} + \dots + Q_1)
 \end{aligned} \tag{7.1}$$

**Algorithm 6** Fixed-base windowing method for point multiplication[2]INPUT: Window width  $w$ ,  $d = \lceil m/w \rceil$ ,  $k = (K_{d-1}, \dots, K_1, K_0)_{2^w}$ OUTPUT:  $kP$ 

- 
- 1: Precompute  $P_i = 2^{wi}P, 0 \leq i \leq d-1$
  - 2:  $A \leftarrow \text{infinity}, B \leftarrow \text{infinity}$
  - 3: **for**  $j$  from  $2^w - 1$  downto 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 [2] we noticed they are all memory friendly implementations which does not take a lot of memory space for precomputation. However, we are working on a different task and aims to repeatedly run point multiplication method for many many times. We have implemented an extreme version of window method which will take much more precomputation space than methods introduced in [2].

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 tableINPUT: Window width  $w$ ,  $d = \lceil m/w \rceil$ ,  $k = (K_{d-1}, \dots, K_1, K_0)_{2^w}$ 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:  $A \leftarrow \text{infinity}$
  - 3: **for**  $i$  from 0 to  $d-1$  **do**
  - 4:    $A \leftarrow A + P_{i,j}$  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<sup>4</sup>, our benchmark results are shown in table 7.1. Note that precomputation stores elliptic curve point  $P = x, y$  where  $x$  and  $y$  are 32 bits integer array of size 10. So one store point need 80 Bytes memory space.

**Table 7.1:** Time cost for different window width  $w$ , point addition method secp256k1 library [1] 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

### 7.3.3 Point Representation

As we described in section 6.3.3, representing a point in affine coordinate  $P(x, y)$  on an elliptic curve over  $\mathbb{F}_p$ , the field operations for calculating point addition need 2 multiplications, 1 square and one modular inverse (for short, 2M+1S+1I). Modular inverse is more expensive operation compare 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 are benchmarking are: openssl-1.0.2a (released in March 2015) and mpir-2.5.2 (released in Oct 2012), and the Pieter Wullie's implementation on github [1] <sup>5</sup>.

The results are shown in table 7.2. The benchmarking shows modular inverse is much more expensive than multiplication and square. It is also important to notice, for openssl big number library, square operation is more expensive than multiplication, and for MPIR library, 1 square = 0.75 multiplication. As modular inverse is 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  $(X : Y : Z)$ ,  $Z \neq 0$ ,

<sup>4</sup>larger than 22 will take too long for precomputation and my laptop start to have slow response

<sup>5</sup>with the following configuration: USE\_NUM.GMP USE\_FIELD\_10x26  
USE\_FIELD\_INV\_NUM USE\_SCALAR\_8x32 USE\_SCALAR\_INV\_BUILTIN



**Table 7.2:** 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

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  $(\frac{X}{Z^2}, \frac{Y}{Z^3})$ . 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 need for point addition and point doubling are shown in table 7.3. We see that Jacobian coordinates yield the fastest point doubling, while mixed Jacobian-affine coordinates yield the fastest point addition.

**Table 7.3:** Operation counts for point addition and doubling. A = affine, P = standard projective, J = Jacobian [2, 3]

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 [2, 3] for other detailed equations in different coordinates. Here we 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 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:

$$\begin{aligned} X_3 &= (Y_2 Z_1^3 - Y_1)^2 - (X_2 Z_1^2 - X_1)^2 (X_1 + X_2 Z_1^2) \\ Y_3 &= (Y_2 Z_1^3 - Y_1)(X_1(X_2 Z_1^2 - X_1)^2 - X_3) - Y_1(X_2 Z_1^2 - X_1)^3 \\ Z_3 &= (X_2 Z_1^2 - X_1)Z_1 \end{aligned} \quad (7.2)$$

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:

$$\begin{aligned} A &\leftarrow Z_1^2, B \leftarrow Z_1 \cdot A, C \leftarrow X_2 \cdot A, D \leftarrow Y_2 \cdot B, E \leftarrow C - X_1, \\ F &\leftarrow D - Y_1, G \leftarrow E^2, H \leftarrow G \cdot E, I \leftarrow X_1 \cdot G, \\ X_3 &\leftarrow F^2 - (H + 2I), Y_3 \leftarrow F \cdot (I - X_3) - Y_1 \cdot H, Z_3 \leftarrow Z_1 \cdot E. \end{aligned}$$

In [120]

**secp256k1 point addition formulas** In the latest version, secp256k1 point addition formulas are based on [121] which introduced a strongly unified addition formulas for standard projective coordinate. Bitcoin developers implemented mixed coordinate formula (Jacobian-Affine) version based on [121].

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 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:

$$\begin{aligned} X_3 &= 4(K^2 - H) \\ Y_3 &= 4(R(3H - 2K^2) - G^2) \\ Z_3 &= 2FZ_1 \end{aligned} \quad (7.3)$$

where

$$\begin{aligned} A &= Z_1^2, B = Z_1 \cdot A, C = X_2 \cdot A, D = Y_2 \cdot B, E = X_1 + C \\ F &= Y_1 + D, G = F^2, H = E \cdot G, I = E^2, J = X_1 \cdot C, K = I - J \end{aligned}$$

**Bernstein-Lange point addition formulas** In [122], Bernstein introduced the following method which take 7M+4S, the explicit formulas are given as following [120]

$$\begin{aligned} X_3 &= r^2 - J - 2V \\ Y_3 &= r \cdot (V - X_3) - 2Y_1 \cdot J \\ Z_3 &= (Z_1 + H)^2 - Z_1^2 - H^2 \end{aligned} \tag{7.4}$$

where

$$\begin{aligned} U2 &= X_2 \cdot Z_1^2, S2 = Y_2 \cdot Z_1^3 \\ H &= U2 - X_1, I = 4H^2 \\ J &= H \cdot I, r = 2(S2 - Y_1), V = X_1 \cdot I \end{aligned}$$

#### Detailed Field Operation Benchmarks

From the results of table 7.2 we saw that Wullie's secp256k1 library [1] have a much faster field multiplication and square speed than openssl and mpir library. Wullie'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 <sup>6</sup>. 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 7.3 in method *secp256k1\_gej\_add\_ge\_var*, which uses 8 multiplications, 3 squares and 12 multiply integer / addition / negation. Equation 7.2 is implemented in *secp256k1\_gej\_add\_ge* which uses 7 multiplications, 5 squares and 21 multiply integer / addition / negation. We have implemented equation 7.4 which

---

<sup>6</sup>Depends on whether compiler support 64 bits integer

take 7 multiplication, 4 squares and 22 multiply integer / addition / negation.

It is important to notice the square and multiplication difference we discussed in table 7.2. In [123] Bernstein listed best operation counts based on different assumptions:  $S = 0M$ ,  $S = 0.2M$ ,  $S = 0.67M$ ,  $S = 0.8M$  and  $S = 1M$ . In [124], the author discussed the ratio  $S/M$  is almost independent of the field of definition and of the implementation, and can be reasonably taken equal to 0.8. Our benchmark results is very similar to  $S = 0.8M$  (see table 7.2). In [120], other field operations are considered as  $0M$ , in table 7.4 our benchmark results shows field addition and other operations have approximately 0.1M cost.

**Table 7.4:** Field operation counts and benchmark results

	#Multiplication 1M	#Square $\approx 0.8 M$	#add/neg/*int $\approx 0.1 M$	#fe_cmov $\approx 0.2 M$	total time cost
secp256k1_gej_add_ge	7	5	15	6	$\approx 0.681 \text{ us}$
secp256k1_gej_add_ge_var	8	3	12	0	$\approx 0.562 \text{ us}$
7M + 4S code	7	4	21	0	$\approx 0.594 \text{ us}$

The secp256k1\_gej\_add\_ge method which is also the default method for key generation, uses 6 secp256k1\_fe\_cmov operations which has a cost approximately 0.2 M. The main reason of writing code in such a way is stated in the code, the author's comments:

*"This formula has the benefit of being the same for both addition of distinct points and doubling"[1]*

The propose 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 7.3.2) and specific library configuration (footnote in section 7.3.3). 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 7.5

**Table 7.5:** 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
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
secp256k1_gej_add_ge	45.85 us	22.16 us	15.35 us	11.23 us	9.23 us
secp256k1_gej_add_ge_var	<b>37.37 us*</b>	17.86 us	12.21 us	8.89 us	<b>7.16 us</b>
7M + 4S code	39.01 us	18.79 us	12.77 us	9.23 us	7.48 us
covert Jacobian to Affine	$\approx 10$ us				
Benchmark on my laptop i7-3520m 2.9 GHz CPU	$\approx 42$ K guesses / sec (single thread)				
Defcon Attack** i7-2600 3.2 GHz CPU	$\approx 130$ K guesses / sec				
Improved Defcon attack**	$\approx 315$ K guesses / sec				

\* Defcon attack [118] is equivalent to this results

\*\* Results are reported by Ryan Castellucci running his Defcon code and our improved code on 8 threads with linux gcc compiler.

Defcon attack [118] published code on github in Aug using a faster version of secp256k1 library <sup>7</sup>, and the results is marked as \* in table 7.5. Our best result using 1.09 GB precomputation memory gives  $\approx$  **2.5 times speed up** for key generation process than the current known best attack.

## 7.4 Experiment Results

We have collected all hash160 on blockchain data (until June 2015), 89,872,723 unique addresses has ever been used, we have cracked 18,250 addresses in total. This is still on going research. I'll show some interesting results during the viva.

### 7.4.1 Network Stress Test

Currently the maximum block size is one megabyte per block. On 11 November 2014, when the block is around 30% full, David Hunduson wrote a blog post analyzing what will happen on the network as we approach 100% full blocks. David Hunduson did a simulation which shows if the block is 100% full then 10% of all transactions would still not have received a confirmation after 22800 seconds (6.3

---

<sup>7</sup>Also written by Pieter Wullie one year ago, this version is performance focused and using 8M+3S

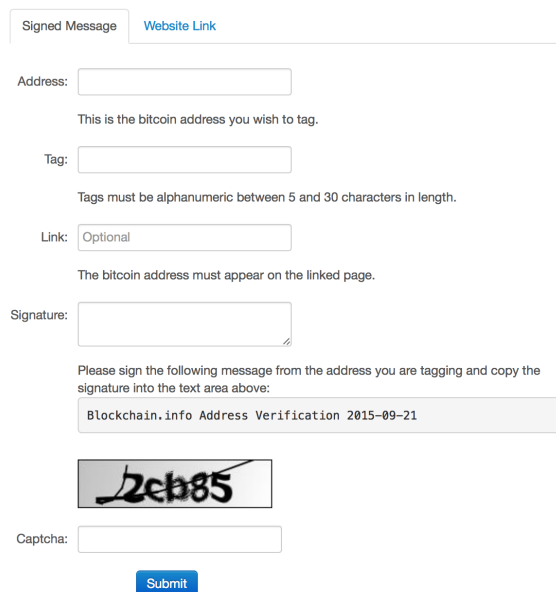
hours).

Later on 4th May, head Bitcoin developer Gavin Andersen discussed this problem in his blog and few days later he suggested to increase the maximum block size to 20MB. Over the past few months, the Bitcoin community has been roiling with a debate about whether or not to implement a code change that would enable Bitcoin to handle worldwide adoption by increasing the size of the blocks on the chain. The catch is that it would split the blockchain in the process, requiring everyone to move over to a new version of the ledger.

As a result of this debate, CoinWallet.eu, a Bitcoin exchange in UK decided to launch a stress test for the Bitcoin network. CoinWallet launched two stress test

### 7.4.2 Disclosure of results

This is still under discussion. All the addresses are currently empty, but some addresses are still in used quite recently. One of the ideas we currently have in mind is to tag the cracked addresses on blockchain.info. See figure 7.3 and 7.4. But the process is not automated due to captcha requirement and we do have a lot of cracked password.



Signed Message [Website Link](#)

Address:

This is the bitcoin address you wish to tag.

Tag:

Tags must be alphanumeric between 5 and 30 characters in length.


Link:  Optional

The bitcoin address must appear on the linked page.

Signature:

Please sign the following message from the address you are tagging and copy the signature into the text area above:

Blockchain.info Address Verification 2015-09-21



Captcha:

**Figure 7.3:** Blockchain.info tag address page

### Brainwallet - password1 Addresses are ide

#### Summary

Address [19VAb9zAhpWLWfEuqw9HXup2zaNoNPPyE](#)

Hash 160 [5d14a857fce8da8edfb8f7d1c4bbc316622b7227](#)

Tools [Taint Analysis](#) - [Related Tags](#) - [Unspent Outputs](#)

**Figure 7.4:** Example of tagged brainwallet address

## Chapter 8

# General Conclusions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.



## **Appendix A**

# **An Appendix About Stuff**

(stuff)

## **Appendix B**

# **Another Appendix About Things**

(things)

## Appendix C

# Colophon

*This is a description of the tools you used to make your thesis. It helps people make future documents, reminds you, and looks good.*

(example) This document was set in the Times Roman typeface using  $\text{\LaTeX}$  and Bib $\text{\TeX}$ , composed with a text editor.

# Bibliography

- [1] Petier Wullie. bitcoin secp256k1 library, version 2015/08/11. <https://github.com/bitcoin/secp256k1>.
- [2] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.
- [3] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. *Software implementation of the NIST elliptic curves over prime fields*. Springer, 2001.
- [4] Nicolas Courtois. Guiding principles of effective crypto and security engineering. [http://www.nicolascourtois.com/papers/sc/Pr\\_Crypto\\_eng\\_v12.pdf](http://www.nicolascourtois.com/papers/sc/Pr_Crypto_eng_v12.pdf).
- [5] Gustavus J Simmons et al. *Contemporary cryptology*, volume 3. IEEE press New York, 1992.
- [6] Ronald Cramer and Victor 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.
- [7] Lars R Knudsen. Block ciphersa survey. In *State of the Art in Applied Cryptography*, pages 18–48. Springer, 1998.
- [8] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

- [9] Donald Watts Davies and Wyn L Price. *Security for computer networks: and introduction to data security in teleprocessing and electronic funds transfer*. John Wiley & Sons, Inc., 1989.
- [10] David Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster, 1996.
- [11] Horst Feistel. Cryptography and computer privacy. *Scientific american*, 228:15–23, 1973.
- [12] NIST FIPS PUB. 46-3. data encryption standard. *Federal Information Processing Standards, National Bureau of Standards, US Department of Commerce*, 1977.
- [13] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.
- [14] Brouce Schneier. Applied cryptography. protocols, algorithms, and source code in c/bruce schneier, 1996.
- [15] Lars Ramkilde Knudsen. Block ciphers: analysis, design and applications. *DAIMI Report Series*, 23(485), 1994.
- [16] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Advances in CryptologyEUROCRYPT93*, pages 386–397. Springer, 1994.
- [17] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of feal cipher. In *Advances in CryptologyEurocrypt92*, pages 81–91. Springer, 1993.
- [18] Burton S Kaliski Jr and Matthew JB Robshaw. Linear cryptanalysis using multiple approximations. In *Advances in CryptologyCrypto94*, pages 26–39. Springer, 1994.
- [19] Joe Kilian and Phillip Rogaway. How to protect des against exhaustive key search. In *Advances in CryptologyCRYPTO96*, pages 252–267. Springer, 1996.

- [20] Lawrence Brown, Josef Pieprzyk, and Jennifer Seberry. Lokia cryptographic primitive for authentication and secrecy applications. In *Advances in CryptologyAUSCRYPT'90*, pages 229–236. Springer, 1990.
- [21] Nicolas T Courtois. Feistel schemes and bi-linear cryptanalysis. In *Advances in Cryptology—CRYPTO 2004*, pages 23–40. Springer, 2004.
- [22] E. Biham and A. Shamir. *Differential cryptanalysis of the data encryption standard*. Springer-Verlag, 1993.
- [23] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.
- [24] Don Coppersmith. The development of des. *Invited talk at CRYPTO*, 2000.
- [25] Nicolas T Courtois, Guilhem Castagnos, and Louis Goubin. What do des s-boxes say to each other? *IACR Cryptology ePrint Archive*, 2003:184, 2003.
- [26] Lars Ramkilde Knudsen. Cryptanalysis of loki. In *Advances in CryptologyASIACRYPT'91*, pages 22–35. Springer, 1993.
- [27] James L Massey. Safer k-64: A byte-oriented block-ciphering algorithm. In *Fast Software Encryption*, pages 1–17. Springer, 1994.
- [28] Nicolas T Courtois and Michał Misztal. First differential attack on full 32-round gost. In *Information and Communications Security*, pages 216–227. Springer, 2011.
- [29] Nicolas Courtois and Michał Misztal. Differential cryptanalysis of gost. *IACR Cryptology ePrint Archive*, 2011:312, 2011.
- [30] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers simon and speck. In *Fast Software Encryption*, pages 546–570. Springer, 2014.
- [31] Hoda AlKhzaimi and Martin M Lauridsen. Cryptanalysis of the simon family of block ciphers. *IACR Cryptology ePrint Archive*, 2013:543, 2013.

- [32] Nicolas Courtois. The best differential characteristics and subtleties of the biham-shamir attacks on des. *IACR Cryptology ePrint Archive*, 2005:202, 2005.
- [33] LarsR. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer Berlin Heidelberg, 1995.
- [34] Nicolas T Courtois. An improved differential attack on full gost. *IACR Cryptology ePrint Archive*, 2012:138, 2012.
- [35] Theodosios Mourouzis. *Optimizations in algebraic and differential cryptanalysis*. PhD thesis, UCL (University College London), 2015.
- [36] Nicolas T Courtois, Theodosios Mourouzis, Michał Misztal, Jean-Jacques Quisquater, and Guangyan Song. Can gost be made secure against differential cryptanalysis? *Cryptologia*, 39(2):145–156, 2015.
- [37] C. Shannon. Communication theory of secrecy systems. In *Bell System Technical Journal* 28, 1949.
- [38] Nicolas T Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology ASIACRYPT 2002*, pages 267–287. Springer, 2002.
- [39] Nicolas Courtois. 100 years of cryptanalysis: Compositions of permutations. [http://www.nicolascourtois.com/papers/code\\_breakers\\_enigma\\_block\\_teach.pdf](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.
- [40] Igor Semaev. New algorithm for the discrete logarithm problem on elliptic curves. *Cryptology ePrint Archive*, Report 2015/310, 2015.
- [41] Igor Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

- [42] Aviezri S Fraenkel and Yaacov Yesha. Complexity of solving algebraic equations. *Information Processing Letters*, 10(4):178–179, 1980.
- [43] Jean-Charles Faugere. A new efficient algorithm for computing gröbner bases ( $\{i\} \subseteq \{f_i/i\} \subseteq \text{sub}_i \subseteq 4_i/\text{sub}_i$ ). *Journal of pure and applied algebra*, 139(1):61–88, 1999.
- [44] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology EUROCRYPT 2000*, pages 392–407. Springer, 2000.
- [45] Pouyan Sepehrdad. *Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2012.
- [46] Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over  $\text{gf}(2)$  via sat-solvers. Cryptology ePrint Archive, Report 2007/024, 2007. <http://eprint.iacr.org/>.
- [47] G. Bard, N. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over  $\text{gf}(2)$  via sat-solvers. 2007.
- [48] Jean-Charles Faugere. A new efficient algorithm for computing gröbner bases ( $f \in 4$ ). *Journal of pure and applied algebra*, 139(1):61–88, 1999.
- [49] N. Courtois and G. Bard. Algebraic cryptanalysis of the data encryption standard. In *In IMA Int. Conf. volume 4887*, Springer, 2007.
- [50] Nicolas T Courtois, Gregory V Bard, and David Wagner. Algebraic and slide attacks on keeloq. In *Fast Software Encryption*, pages 97–115. Springer, 2008.



- [51] Nicolas T Courtois, Jerzy A Gawinecki, and Guangyan Song. Contradiction immunity and guess-then-determine attacks on gost. *Tatra Mountains Mathematical Publications*, 53(1):65–79, 2012.
- [52] Nicolas Courtois. Algebraic complexity reduction and cryptanalysis of gost. *IACR Cryptology ePrint Archive*, 2011:626, 2011.
- [53] Jeremy Erickson, Jintai Ding, and Chris Christensen. Algebraic cryptanalysis of sms4: Gröbner basis attack and sat attack compared. In *Information, Security and Cryptology–ICISC 2009*, pages 73–86. Springer, 2010.
- [54] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing–SAT 2009*, pages 244–257. Springer, 2009.
- [55] Nicolas T Courtois, Sean O’Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the hitag2 stream cipher. In *Information Security*, pages 167–176. Springer, 2009.
- [56] Nicolas Courtois, Karsten Nohl, and Sean O’Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.
- [57] N. Courtois, P. Sepehrdad, P. Susil, and S. Vaudenay. Elimlin algorithm revisited. In *Fast Software Encryption*, pp. 306–325, Springer Berlin Heidelberg, 2012.
- [58] J. Boyar, M. Find, and R. Peralta. Four measures of nonlinearity. In *Algorithms and Complexity*, pp. 61–72. Springer Berlin Heidelberg, 2013.
- [59] 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.

- [60] Gosudarstvennyi Standard GOST. 28147-89,. *Cryptographic protection for data processing systems, Government Committee of the USSR for Standards*, 1989.
- [61] Bruce Schneier. *Section 14.1 GOST (2nd ed.) Applied cryptography*. john wiley & sons, 1996.
- [62] 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/>.
- [63] Axel Poschmann, San Ling, and Huaxiong Wang. 256 bit standardized crypto for 650 ge–gost revisited. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 219–233. Springer, 2010.
- [64] Takanori Isobe. A single-key attack on the full gost block cipher. *Journal of cryptology*, 26(1):172–189, 2013.
- [65] Nicolas T Courtois. Security evaluation of gost 28147-89 in view of international standardisation. *Cryptologia*, 36(1):2–13, 2012.
- [66] Orhun Kara. Reflection cryptanalysis of some ciphers. In *Progress in Cryptology-INDOCRYPT 2008*, pages 294–307. Springer, 2008.
- [67] Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved attacks on full gost. In *Fast Software Encryption*, pages 9–28. Springer, 2012.
- [68] Nicolas T Courtois. Low-complexity key recovery attacks on gost block cipher. *Cryptologia*, 37(1):1–10, 2013.
- [69] Nicolas T Courtois and Michał Misztal. Aggregated differentials and cryptanalysis of pp-1 and gost. *Periodica Mathematica Hungarica*, 65(2):177–192, 2012.

- [70] Alex Biryukov and David Wagner. Advanced slide attacks. In *Advances in CryptologyEUROCRYPT 2000*, pages 589–606. Springer, 2000.
- [71] Bard Gregory V Courtois, Nicolas T and Andrey Bogdanov. Periodic ciphers with small blocks and cryptanalysis of keeloq. *Tatra Mt. Math. Publ*, 41:167–188, 2008.
- [72] Nicolas T Courtois and Blandine Debraize. Algebraic description and simultaneous linear approximations of addition in snow 2.0. In *Information and Communications Security*, pages 328–344. Springer, 2008.
- [73] 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.
- [74] Igor Semaev. Sparse algebraic equations over finite fields. *SIAM Journal on Computing*, 39(2):388–409, 2009.
- [75] Niklas Sörensson, Niklas Eén, and Mate Soos. Cryptominisat 2.92, an open-source sat solver package based on earlier minisat software.
- [76] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. 2010.
- [77] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis. 2000.
- [78] 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.
- [79] 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.

- [80] F. Standaert, G. Piret, G. Rouvroy, J. Quisquater, and J. Legat. Iceberg : An involutinal 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.
- [81] 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.
- [82] 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.
- [83] 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.
- [84] A. Bogdanov, L. Knudsen, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and Y. Vikkelsøe. Present: An ultra-lightweight block cipher. In *In the proceedings of CHES 2007*, 2007.
- [85] 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.
- [86] 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.

- [87] 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.
- [88] 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.
- [89] J. Boyar and R. Peralta. A depth-16 circuit for the aes s-box. 2011.
- [90] 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.
- [91] H. Alkhzaimi and M. Lauridsen. Differential and linear cryptanalysis of reduced-round simon. In *Cryptology ePrint Archive, Report 2013/543*, 2013.
- [92] N. Courtois. Basic equation solving toolbox - <http://www.cryptosystem.net/aes/tools.html>. 2010.
- [93] I. Semaev and M. Mikus. Methods to solve algebraic equations in cryptanalysis. In *In Tatra Mountains Mathematic Publications, Vol. 45, pp. 107-136*, 2010.
- [94] Ramachandran Balasubramanian and Neal Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the meneze-sokamotovanstone algorithm. *Journal of cryptology*, 11(2):141–145, 1998.
- [95] Standard specifications for public key cryptography annex a. pages 76–172. IEEE P1363 / D9, Feb 1999.
- [96] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

- [97] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985.
- [98] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management-part 1: General (revised). In *NIST special publication*. Citeseer, 2006.
- [99] Michiel Kusters and Sze Ling Yeo. Notes on summation polynomials. *arXiv preprint arXiv:1503.08001*, 2015.
- [100] Claus Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147(01):75–104, 2011.
- [101] Pierrick Gaudry. Index calculus for abelian varieties and the elliptic curve discrete logarithm problem. *IACR Cryptology ePrint Archive*, 2004:73, 2004.
- [102] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaél Renault. Using symmetries in the index calculus for elliptic curves discrete logarithm. *Journal of cryptology*, 27(4):595–635, 2014.
- [103] Jean-Charles Faugère, Ludovic Perret, Christophe Petit, and Guénaél 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.
- [104] Christophe Petit and Jean-Jacques Quisquater. On polynomial systems arising from a weil descent. In *Advances in Cryptology–ASIACRYPT 2012*, pages 451–466. Springer, 2012.
- [105] Yun-Ju Huang, Christophe Petit, Naoyuki Shinohara, and Tsuyoshi Takagi. Improvement of faugere et al.s method to solve ecdlp. In *Advances in Information and Computer Security*, pages 115–132. Springer, 2013.
- [106] Steven 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.

- [107] Nicolas Courtois. Half of all elliptic curves broken??? <http://blog.bettercrypto.com/?p=1544>, 2015.
- [108] Nicolas Courtois, Theodosios Mourouzis, Guangyan Song, Pouyan Sepehrdad, and Petr Susil. Combined algebraic and truncated differential cryptanalysis on reduced-round simon. In *SECRYPT*, pages 399–404, 2014.
- [109] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
- [110] Daniel J Bernstein and Tanja Lange. ebacs: Ecrypt benchmarking of cryptographic systems, 2009.
- [111] Arjen K Lenstra and Eric R Verheul. Selecting cryptographic key sizes. *Journal of cryptology*, 14(4):255–293, 2001.
- [112] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, 2001.
- [113] PUB FIPS. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 2000.
- [114] IEEE Working Group et al. Ieee 1363-2000: Standard specifications for public key cryptography. *IEEE Standard, IEEE, New York, NY*, 10017, 2000.
- [115] ANSI ANSI. X9. 62: 2005: Public key cryptography for the financial services industry. *The elliptic curve digital signature algorithm (ECDSA)*, 2005.
- [116] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.
- [117] SEC Certicom. Sec 2: Recommended elliptic curve domain parameters. *Proceeding of Standards for Efficient Cryptography, Version*, 1, 2000.

- [118] Ryan Castellucci. Cracking cryptocurrency brainwallets. <https://www.defcon.org/html/defcon-23/dc-23-index.html>.
- [119] Ernest F Brickell, Daniel M Gordon, Kevin S McCurley, and David B Wilson. Fast exponentiation with precomputation. In *Advances in CryptologyEUROCRYPT92*, pages 200–207. Springer, 1993.
- [120] Daniel J Bernstein and Tanja Lange. Explicit-formulas database, 2007.
- [121] Eric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography*, pages 335–345. Springer, 2002.
- [122] Daniel J Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In *Advances in cryptology–ASIACRYPT 2007*, pages 29–50. Springer, 2007.
- [123] Tanja Lange Daniel J. Bernstein. Explicit-formulas database. <https://www.hyperelliptic.org/EFD/>.
- [124] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology ASIACRYPT98*, pages 51–65. Springer, 1998.