5-2019

Analysis of a Group of Automorphisms of a Free Group as a Platform for Conjugacy-Based Group Cryptography

Pavel Shostak
The Graduate Center, City University of New York

Analysis of a group of automorphisms of a free group as a platform for conjugacy-based group cryptography

by

Pavel Shostak

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2019

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Professor Alexei Miasnikov

Date	Chair of Examining Committee

Professor Robert Haralick

Date	Executive Officer

Supervisory Committee:

Professor Alexei Miasnikov

Professor Robert Haralick

Professor Vladimir Shpilrain

Professor Alexander Ushakov

THE CITY UNIVERSITY OF NEW YORK

Abstract

ANALYSIS OF A GROUP OF AUTOMORPHISMS OF A FREE GROUP AS A PLATFORM FOR CONJUGACY-BASED GROUP CRYPTOGRAPHY

by

PAVEL SHOSTAK

Adviser: Professor Alexei Miasnikov

Let F be a finitely generated free group and $\mathrm{Aut}(F)$ its group of automorphisms. In this monograph we discuss potential uses of $Aut(F)$ in group-based cryptography. Our main focus is on using $\mathrm{Aut}(F)$ as a platform group for the Anshel-Anshel-Goldfeld protocol, Ko-Lee protocol, and other protocols based on different versions of the conjugacy search problem or decomposition problem, such as Shpilrain-Ushakov protocol. We attack the Anshel-Anshel-Goldfeld and Ko-Lee protocols by adapting the existing types of the length-based attack to the specifics of $Aut(F)$[1]. We also present our own version of the length-based attack that significantly increases the attack' success rate. After discussing the attacks, we discuss the ways to make keys from $Aut(F)$ resistant to different versions of the length-based attacks including our own.

---

[1] The CUNY HPCC is operated by the College of Staten Island and funded, in part, by grants from the City of New York, State of New York, CUNY Research Foundation, and National Science Foundation Grants CNS-0958379, CNS-0855217 and ACI 1126113.

Acknowledgments

I want to thank Professor Alexei Miasnikov for making it all possible.

Also, I want to thank my committee members Professor Vladimir Shpilrain and Professor Robert Haralick for their valuable comments and technical support.

My special thanks go to Professor Alexander Ushakov whose help, guidance and support went far beyond common advice and mathematical expertise.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Anshel-Anshel-Goldfeld protocol [Anshel et al., 1999] (subsequently called the AAG protocol) is one of the major protocols in group-based cryptography. The security of the AAG protocol was extensively analyzed for braid groups [Shpilrain and Ushakov, 2006b] [Hofheinz and Steinwandt, 2003] [Garber et al., 2006] [Miasnikov et al., 2005] [Miasnikov et al., 2006] [Myasnikov and Ushakov, 2007]. The AAG protocol used on the platform of braid groups was successfully attacked [Shpilrain and Ushakov, 2006b] using the length-based attack (subsequently called the LBA attack). However, the security of the AAG protocol was not sufficiently analyzed for different platform groups. Here we analyze the AAG protocol on the platform of $Aut(F)$, the group of automorphisms on a free group F. We want to evaluate the efficiency of the LBA attack in case of the $Aut(F)$ platform group.

Furthermore, the AAG protocol is based on the subgroup-restricted simultaneous conjugacy search problem. It means that if we show that the LBA attack is efficient for the AAG protocol, it also would be efficient against other protocols that rely on the conjugacy search problem including the Ko-Lee protocol and to an extent the Ushakov-Shpilrain protocol.

Also, we want to analyze the weaknesses and limitations of the LBA attack itself. It might allow us to generate LBA-resistant keys.

Chapter 2

History of classic public-key cryptography

The main purpose of computers is dealing with data so the need for efficient data encryption for safe data storage was there from the beginning of computers. However, the invention of networks and their subsequent rapid proliferation promoted efficient cryptography to one of the top priorities. The cryptography itself is nothing new but due to the specifics of networks all classic cryptography algorithms are inapplicable. All classic algorithms for encrypting and decrypting data imply that communicating parties privately agree a secret encryption/decryption key. It is crucial that key agreement is private and not observed by any third-party. By classic algorithms we mean algorithms developed in pre-networking era, all kinds of substitution and transposition ciphers that were known from ancient times.

Such private key agreement is impossible in networks where all communications are a potential target for eavesdropping. In open sources, the first solution appeared in 1976 when Whitfield Diffie and Martin Hellman published their paper [Diffie and Hellman, 1976] on public-key cryptography suggesting there a procedure that is known now as Diffie-Hellman key exchange protocol. Martin Hellman himself however suggested [Hellman, 2002] that the protocol should be called "Diffie-Hellman-Merkle" since it uses a public key distribution system, a concept developed by Merkle.

Diffie-Hellman key exchange protocol is based on the complexity of the discrete logarithm problem in the finite cyclic groups, that is recovering a from g and g^a. As it stands now, this problem does not have efficient solution for a good [Menezes et al., 1996] choice of parameters. In general, if function f is easy to compute but its f^{-1} is hard to compute then such f is called *one-way trapdoor function* or just *trapdoor function*. One-way trapdoor functions make the foundation of all public-key cryptography algorithms.

Diffie-Hellman key exchange protocol allows two communicating parties traditionally referred as Alice and Bob to establish a shared secret key in the presence of the eavesdropper traditionally referred as Eve. All communications between Alice and Bob are known to Eve; Eve knows the rules of the key exchange protocol as well but still Eve is not able to compute the shared secret key of Alice and Bob. Such key exchange protocol made encryption/decryption possible in networks where all communications are a potential target for eavesdropping.

Diffie-Hellman key exchange protocol

---

1. Alice and Bob choose a finite cyclic group G with a generating element g.
2. Alice picks a random natural number a and publishes g^a. a itself remains an Alice's secret key.
3. Bob picks a random natural number b and publishes g^b. b itself remains a Bob's secret key.
4. Alice computes $K_A = (g^b)^a = g^{ba}$
5. Bob computes $K_B = (g^a)^b = g^{ab}$

Since $ab = ba$, $K_A = K_B$. Thus, Alice and Bob posses the same shared secret key.

Later, in 1985, ElGamal suggested a cryptosystem [ElGamal, 1985] based on Diffie-Hellman key exchange protocol.

ElGamal cryptosystem

1. Alice and Bob choose a finite cyclic group G with a generating element g.
2. Alice picks a random natural number a and publishes g^a. a itself remains an Alice's secret key.
3. To send a message $m \in G$, Bob picks a random natural number b and sends $m' = m \cdot (g^a)^b$ and g^b to Alice.
4. Alice recovers $m = m' \cdot ((g^b)^a)^{-1}$

ElGamal encryption has a beneficial property of being *probabilistic* meaning that a single plaintext can be encrypted in many different cyphertexts. However, ElGamal is computationally intensive for Alice and Bob for the realistic parameters and the size of cyphertext is twice as large as plaintext. As a result, usually ElGamal is used not to encrypt the message itself but to establish symmetric shared secret key between Alice and Bob.

The original ElGamal cryptosystem later was found to be vulnerable to the chosen cyphertext attack. Later, the advanced versions of ElGamal cryptosystem were developed to fix that problem. One such cryptosystem based on ElGamal is Cramer-Shoup cryptosystem [Cramer and Shoup, 1998] suggested in 1998. The Cramer-Shoup cryptosystem is proven to be resistant to adaptive chosen cyphertext attack.

In 1978 Ron Rivest, Adi Shamir and Leonard Adleman suggested [R. Rivest, 1978] a cryptosystem that become known as RSA. Since then, RSA has become one of the major ways to establish private communications in the Internet.

RSA cryptosystem

1. Alice's private key is a pair p, q of large prime numbers.

2. Alice's public key consists of the product $n = pq$, an integer e such that $1 < e < \varphi(n)$, where $\varphi(n) = (p-1)(q-1)$ is the Euler function of n and e and $\varphi(n)$ are relatively prime.

3. Bob encrypts the message m which is an integer number $0 < m < n$ by computing $c \equiv m^e \ (mod\, n)$ and sends c to Alice.

4. In order to decrypt c, Alice finds an integer d such that $de \equiv 1 \ (mod \ \varphi(n))$ and computes $c^d \equiv (m^e)^d \equiv m^{ed} \ (mod\, n)$ Since $ed \equiv 1 + k\varphi(n)$,

$$m^{ed} \equiv m^{1+k\varphi(n)} \equiv m(m^k)^{\varphi(n)} \equiv m \ (mod\, n)$$

The security of RSA is based on the hardness of factorization problem. If one finds an efficient method to find p and q from $n = pq$ the RSA encryption will be broken. For example, the Pollards "p-1 algorithm" [Pollard, 1974] allows efficient factorization for certain specific cases which obviously should be avoided when choosing Alice's private key. Another promising approach is the Shor's factorization algorithm. But the Shor's algorithm is quantum and requires quantum computer to run.

Another classic public-key cryptography algorithm is Rabin's cryptosystem. It was the first algorithm where it was proven that recovering plaintext from the cyphertext is as hard as factoring.

Rabin's cryptosystem

1. Alice chooses the pair of large primes p, q, where $p \equiv q \equiv 3 \ (mod \ 4)$ and shares a public key $n = pq$.

2. Bob encrypts the message m which is an integer number $0 < m < n$ by computing $c \equiv m^2 \ (mod\, n)$ and sends c to Alice.

3. Alice computes the following values

$$m_p = c^{(p+1)/4} \; modp$$

and

$$m_q = c^{(q+1)/4} \; modq$$

Then she computes

$$\pm r = (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \; mod \; n$$

and

$$\pm s = (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \; mod \; n$$

where $y_p \cdot p + y_q \cdot q = 1$. $\pm r$ and $\pm s$ are square roots of c modulo p and modulo q, and therefore also modulo n.

All of the above classic public-key cryptography algorithms function on the groups of integers. The security of these algorithms is based on the complexity of different problems like factorization and the discrete logarithm problem for the groups of integers. However, there exist a number of quantum algorithms efficiently solving these problems and therefore breaking the cryptosystems based on them. As of now, quantum computers needed to run such algorithms barely exist, but the recent advances in that field [Knight, 2017] [Hsu, 2018] [Kelly, 2018] indicate the need for new cryptosystems resistant to quantum algorithms attacks. In fact, such a need was instantly obvious since at least 1997 when Peter Shor published [Shor, 1997] his quantum algorithm for integer factorization and discrete logarithm problem. Another prominent quantum algorithm is Grover's algorithm [Grover, 1996] that allows quadratic speedup for brute-force attacks and therefore sets a new standards for brute-force security for any future cryptosystems.

2.1 Quantum-resistant cryptography

There are a few different areas like lattice-based and multivariate cryptography from where such quantum-resistant cryptosystems appeared. We are interested in cryptography algorithms based on the non-commutative groups. These algorithms use hard group theory problems like the conjugacy search problem and the decomposition problem as their trapdoor functions. The notable examples of such cryptosystems are Anshel-Anshel-Goldfeld protocol [Anshel et al., 1999], Ko-Lee protocol [Ko et al., 2000] and Shpilrain-Ushakov protocol [Shpilrain and Ushakov, 2006a]. Those protocols will be defined later in the text.

Those cryptosystems can use the wide range of non-commutative groups as their base groups as long as there is no known efficient solution to the trapdoor function problem. One of the first candidates to such groups were braid groups. One way to define braid group is

$$B_n = \langle \sigma_1 \dots \sigma_{n-1} \mid \sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1},\ \sigma_i \sigma_j = \sigma_j \sigma_i \ where\ |i-j| \geq 2 \rangle.$$

There was an extensive analysis of braid groups as base groups for non-commutative group-based cryptography and as a result of this research the number of weaknesses were discovered and the number of attacks was suggested [F.Garside, 1969] [Birman et al., 2007] [Franco and González-Meneses, 2003] [Gebhardt, 2005] [Gebhardt, 2006] [E. El-Rifai, 1994] [V. Gebhardt, 2008] [Hofheinz and Steinwandt, 2003] [Maffre, 2005] [Maffre, 2006] [Myasnikov and Ushakov, 2007] [Longrigg and Ushakov, 2009] [Garber et al., 2006] [Garber et al., 2005] [Hughes, 2002] [Lee and Lee, 2002] [J. Cheon, 2003]. The key problem of braid groups is that they are linear [Bigelow, 2001]. For example, their linearity can be used to to solve group problems that make up the basis of AAG and Shpilrain-Ushakov protocols [Tsaban, 2013]. These finding are one of the reasons groups of automorphisms look interesting as a platform for group-based cryptography. On the one hand, groups of automorphisms are complex and do not have simple solutions for problems like conjugacy search problem. On the other hand, groups of automorphisms are not linear. Thus, they do not have the same vulnerabilities as braid groups even though braid groups are the subgroups of the groups of automorphisms.

Attempts were made to redeem braid groups for the group-based cryptography by introducing algebraic eraser [Anshel et al., 2006]. This protocol is also often called Anshel-Anshel-Goldfeld-Lemieux. Since then, a number of attacks was suggested [Myasnikov and Ushakov, 2009] [S. Blackburn, 2016]. After the first successful attacks, a few improvements to the original protocol were suggested [D. Atkins, 2016] [Gunnells, 2011]. Moreover, the technique of E-multiplication used in algebraic eraser is also used for the Kayawood key agreement protocol [I. Anshel, 2017a] and Walnut digital signature algorithm [I. Anshel, 2017b]. Walnut and Kayawood also have been studied and some strong attacks are suggested [D. Hart, 2017] [W. Beullens, 2018] [M. Kotov, 2018].

Other than braid groups, Thompson groups that can be presented as $\langle x_0, x_1 \ldots \mid x_i^{-1} x_k x_i = x_{i+1} \ where \ k > i \rangle$ were tried as base groups for group-based cryptography [Shpilrain and Ushakov, 2005] but also were found to be insecure [Ruinsky et al., 2007] [Matucci, 2008].

Yet another type of groups that look appealing is polycyclic groups. Polycyclic groups can be defined as follows

$$\langle a_1, \ldots, a_n \mid a_j^{a_i} = w_{ij},\ a_j^{a_i^{-1}} = v_{ij},\ a_k^{r_k} = u_{kk},\ \text{for } 1 \le i < j \le n,\ k \in I \rangle,$$

where $I \subseteq \{1, \ldots, n\}$ and $r_i \in \mathbb{N}$ of $i \in I$ and $w_{ij},\ v_{ij},\ u_{jj}$ are words in generators

$a_{j+1}, \ldots, a_n$. Polycyclic groups were analyzed as a potential platform group for group-based cryptography [Eick and Kahrobaei,]. Some properties of polycyclic groups (easily solvable word problem and hard conjugacy search and decomposition problems) were shown [Garber et al., 2015]. But eventually ways to attack cryptosystems based on polycyclic groups were developed [Kotov and Ushakov,]. The main disadvantage of polycyclic groups is the fact that they are linear, which proved to be a weakness for some classes of groups like braid groups.

As mentioned above, groups of automorphism of a free group have some nice properties as complex group behaviour and no known solutions for problems like conjugacy search problem and decomposition problem. For some time, this group remained impractical because the only known normal form of its elements is so massive that no practical usage was possible. Dealing with normal forms of automorphisms comes down to dealing with very long words. So there was a need to find a way to store, compare and reduce such very long words.

One possible way to solve this problem appeared from the compression techniques. The idea is to compress very long words to a very high degree (exponential compression) and perform all operations on them without fully decompressing them. Straight-line programs (abbreviated as SLP) allow such level of compression. In a nutshell, SLP is a context-free grammar that generates a single word. See Section 4 for more detail.

The next challenge was to develop algorithms to deal with words compressed using SLP right in compressed for without fully decompressing them. Such techniques were proposed for example in [Schleimer, 2008] [Plandowski, 1994].

Chapter 3

Preliminaries to group theory

In this section, we lay down the the basic group theory concepts that we use in our work.

3.1 Finitely presented groups

Definition 3.1.1. *A* group *is a pair $(G, \cdot)$, where G is a set and $\cdot$ is a binary operation on G satisfying the following conditions:*

1. *For every two elements $a, b \in G$ there exists a single element $c \in G$ such that $a \cdot b = c$. Usually we omit the $\cdot$ symbol and write simply $ab = c$.*

2. *There exists an element $1 \in G$ such that for any $a \in G$*

$$a \cdot 1 = 1 \cdot a = a.$$

   1 *is called the* identity *element.*

3. *For every element $a \in G$ there exists an element $a^{-1} \in G$ satisfying*

$$aa^{-1} = a^{-1}a = 1,$$

called the inverse *of a.*

4. *The operation $\cdot$ is* associative, *i.e., for every $a, b, c \in G$*

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

A subset $H \subseteq G$ is called a *subgroup* of G if $(H, \cdot)$ is a group itself. For a set $A \subseteq G$ define a set $\langle A \rangle$ of all products of elements from A and their inverses. It is easy to check that $\langle A \rangle$ is a subgroup of G, called the *subgroup generated* by A. If $G = \langle A \rangle$ for some $A \subseteq G$, then we say that G is generated by A or that A is a *generating set* for G, elements of A are called *generators*. We often refer to a product of generators A and their inverses as a *word in generators A*.

If the element $r \in G$ defines the identity element $r = 1$ in group G we call r the *relator of G*. Since by definition every element of group G has an inverse, every generator a of the group G produces a relator aa^{-1}. Such relators are called *trivial relators* and every group has them.

We say that the element $w \in G$ is *derivable* from the set of relators $R = \{r_i\}$ if the element w can be turned into identity by applying the number of the following operations:

- Removing relator r_i or a trivial relator from w if it forms the consecutive block of symbols in w.
- Inserting relator r_i or a trivial relator in any position in w.

If any relator of group G can be derived from the set of relators R, we call such set *a set of defining relators* or *a complete set of relators*.

Any group G can be defined as a pair $\langle A; R \rangle$ where A is the set of generators and R is the set of defining relators. For more information see [Magnus et al., 1976] and [Lyndon and Schupp, 2001]. We call such a pair *a presentation of the group G*.

If the set A of generators of a group G is finite, we call the group G *finitely generated*. If the set of the defining relators of G is also finite, we call the group G *finitely presented*.

3.2 Dehn problems

Dehn problems are the fundamental problems of group theory, introduced by Max Dehn [Dehn, 1911] in 1911. They include the word problem, the conjugacy problem and the isomorphism problem. Below we present the definitions of those problems

Word problem in G. Given a word $w \in G$ decide if $w =_G 1$.

Conjugacy problem. Given two words $w_1, w_2 \in G$ decide if there exists $w_3 \in G$ such that $w_1 = w_3 w_2 w_3^{-1}$.

Isomorphism problem. Given the presentations for groups G_1 and G_2 decide whether G_1 and G_2 are isomorphic.

The one way to solve the word problem is to have a normal form for words in G. Normal form $norm()$ is such a way to present words from G that for two different presentations w_1 and w_2 of the same word $norm(w_1) = norm(w_2)$. Having an easy-to-compute normal form is also important for the AAG protocol because Alice and Bob should be able to compute the shared secret key in the same form.

3.3 Group of automorphisms of the free group

We call group G a free group if there exists a subset $X \subset G$ such that every element of G can be uniquely (up to trivial combinations like $xx^{-1} = \varepsilon$) presented as a product of the finite number of elements from X. We call the set X a free basis. We denote F_r or simply F a free group with alphabet $X = \{x_1, ..., x_r\}$. Set X is a generating set of the group G and its elements $x_i \in X$ are generators.

A homomorphism from a group G to group H is a map $\varphi : G \to H$ satisfying:

$$\varphi(g_1 g_2) = \varphi(g_1)\varphi(g_2)$$

for every $g_1, g_2 \in G$. An *automorphism* of G is a bijective homomorphism $\varphi : G \to G$. The set of automorphisms of the group G is denoted by $Aut(G)$ and forms a group $(Aut(G), \circ)$ under composition, called the group of automorphisms of G.

In our research, we opt to understand the product of automorphisms the following way: for $a_1, a_2 \in Aut(F)$ and a word $w \in F$ $(a_1 a_2)(w) = a_2(a_1(w))$.

Let F be a free group of rank r on the alphabet $X = \{x_1, \ldots, x_r\}$. The set of all automorphisms of the free group $F(X)$ (subsequently called group F) makes up a group $Aut(F)$. We use Nielsen transformations as a generating set of $Aut(F)$. Nielsen transformations can be defined as follows:

Definition 3.3.1. *For a free group F on the alphabet $X = \{x_1, \ldots, x_r\}$ Nielsen transformations are all transformations of the following form:*

1. $x_i \to x_i^{-1}$ *where* $1 \leq i \leq r$
2. $x_i \to x_j$ *and* $x_j \to x_i$ *where* $1 \leq i, j \leq r$
3. $x_i \to x_i x_j$ *where* $1 \leq i, j \leq r$ *and* $i \neq j$
4. $x_i \to x_j^{-1} x_i$ *where* $1 \leq i, j \leq r$ *and* $i \neq j$
5. $x_i \to x_j^{-1} x_i x_j$ *where* $1 \leq i, j \leq r$ *and* $i \neq j$

Nielsen transformations were introduced in [Nielsen, 1921] and a few years later in [Nielsen, 1924] it was proven that the set of Nielsen transformations is the generating set of $Aut(F)$. These results are also presented in [Magnus et al., 1976] in English. Since any automorphism of a free group can be presented as a sequence of Nielsen transformations we can use Nielsen transformations to generate random automorphisms. We will present an algorithm to do so later in this paper.

Nielsen transformations are a convenient tool for purposes like automorphism generation. However, there is another tool worth paying attention to, namely Whitehead automorphisms.

Definition 3.3.2. *Whitehead automorphisms are the automorphisms of the following form:*

1. $x_i \to x_i^{-1}$ *where* $1 \leq i \leq r$
2. $x_i \to x_j$ *and* $x_j \to x_i$ *where* $1 \leq i, j \leq r$
3. *for a fixed* $a \in X^{\pm}$ *every* $x \in X^{\pm}, x \neq a^{\pm}$ *is mapped to one of the elements* $x, xa, a^{-1}x, a^{-1}xa$

By $W(X)$ we denote the set of Whitehead automorphisms of type 3. In [Whitehead, 1936] Whitehead solves the minimization problem for the words in a free group.

Definition 3.3.3. *We say that the word* $w \in F$ *is minimal if there exists no automorphism* ϕ *such that* $|w| > |\phi(w)|$.

Definition 3.3.4. *The automorphic orbit* Orb(w) *of a word* $w \in F$ *is the set of all automorphic images of* w *in* F*:*

$$\mathrm{Orb}(w) = \{v \in F \mid \exists \phi \in Aut(F) \text{ such that } \phi(w) = v\}.$$

Minimization problem requires to find for a given word $w_1 \in G$ such an automorphism ϕ that the word $w_2 = \phi(w_1)$ is minimal.

In [Whitehead, 1936] Whitehead proves the following theorem

Theorem 3.3.5 (Whitehead theorem)**.** *Let* $w_1, w_2 \in F(X)$ *and* $w_2 \in \mathrm{Orb}(w_1)$*. If* $|w_1| > |w_2|$*, then there exists* $t \in W(X)$ *such that* $|w_1| > |t(w_1)|$.

The Whitehead theorem allows to construct the minimization procedure that can be used to disassemble the automorphism in normal form into a sequence of Whitehead automorphisms. Later, the sequence of Whitehead automophisms can be converted into a sequence of Nielsen transformations.

However, there is a practical problem with Whitehead automorphisms - there are too many of them for any realistically large rank of a free group F. As a result, we prefer to use the similar result by Nielsen for the purpose of converting of an automorphism into a sequence of Nielsen transformations.

Definition 3.3.6. *Let W_i be freely-reduced words on alphabet $X = \{x_j\}$. The set of words $\{W_i\}$ is called Nielsen reduced if the following conditions are met for every word $V(W_i)$ in symbols W_i:*

- *Each W-symbol occurring in the word $V(W_i)$ contributes at least one x-symbol to the freely-reduced form of $V(W_i(x_j))$.*
- *The number of x-symbols in $V(W_i(x_j))$ is at least as large as the number of x-symbols in any W_i occurring in $V(W_i)$.*

Theorem 3.3.7. *Let $W = \{W_1...W_m\}$ be a finite m-tuple of freely reduced words in the free generators $\{x_j\}$. Then we can find a sequence $\tau_1...\tau_k$ of Nielsen transformations of rank m such that:*

$$|W| \leq |\tau_1 W| \leq ... \leq |\tau_k...\tau_1 W|$$

and

$$\tau_k...\tau_1 W = \{\tilde{W}_1...\tilde{W}_m\}$$

where $\{\tilde{W}_1...\tilde{W}_t\}$ and $\tilde{W}_{t+1} = 1,\ ...\ ,\tilde{W}_m = 1$

This theorem also allows to disassemble the automorphism in normal form into a sequence of Nielsen transformations. However, the number of Nielsen transformations is much smaller than the number of Whitehead automorphisms for the same rank and that allows faster conversions.

Presenting a given automorphism as a sequence of Nielsen transformations has a certain security advantages that will be discussed further.

3.4 Length of an automorphism

In our work, we use a length-based attack to analyze the security properties of $Aut(F)$. As it follows from its very name, we need length measurement for the length-based attack. In this section we define the length function for automorphisms. Initially, it feels natural to define the length of an automorphism as a minimum number of Nielsens transformations that form it. But such length measurement requires either storing the automorphism as a sequence of Nielsen transformations or having an algorithm to quickly convert the automorphism to that form. Both possibilities are discussed in Section 10.2 and they turn out to be complicated. So we opt for a different definition of length. We define the length of an automorphism ϕ as a sum of lengths of all reduced images of all the generators of the free group F. Further, we will analyze the properties of this length measure and show that it fits the length-based attack well.

$$|\phi| = \sum_{i=1}^{R} |\phi(x_i)|,$$

where x_i are the generators of group F.

In our paper we keep calling the suggested measure *length*, but strictly speaking it is not a length. The proposed measure lacks additivity. Recall that a length function is an additive function which means that $f(gh) = f(g) + f(h)$ for any g and h in the function' domain. Our function does not satisfy this condition. For our measure, it is possible that for $g, h \in Aut(F)$ the additive property is violated and $|gh| \neq |g| + |h|$. It might be more correct to call this measure *size*, but since we use it for *length*-based attack we keep calling it length for the sake of uniformity. Even though the suggested measure does not have the additive property, it suits our needs well. It is relatively easy to compute and it preserves the crucial property of the length of an automorphism usually increasing when multiplied by another automorphism. In Section 7.1 we discuss why this property is important and in Section 7.2 we discuss situations when the length-increasing property of the suggested measurement does not hold.

3.5 Problems in the group of automorphisms

The $Aut(F)$ group can be viewed as a promising base group for the AAG protocol because its word problem can be efficiently solved but there is no known way to solve the conjugacy search problem for $Aut(F)$ efficiently. The basic version of the conjugacy search problem can be stated as follows

Conjugacy search problem in G. Given two elements $w_1, w_2 \in G$ find an element $x \in G$ such that $w_2 = x^{-1} w_1 x$.

As we already mentioned above there is no efficient solution to conjugacy problem in $Aut(F)$. But it is not enough to solve conjugacy problem to break the AAG protocol. To do it we need to solve an even more complex version of the conjugacy problem, namely subgroup-restricted simultaneous conjugacy problem or two different simultaneous conjugacy search problems. This topic is discussed in [Shpilrain and Ushakov, 2006b].

Simultaneous conjugacy search problem. Given tuples $\bar{g} = \{g_1, ..., g_k\}$ and $\bar{h} = \{h_1, ..., h_k\}$ of elements of G, find an element $x \in G$ such that $\bar{g} = x^{-1}\bar{h}x$, i.e., solve the following system of conjugacy equations

$$\begin{cases} g_1 = x^{-1}h_1x \\ \quad\cdots \\ g_k = x^{-1}h_kx \end{cases}$$

Subgroup-restricted simultaneous conjugacy search problem in G. Given a subgroup A generated by the set $\bar{a}$, tuples $\bar{g} = \{g_1, ..., g_k\}$ and $\bar{h} = \{h_1, ..., h_k\}$ of elements of G, find an element $x \in A$ as a product of elements of $\bar{a}$ such that $\bar{g} = x^{-1}\bar{h}x$.

On the other hand, the word problem in $Aut(F)$ has a solution. In case of the $Aut(F)$ group, the normal form of an automorphism $\phi \in Aut(F)$ can be obtained by calculating the images $\phi(x)$ of all letters x of the alphabet X.

Definition 3.5.1. *For an automorphism $\phi \in Aut(F)$ we define a normal form as $norm(\phi) = \{\phi(x_1), ..., \phi(x_R)\}$ where $X = \{x_1, ..., x_R\}$ is the alphabet of X.*

This definition of normal form leads us to a natural way to measure an automorphism's length.

Definition 3.5.2. *The length of an automorphism $\phi \in Aut(F)$ is $|\phi| = \sum_{x \in X} |\phi(x)|$ where X is the alphabet.*

Such a normal form and length measure seem very fitting and convenient but they have the potential problem. The problem is that for realistically complex automorphisms the images of the letters of X are very long. If we multiply an automorphism by the Nielsen transformations we should expect the exponential growth of its length. For example, in our experiments we were constantly dealing with automorphisms whose images of the letters of X had the lengths of an order of 10^{28}. Thus, the lengths of the images of X quickly get so large that no straight-forward brute-force approach can be feasible even on the most powerful computers. We require special approaches and algorithms to work with images of X. Such special approaches and algorithms can be found in the field of straight-line programs or simply SLPs. On the one hand, SLPs allow us to store images of X in a heavily compressed form. On the other hand, SLPs allow us to perform operations on these images right in compressed form without decompressing them.

3.6 Random automorphisms generation

As discussed above, any automorphism from $Aut(F)$ can be presented as a sequence of Nielsen transformations. We use this fact to generate random automorphisms. The algorithm to do so is

Algorithm *Generation of a random automorphism*

1. Set the number N of the Nielsen transformations that will make up the random automorphism.
2. Choose the random Nielsen transformation of type 3 or 4.
3. Choose the random Nielsen transformation of type 3 or 4. Check for possible cancellations with the already existing sequence. If nothing cancels out, add this Nielsen transformation to the sequence. Repeat this step $N-1$ times.
4. Choose a random integer number i from 1 to $2R$ where R is the size of the alphabet X. If $i \leq R$ add $x_i \to x_i^{-1}$ to the sequence of Nielsen transformations.

The method described above has some potential issues with the automorphism's length.

We defined the length of an automorphism as the sum of lengths of images of X. The above algorithm does not allow us to generate an automorphism of the given length. Indeed, the same number of Nielsen transformations can generate automorphisms of different lengths.

In practice we need this algorithm of a random automorphism generation to generate the public sets of Alice and Bob. The elements of those sets are small so it is possible to estimate their length. Our observations show that the algorithm performed sufficiently well and produced the automorphisms of the expected length.

Chapter 4

Introduction to straight-line programs

In this section, we present straight-line programs (or SLPs for short). Straight-line programs are our main tool for presenting automorphisms in the computations. The efficiency of SLPs immediately affects the efficiency of the automorphisms in our experiments.

4.1 Straight-line programs

Definition 4.1.1. *Straight-line program (SLP) over a terminal alphabet* $X = \{x_1, ..., x_R\}$ *is a context-free grammar* $G = \{V, X, S, P\}$ *where* V *is the set of nonterminal symbols,* X *is the set of terminal symbols,* $S \in V$ *is the starting nonterminal symbol, and* P *is the set of production rules where each nonterminal symbol* $V_i \in V$ *has exactly one production rule of one of the following forms:*

- $V_i \to x$ *where* $x \in X$.
- $V_i \to V_j V_k$ *where* $V_j, V_k \in V$ *and* $j, k < i$.

Every straight-line program describes a single word $w(S)$. We can define the word produced by the nonterminal symbol $v \in V$ the following way:

$$w(v) = \begin{cases} \epsilon & if\ P(v) = \epsilon \\ x & if\ P(v) = x \in X \\ w(A)w(B) & if\ P(v) = AB \end{cases}$$

The length of this word can grow exponentially when we increase the number of production rules. It allows us to use SLPs representing words in a highly compressed form.

The depth of an SLP $\mathcal{P}$ is $depth(S)$. The depth of a terminal symbol $v \in V$ is defined as follows

$$depth(v) = \begin{cases} 1 & if\ P(v) = x \in X \cup \{\epsilon\} \\ 1 + max(w(A), w(B)) & if\ P(v) = AB \end{cases}$$

4.2 Operations on SLPs

In this section we mention the existence and complexities of the important algorithms that deal with SLPs. These algorithms make the operation on automorphisms presented as SLPs possible. All these algorithms are nicely described in [Miasnikov et al., 2011].

Word segments algorithms. Firstly, there is an algorithm that takes an input of an SLP $\mathcal{P}$ and an integer number $\alpha \in [1, |w(\mathcal{P})|]$ and returns an SLP $\mathcal{P}'$ such that $w(\mathcal{P}') = w(\mathcal{P})[: \alpha]$. Symmetrically, there is an algorithm that takes an input of an SLP $\mathcal{P}$ and an integer number $\alpha \in [1, |w(\mathcal{P})|]$ and returns an SLP $\mathcal{P}'$ such that $w(\mathcal{P}') = w(\mathcal{P})[\alpha :]$. These algorithms allow specifying the initial and terminal segments correspondingly of a word in form of SLP. It is important to mention that the resulting word is also presented in form of SLP and all manipulations are carried out on SLP without any decompression.

The above algorithms for obtaining the initial and terminal segments of an SLP-presented word can be combined to get the algorithm that produces the SLP representing a middle segment of the word. This algorithm takes an input of an SLP $\mathcal{P}$ and an integer numbers

$0 \geq \alpha \geq \beta \geq |w(\mathcal{P})|$ and returns an SLP $\mathcal{P}'$ such that $w(\mathcal{P}') = w(\mathcal{P})[\alpha : \beta]$.

All three algorithms presented in this section have a good complexity $O(depth(\mathcal{P}))$.

Word inversion. Simple algorithm allows taking an SLP $\mathcal{P}$ that defines a word $w(\mathcal{P})$ and returning an SLP $\mathcal{P}'$ which defines a word $w(\mathcal{P}') = (w(\mathcal{P}))^{-1}$. The only thing this algorithm does is to change all original production rules of $\mathcal{P}$ according to the following rule

$$P'(v') = \begin{cases} x^{-1} & if\ P(v) = x \in X \cup \{\epsilon\} \\ BA & if\ P(v) = AB \end{cases}$$

The word inversion algorithm has a complexity $O(|\mathcal{P}|)$.

4.3 Automorphisms presented in form of SLPs

In the case of automophisms, we can use SLPs to present the images of letters of X in compressed form. Since SLPs can potentially provide us with exponential compression it becomes possible to store the above-defined normal forms of automorphisms in computer memory.

One can argue that SLPs are not guaranteed to provide exponential compression. Indeed, the SLP presentation of a random word from X^* can provide no significant compression. But in case of automorphisms we deal not with random words in X^* but with words generated by the automorphisms from letters of X. Each automorphism can be presented as the sequence of Nielsen transformations. Nielsen transformations are very similar to production rules of SLPs. The starting non-terminal symbol S of the SPL is similar to the letter $x \in X$ that is mapped to the word represented by the SLP. Hence, we can expect that the number of production rules of the SLP that represents an image of a certain letter $x \in X$ is similar to the number of Nielsen transformations that make up the automorphism. Of cause, there is no strictly defined correlation between them, but the high level of compression is to be expected.

However, just storing automorphisms in computer memory is not enough. We need to perform basic operations on automorphisms such as comparing two automorphisms, multiplying them, performing free reduction of an automorphism, computing the automorphisms length. Since the uncompressed forms of automorphisms can be huge, we cannot deal with the uncompressed forms. We need algorithms that allow dealing with automorphisms in the compressed form. Because we present an automorphism $\phi \in Aut(F)$ as a set of images $norm(\phi) = \{\phi(x_1), ...\phi(x_R)\}$, the problem of dealing with automorphisms in a compressed form is equivalent to the problem of dealing with words from X^* in a compressed form. The techniques that allow doing so (like the technique to reduce the word in the compressed form) were described in a number of papers, for example [Schleimer, 2008] [Lohrey, 2012] [Jez, 2015] [Lifshits, 2007].

For our experiments we used a C++ library written by Dmitry Panteleev and Pavel Morar. This library provides classes for efficient storage and manipulations on automorphisms. The library is based primarily on the algorithms described in [Lifshits, 2007].

Chapter 5

Complexity of operations on the automorphisms presented in the form of the SLPs

In our complexity evaluations, we care mostly for reductions and multiplications (or rather conjugations that is two multiplications at a time). The complexity of reduction operation is discussed in [Miasnikov et al., 2011]. It it proven there that the complexity of reduction of a word W presented as SLP $\mathcal{P}$ is $O(|\mathcal{P}|^7)$ where $|\mathcal{P}|$ is the number of production rules in $\mathcal{P}$. This is a high complexity and later we present the actual measurements of how expensive reduction is in practice.

Another operation on automorphisms that is theoretically complex is equality test. We check the equality of two automorphisms by checking if the images of the letters from X match. Since each image is presented in the SLP form, one needs Plandowski's algorithm [Plandowski, 1994] that decides if two SLPs $\mathcal{P}_A$ and $\mathcal{P}_B$ define the same word. The complexity of the Plandowski's algorithm is $O((|\mathcal{P}_A|+|\mathcal{P}_B|)^5)$ where $|\mathcal{P}_A|$ is the number of production rules in $\mathcal{P}_A$ and $|\mathcal{P}_B|$ is the number of production rules in $\mathcal{P}_B$. However, Plandowski's algorithm considers reduced and non-reduced versions of the same word to be different. Thus, to use it, we need to reduce both words presented by the SLPs $\mathcal{P}_A$ and $\mathcal{P}_B$. Considering the complexity of reduction discussed above, we get the following total cost of comparison $O((|\mathcal{P}_A| + |\mathcal{P}_B|)^5 + |\mathcal{P}_A|^7 + |\mathcal{P}_B|^7) = O(|\mathcal{P}_A|^7 + |\mathcal{P}_B|^7)$.

Checking the equality of two words presented in an SLP form looks like a complex procedure. However, in practice it is possible to do the set of simple checks that gives us the answer fast. For example, before applying the Plandowski's algorithm we can just compare the length of the images of letters of X and if these lengths are different we have the negative answer fast. After all, the equality checks can be fast most of the time especially for SLPs presenting the already reduced words. One more observation, LBA attacks require minimal amounts of equality checks.

The operation that LBA attacks require a lot is the length measurement. Fortunately, funding the length of the word W presented by the SLP $\mathcal{P}$ is a simple straightforward task of linear complexity $O(|\mathcal{P}|)$. It requires one addition operation per each nonterminal symbol.

Algorithm *Computing the length of the word W described by the SLP $\mathcal{P}$*

1. Determine the length of the image of each nonterminal symbol V_i starting from $i = 0$ following the rule:
    - if the production rule for V_i is of the form $V_i \to x$ then the length of the image $|V_i| = 1$.
    - if the production rule for V_i is of the form $V_i \to V_j V_k$ then the length of the image $|V_i| = |V_j| + |V_k|$
2. The length of the image of the last nonterminal symbol is the length of the word W described by $\mathcal{P}$.

It is worth mentioning that since the lengths of nonterminal symbols tend to grow exponentially, the additions in the above algorithm require long integer arithmetic and are not as fast as a usual integer addition.

The next important operation on automorphisms is inversion. There is no known algorithm that allows inverting an automorphism stored in the form of SLP. We solve this problem by keeping the inverted form of the automorphism together with its original form. It makes the inverse of the automorphism instantly available but significantly increases the duration of other operations like multiplication and assignment. Also, it increases the memory consumption of our program. The inverse of a new automorphism is obtained during its generation since we generate a new automorphism by multiplying a number of Nielsen transformations and we can easily invert the sequence of Nielsen transformations.

Another operation that is very important for LBA attacks is multiplication of two automorphisms. In theory, this operation is simple $O(|X|^2)$ where $|X|$ is the number of terminal symbols. However, in practice the particular implementation that we are dealing with requires a significant amount of memory-related operations. Later we will provide the actual measurements of how fast the multiplication of two automorphisms is in the library that we are using.

Chapter 6

Protocols of group-based cryptography

Here are the examples of some typical protocols of group-based cryptography. The main interest of our research is the Anshel-Anshel-Goldfeld protocol but the results are relevant for all conjugacy-search-based and decomposition-based protocols.

6.1 Protocol description

Let $Aut(F)$ be a group of all automorphisms of a free group F of rank R. The set of Nielsen transformations $\{Nil_i\}$ is the generating set of $Aut(F)$. We implement the Anshel-Anshel-Goldfeld protocol on the platform group $Aut(F)$. Protocol steps are:

Anshel-Anshel-Goldfeld protocol

- Alice randomly generates N_1-tuple of words from F

$$\bar{a} = \{a_1, \ldots, a_{N_1}\}$$

Tuple $\bar{a}$ is called *Alice's public set*. The number of Nielsen transformations used to generate each element of this tuple is L_A.

- Bob randomly generates N_2-tuple of words from F

  $$\bar{b} = \{b_1, \ldots, b_{N_2}\}$$

  Tuple $\bar{b}$ is called *Bob's public set.* The number of Nielsen transformations used to generate each element of this tuple is L_B.

- Alice randomly generates a product $A = a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \ldots a_{s_L}^{\varepsilon_L}$, where $0 < s_i \leq N_1$ and $\varepsilon_i = \pm 1$ for each $1 \leq i \leq L$. The word A is called *Alice's private key.*

- Bob randomly generates a product $B = b_{t_1}^{\delta_1} b_{t_2}^{\delta_2} \ldots b_{t_L}^{\delta_L}$, where $0 < t_i \leq N_2$ and $\delta_i = \pm 1$ for each $1 \leq i \leq L$. The word B is called *Bob's private key.*

- For all $1 \leq i \leq N_2$ Alice computes $b_i' = A^{-1} b_i A$ and makes them reduced. Then Alice transmits all b_i' to Bob.

- For all $1 \leq i \leq N_1$ Bob computes $a_i' = B^{-1} a_i B$ and makes them reduced. Then Bob transmits all a_i' to Alice.

- Alice computes $K_A = A^{-1} a_{s_1}'^{\varepsilon_1} a_{s_2}'^{\varepsilon_2} \ldots a_{s_L}'^{\varepsilon_L} = A^{-1} B^{-1} A B$.

- Bob computes $K_B = b_{t_L}'^{-\delta_L} \ldots b_{t_2}'^{-\delta_2} b_{t_1}'^{-\delta_1} B = A^{-1} B^{-1} A B$.

  One can see that $K_A = K_B$ so $K = K_A = K_B$ is now a *shared secret key* of Alice and Bob.

The AAG protocol itself does not require any specific techniques of Alice's and Bob's private keys generation. And it means that generating Alice's secret key A for example can be as simple as generating a random sequence of s_L elements of $\bar{a}$ and doing $s_L - 1$ automorphism multiplications. Of course, when generating the random sequence we must make sure that adjacent elements of this sequence does not cancel each other out. Strictly speaking, things can get more complicated if a lot of elements of $\bar{a}$ commute with each other but controlling the length of A is still easy.

In order to obtain the shared secret key, the attacker must obtain the Alice's private key A and the Bob's private key B. This will allow constructing the shared private key $K = A^{-1}B^{-1}AB$. To do it, the attacker has to solve the simultaneous conjugacy search problem for sets $\bar{a}$ and $\bar{a'}$, and the simultaneous conjugacy search problem for sets $\bar{b}$ and $\bar{b'}$. The shared secret key can be computed from the know Alice's private key A alone by computing $K = K_A = A^{-1}a'^{\varepsilon_1}_{s_1}a'^{\varepsilon_2}_{s_2}\dots a'^{\varepsilon_L}_{s_L} = A^{-1}B^{-1}AB$. But to do it the attacker needs to know the Alice's private key as a word in $\bar{a}$, $A = a^{\varepsilon_1}_{s_1}a^{\varepsilon_2}_{s_2}\dots a^{\varepsilon_L}_{s_L}$. Symmetrically, the shared secret key can be obtained from knowing $B = b^{\delta_1}_{t_1}b^{\delta_2}_{t_2}\dots b^{\delta_L}_{t_L}$ as a word in $\bar{b}$. Thus, the attacker needs to solve only one conjugacy search problem, but since now the private key should be found as a product of given generators, the attacker should solve a different version of conjugacy search problem, namely the subgroup-restricted simultaneous conjugacy search problem.

The AAG protocol can be used on different platform groups. For example, [Myasnikov and Ushakov, 2007] analyses the AAG protocol used with braid groups. In this paper, we analyze the AAG properties when used on the platform of $Aut(F)$.

However, the LBA that we will describe here can be used to attack other encryption protocols that use the complexity of conjugacy search problem or even decomposition problem. We can mention just two such protocols - Ko-Lee [Ko et al., 2000] and Ushakov-Shpilrain [Shpilrain and Ushakov, 2006a].

Ko-Lee protocol

- Alice and Bob agree on publicly visible group G and two of its subgroups $A \subset G$ and $B \subset G$ such that for any $x \in A$ and $y \in B$ $xy = yx$. Also Alice and Bob agree on a word $w \in G$.

- Alice chooses her private key $a \in A$.
- Bob chooses his private key $b \in B$.
- Alice computes a normal form of $a^{-1}wa$ and sends it to Bob.
- Bob computes a normal form of $b^{-1}wb$ and sends it to Alice.
- Alice computes a normal form of $K_a = a^{-1}(b^{-1}wb)a$.
- Bob computes a normal form of $K_b = b^{-1}(a^{-1}wa)b$.

  Since subgroups A and B commute, $ab = ba$ so $K_a = K_b = K$ where K is the shared secret key.

The attack on the Ko-Lee protocol is similar to the attack on AAG. You just consider the set $\bar{b}'$ to contain the single element $\bar{b}' = \{a^{-1}wa\}$, $\bar{b}$ to contain the single element $\bar{b} = \{w\}$ and the set $\bar{a}$ to contain the generating elements of the subgroup A from the Ko-Lee protocol. Furthermore, the attack is simplified by the fact that it is enough to solve the basic conjugacy search problem. Indeed, if we find the Alice's private key a in any generators by solving the conjugacy search problem for the word w and the publicly available $a^{-1}wa$, we can compute the shared secret key $K = a^{-1}(b^{-1}wb)a$ because $b^{-1}wb$ is also public. Symmetrically, the same applies to finding the Bob's private key b from w and $b^{-1}wb$.

The Ushakov-Shpilrain protocol deals with the decomposition problem rather than conjugacy problem so LBA should be less efficient. Still LBA attack can be attempted on it. Here is the Ushakov-Shpilrain protocol itself.

Ushakov-Shpilrain protocol

- Alice and Bob publicly agree on a word $w \in G$.
- Alice chooses an element $a_1 \in G$, chooses a subgroup of $C_G(a_1)$, and publishes its generators $A = \{\alpha_1, ..., \alpha_k\}$.

- Bob chooses an element $b_2 \in G$, chooses a subgroup of $C_G(b_2)$, and publishes its generators $B = \{\beta_1, ..., \beta_m\}$.
- Alice chooses an element a_2 in $\langle \beta_1, ..., \beta_m \rangle$ and sends the normal form $P_A = a_1 w a_2$ to Bob.
- Bob chooses an element b_1 in $\langle \alpha_1, ..., \alpha_k \rangle$ and sends the normal form $P_B = b_1 w b_2$ to Bob.
- Alice computes $K_A = a_1 P_B a_2$.
- Bob computes $K_B = b_1 P_A b_2$.

Since $a_1 b_1 = b_1 a_1$ and $a_2 b_2 = b_2 a_2$, we have $K = K_A = K_B$ the shared secret key.

The Ushakov-Shpilrain protocol adds significant complications for the attacker because now instead of conjugacy search problem the attacker must solve even more complicated decomposition problem. The decomposition problem is the problem of finding words $x, y \in G$ while being given the words $w_1 \in G$ and $w_2 = x w_1 y \in G$.

Chapter 7

Length-based attack

We evaluate the suitability of $Aut(F)$ for group-based cryptography by trying to solve the conjugacy search problem using the length-based attack. If we are able to solve conjugacy search problem, we can compute private keys of Alice and Bob and eventually the shared secret key.

7.1 Fundamentals of length-based attack

To break an instance of AAG encryption one must find a shared secret key $K = A^{-1}B^{-1}AB$ using publicly shared data (which is the sets $\bar{a}$, $\bar{b}$, $\bar{a}'$, $\bar{b}'$). It is easy to see that for finding the key $K = A^{-1}B^{-1}AB$ it is sufficient to find A as a product of generating elements from $\bar{a}$ because having $A = a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \ldots a_{s_L}^{\varepsilon_L}$ and set a' we can construct $K = A^{-1}B^{-1}AB = A^{-1} a_{s_1}'^{\varepsilon_1} a_{s_2}'^{\varepsilon_2} \ldots a_{s_L}'^{\varepsilon_L}$. Symmetrically, it is sufficient to find B as a product of generating elements from $\bar{b}$ to find K.

In fact, by [Shpilrain and Ushakov, 2006b], it is sufficient to find such $\widetilde{A} \in \langle a_1, \ldots, a_{N_1} \rangle$ that $b_i' = \widetilde{A}^{-1} b_i \widetilde{A}$ for all $1 \leq i \leq N_2$, i.e., to solve subgroup-restricted simultaneous conjugacy problem, i.e., solve the following system of conjugacy equations

$$\begin{cases} b'_1 = \widetilde{A}^{-1} b_1 \widetilde{A} \\ \qquad \dots \\ b'_{N_2} = \widetilde{A}^{-1} b_{N_2} \widetilde{A} \end{cases} , where\ \widetilde{A} \in \langle a_1, \dots, a_{N_1} \rangle$$

Any such $\widetilde{A}$ can be used for computing a shared key K. Symmetrically, it is sufficient to find such $\widetilde{B} \in \langle b_1, \dots, b_{N_2} \rangle$ that $a'_i = \widetilde{B}^{-1} a_i \widetilde{B}$ for all $1 \le i \le N_1$.

The length-based attack attempts finding the elements $a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_L}^{\varepsilon_L} = A$ using the elements of set b'. The underlying fact used for the length-based attack is that the length of an automorphism usually increases when multiplied by another automorphism, so $|A^{-1} b_i A| > |A^{-1} b_i| > |b_i|$ with high probability. This brings out the problem of defining a way to measure the length of an automorphism. We introduced the length function for automorphisms in Section 3.4

Now lets take a look at a single element b'_i

$$b'_i = A^{-1} b_i A = a_{s_L}^{-\varepsilon_L} \dots a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1}\ b_i\ a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_L}^{\varepsilon_L}$$

Because of the above-mentioned property that usually conjugation increases the length of the automorphism, we can assume that

$$|b'_i| < |a_{s_L}^{\varepsilon_L} b'_i a_{s_L}^{-\varepsilon_L}| = |a_{s_{L-1}}^{-\varepsilon_{L-1}} \dots a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1}\ b_i\ a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_{L-1}}^{\varepsilon_{L-1}}|$$

We can say that the conjugation of b'_i by $a_{s_L}^{-\varepsilon_L}$ peels off the outermost element that make up b'_i. We can go further and conjugate b'_i by $a_{s_{L-1}}^{-\varepsilon_{L-1}} a_{s_L}^{-\varepsilon_L}$ instead of just $a_{s_L}^{-\varepsilon_L}$. With the high probability that depends on elements a_s and the way A is generated, it will decrease the length even further because it peels off two automorphisms from each side of b'_i instead of just one, so

$$|b'_i| > |a_{s_L}^{\varepsilon_L} b'_i a_{s_L}^{-\varepsilon_L}| > |a_{s_L}^{\varepsilon_L} a_{s_{L-1}}^{\varepsilon_{L-1}} b'_i a_{s_{L-1}}^{-\varepsilon_{L-1}} a_{s_L}^{-\varepsilon_L}| =$$

$$= |a_{s_{L-2}}^{-\varepsilon_{L-1}} \dots a_{s_1}^{-\varepsilon_1}\; b_i\; a_{s_1}^{\varepsilon_1} \dots a_{s_{L-1}}^{\varepsilon_{L-2}}|$$

By making further steps in this direction we hope to come to the point when, after the next step of length reduction, we end up with the sequence S of elements of Alice's public set that reduces the length of $\bar{b}'$ to the length of $\bar{b}$ when you conjugate $\bar{b}'$ by it. Further, if $S^{-1}\bar{b}'S = \bar{b}$, then we have that $S^{-1} = \widetilde{A} \in \langle a_1, \dots, a_{N_1} \rangle$. As we have discussed in the beginning of this section, it is enough to compute the shared secret key K.

7.2 Length-based attack with backtracking

The idea discussed above is used for the length-based attack (subsequently called the LBA). The LBA we present next uses backtracking and will work even if at some point there are multiple elements of Alice's public set that can shorten the current automorphism. Later in this paper, we will discuss the way to improve it further. In the algorithm below, the length of the set of automorphisms should be understood as the sum of lengths of all automorphisms in the set.

Algorithm *LBA with backtracking*

1. Initialize a set $S = \{(\bar{b}', 1)\}$, where 1 is the identity element of $Aut(F)$.
2. If $S = 0$ then output *FAIL*.
3. Choose a pair $(c, x) \in S$ with the least $|c^x|$.
4. For each $i = 1 \dots N_1$ and each $\epsilon = \pm 1$ compute $\Delta_{i,\epsilon} = |c^x| - |c^{xa_i^\epsilon}|$.
    - If $c^{xa_i^\epsilon} = \bar{b}$ then output the final result $(xa_i^\epsilon)^{-1}$.

- If $\Delta_{i,\epsilon} > 0$ then add (c, xa_i^{ϵ}) to the set S.

5. Delete (c, x) from the set S.

6. Go to Step 2.

As we have already mentioned the whole LBA is based on a heuristic fact that the length of an automorphism usually increases when multiplied by another automorphism. With high probability when we multiply $a_{s_1}^{\varepsilon_1} \dots a_{s_{L-1}}^{\varepsilon_{L-1}}$ by $a_{s_L}^{\varepsilon_L}$ the length of the resulting automorphism increases. But what happens if such multiplication actually decreases the length? It means that on Step 4 of the algorithm $\Delta_{s_L}^{-\epsilon_L} \leq 0$ and consequently the algorithm will not be able to find $a_{s_L}^{\epsilon_L}$ component of A and, as a result, will not be able to find A.

Fail condition: if for Alice's private key $A = a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_L}^{\varepsilon_L}$ there exists $1 < i \leq L$ such that $|a_{s_1}^{\varepsilon_1} \dots a_{s_{i-1}}^{\varepsilon_{i-1}}| \geq |a_{s_1}^{\varepsilon_1} \dots a_{s_{i-1}}^{\varepsilon_{i-1}} a_{s_i}^{\varepsilon_i}|$ then the length-based attack with backtracking fails. We say that a *peak* happens in the i-th position in A.

There is a significant chance of the above condition being met for some parameters of AAG. To be more specific, the above fail condition happens more often for parameters with a large rank R of the free group F and a small amount of Nielsen transformations forming each element of the Alice's public set $\bar{a}$. In Table 7.1 you can see the results of the LBA attack with backtracking. In each cell we put the percentage of successful attacks. Each percentage is calculated based on 20 experiments. Alice's private key is generated by 40 elements from Alice's public set. Rows track the number of Nielsen transformations that make up each element of Alice's and Bob's public sets. Columns track the rank of a free group F. An experiment is successful if it finds the Alice's private key within 30 hours.

The next interesting question is this: What makes the above-mentioned fail condition for the LBA with backtracking become true. Based on our observations that follow the idea from [Myasnikov and Ushakov, 2007], the most frequent pattern that activates the fail condition is this: Somewhere in the Alice's private key A there is a fragment $a_i^{\pm 1} a_j a_i^{\mp 1}$ such

Number of Nielsen transformations	Rank 10	Rank 15	Rank 20
6	70%	40%	15%
7	90%	50%	45%
8	95%	85%	50%
9	100%	85%	65%

Table 7.1: Success rate of classic LBA attack with backtracking

that $|a_i^{\pm 1} a_j a_i^{\mp 1}| \leq |a_i^{\pm 1} a_j|$. The inverse of such $a_i^{\pm 1} a_j a_i^{\mp 1}$ often also causes the LBA to fail. On the one hand, the chance for a fragment of a form $a_i^{\pm 1} a_j a_i^{\mp 1}$ to appear in a randomly generated Alice's private key is significant. The smaller is the Alice's private set, the higher this chance is. On the other hand, the chance of $|a_i^{\pm 1} a_j a_i^{\mp 1}| \leq |a_i^{\pm 1} a_j|$ is also significant when the rank R of the free group F is relatively large and the elements of Alice's private set are generated by a small number of Nielsen transformations. To check if this case really attribute to the majority of LBA fails we have encountered in our experiments, we analyzed all failed LBA experiments from the Table 7.1. The results of this analysis are presented in Table 7.2. In each cell we state how many times in the failed experiments Alice's secret key had a fragment of the form $a_i^{\pm 1} a_j a_i^{\mp 1}$ such that $|a_i^{\pm 1} a_j a_i^{\mp 1}| \leq |a_i^{\pm 1} a_j|$. For example, 3/6 means that in 3 out of 6 unsuccessful experiments an Alice's private key contained such segment.

Number of Nielsen transformations	Rank 10	Rank 15	Rank 20
6	3/6	10/12	14/17
7	2/2	7/10	7/11
8	0/1	1/3	8/10
9	0/0	1/3	4/7

Table 7.2: Failures because of specific patterns

The initial analysis that we performed for Table 7.2 is too primitive to capture the potentially complex behaviour of automorphisms. Therefore, we are not claiming that the presented numbers indicate the problem precisely. Nevertheless, they show that the case we have described above is the potential troublemaker for the LBA and it might be worth it investigating it further. Also, this result explains the regularity that we observe in the experiments with the LBA with backtracking (see Table 7.1) that the longer the elements of the Alice's public set are and the smaller is the rank of F the better the LBA performs. Indeed, the longer the two automorphisms a_i and a_j are the smaller is the chance that $a_i^{\pm 1} a_j a_i^{\mp 1}$ such that $|a_i^{\pm 1} a_j a_i^{\mp 1}| \leq |a_i^{\pm 1} a_j|$. Also, that is the reason why we tend to conduct our experiments with shorter elements of the Alice's public set, to focus our attention on the most problematic area.

7.3 Enhanced LBA

From Table 7.2 it can be concluded that the case described above causes the significant part of LBA failures. [Myasnikov and Ushakov, 2007] suggests to enhance the standard LBA with backtracking to enable it handling this problematic situation. In this extended attack, it is suggested to run the usual LBA with backtracking but on an extended set of Alice's public automorphisms, rather than on the original set $\bar{a}$. In our case, this extended set contains all automorphisms of the original set $\bar{a}$, plus all conjugates $a_i^{\pm} a_j a_i^{\mp}$ of pairs $a_i, a_j \in \bar{a}$ where $1 \leq i, j \leq N_1$ and $i \neq j$ for which $|a_i^{\pm} a_j a_i^{\mp}| \leq |a_i^{\pm} a_j|$. Also, we add to the extended set all products $a_i^{\pm} a_j^{\pm}$ of pairs $a_i, a_j \in a$ where $1 \leq i, j \leq N_1$ and $i \neq j$ for which $|a_i^{\pm} a_j^{\pm}| \leq |a_i^{\pm}|$.

These elements that we use for the LBA along with the original set $\bar{a}$ hide inside them the most probable cases of fail condition being true. For example, if a decrease in length happens when we multiply $a_i^{-1} a_j$ by a_i ($|a_i^{-1} a_j a_i| \leq |a_i^{-1} a_j|$) and if the fragment $a_i^{-1} a_j a_i$ appears in the Alice's private key, the proposed enhanced LBA is still able to handle it. Still, it is not able to correctly find the a_j^{-1} step but it does not need to. The enhanced LBA has the whole conjugate $a_i^{-1} a_j a_i$ in its extended guessing set and it can conjugate b' by $(a_i^{-1} a_j a_i)^{-1}$.

The following algorithm is used to create an extended guessing set for the Enhanced

LBA.

Algorithm *Create an extended set*

---

1. Initialize set $Ext = \bar{a}$.
2. For all $1 \leq i < j \leq N_1$, and all $\varepsilon_i, \varepsilon_j = \pm 1$ for which $|a_i^{\varepsilon_i} a_j^{\varepsilon_j}| \leq |a_i|$ add $a_i^{\varepsilon_i} a_j^{\varepsilon_j}$ to Ext.
3. For all $1 \leq i, j \leq N_1$, $i \neq j$ and all $\varepsilon_i = \pm 1$ for which $|a_i^{\varepsilon_i} a_j a_i^{-\varepsilon_i}| \leq |a_i^{\varepsilon_i} a_j|$ add $a_i^{\varepsilon_i} a_j a_i^{-\varepsilon_i}$ to Ext.

Then we formulate the algorithm of the enhanced LBA:

Algorithm *Enhanced LBA*

---

1. Run the *Create an extended set* algorithm to create the set Ext.
2. Initialize a set $S = \{(\bar{b}', 1)\}$, where 1 is the identity element of $Aut(F_R)$.
3. If $S = 0$ then output *FAIL*.
4. Choose a pair $(c, x) \in S$ with the least $|c^x|$.
5. For each $\xi \in Ext$ and each $\epsilon = \pm 1$ compute $\Delta_{i,\epsilon} = |c^x| - |c^{x\xi_i^\epsilon}|$.
    - If $c^{x\xi_i^\epsilon} = \bar{b}$ then output the final result $(x\xi_i^\epsilon)^{-1}$.
    - If $\Delta_{i,\epsilon} > 0$ then add $(c, x\xi_i^\epsilon)$ to the set S.
6. Delete (c, x) from the set S.
7. Go to Step 3.

It is to be expected that for some parameters the extended set Ext will be much larger than the Alice's public set $\bar{a}$. Because of that the extended LBA algorithm can run much longer compared to the classic LBA with backtracking, so later we will propose some optimizations to speed up the enhanced LBA.

Number of Nielsen transformations	Rank 10	Rank 15	Rank 20
6	85%	75%	40%
7	100%	80%	90%
8	95%	100%	85%
9	100%	95%	85%

Table 7.3: Success rate of enhanced LBA on the extended guessing set

In Table 7.3 you can see the performance of the enhanced LBA.

Table 7.3 shows the significant improvement in efficiency of the enhanced LBA over the standard LBA with backtracking. The improvements are most obvious on larger ranks of F and shorter elements of $\bar{a}$, where the standard LBA lacks efficiency.

7.4 Properties of the $\bar{b}$ set

Next, we try to decrease the computational load of the suggested LBA attacks on AAG protocol. One way to make the LBA attacks more efficient is to use not all the public data but only the useful part of it. We examine the public set $\bar{b}'$ and try to understand what part of $\bar{b}'$ we really need for the efficient LBA attack.

In our experiments we use the public set of Bob $\bar{b}$ with 10 automorpisms in it. As a result, we have 10 automorphisms in the set $\bar{b}'$. This means that when we guess the next element from $\bar{a}$ that decreases the total length of $\bar{b}'$, we perform all 10 conjugations and reductions, one for each element of the $\bar{b}'$.

But the size of $\bar{b}'$ has a very little effect on the whole procedure of LBA. Theoretically, it is possible for some element of b_i to interact with the Alice's private key in a specific way to interfere with the LBA. Alice's private key can be presented as a sequence of elements from $\bar{a}$: $A = a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_L}^{\varepsilon_L}$. Then the situation in which $|a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1}| \leq |b_i|$ poses the potential problem, because it does not allow LBA to correctly guess a_{s_1}. In this case, other elements

Number of Nielsen transformations	Rank 10	Rank 15	Rank 20
6	0.13%	0.46%	1.31%
7	0.01%	0.11%	0.3%
8	0%	0.01%	0.11%
9	0%	0%	0.01%

Table 7.4: Chance of $|a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1}| \leq |b_i|$

of $\bar{b}'$ will still show the right way for the LBA.

Intuitively, it looks like the chance of $|a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1}| \leq |b_i|$ for randomly generated b_i and a_{s_1} is rather small. To evaluate this intuition we have conducted the number of experiments. For all parameters that we used in our experiments we were generating many random b_i and a_{s_1} and check if any of these pairs satisfy the condition $|a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1}| \leq |b_i|$. For a given rank r of free group F and number of Nielsen automorphisms L the experiment can be described the following way:

1. Generate random $b_i \in Aut(F)$ which is L Nielsen transformations long.
2. Generate random $a_{s_1} \in Aut(F)$ which is L Nielsen transformations long.
3. If $|a_{s_1}^{-1} b_i a_{s_1}| \leq |b_i|$, then experiment is successful. Otherwise the experiment is a failure.

The results of these experiments are presented in Table 7.4. Each cell contains the percentage of successful experiments, based on 10000 experiments.

From the Table 7.4 it can be concluded that for a randomly generated b_i and a_{s_1} the chance of $|a_{s_1}^{-1} b_i a_{s_1}| \leq |b_i|$ is very low. Theoretically, it is also possible that $|a_{s_2}^{-1} a_{s_1}^{-1} b_i a_{s_1} a_{s_2}| \leq |a_{s_1}^{-1} b_i a_{s_1}|$ while $|a_{s_1} a_{s_2}| > |a_{s_1}|$, but the chance of that happening should be even smaller.

We can conclude that most of the time when we minimize the total length of the set $\bar{b}'$, problems come from the Alice's private key A, and not from the elements of $\bar{b}$. With these considerations in mind, we can opt to use not the whole set $\bar{b}'$ but a part of it. There is no reason to use all elements of $\bar{b}'$ so we modify our enhanced LBA in the following way:

Algorithm *Enhanced LBA on partial* $\bar{b}$

```
1. Run the Create an extended set algorithm to create set $Ext$.
2. Choose a parameter $P \in \mathbb{N}$ such that $0 < P \leq N_2$. Create a set $\bar{b}''$ out of the first $P$ elements of the set $\bar{b}'$
3. Initialize a set $S = \{(\bar{b''}, 1)\}$, where 1 is the identity element of $Aut(F)$.
4. If $S = 0$ then output FAIL.
5. Choose a pair $(c, x) \in S$ with the least $|c^x|$.
6. For each $\xi \in Ext$ and each $\epsilon = \pm 1$ compute $\Delta_{i,\epsilon} = |c^x| - |c^{x\xi_i^\epsilon}|$.
    • If $\bar{b'}^{x\xi_i^\epsilon} = \bar{b}$ then output the final result $(x\xi_i^\epsilon)^{-1}$.
    • If $\Delta_{i,\epsilon} > 0$ then add $(c, x\xi_i^\epsilon)$ to the set $S$.
7. Delete $(c, x)$ from the set $S$.
8. Go to Step 3.
```

For our experiments we use P from 1 to 5 depending on other parameters.

Since we introduced this improvement for the enhanced LBA, we also introduced it to the classic LBA with backtracking. We have rerun all classic LBA experiments, which failed to complete in 30 hours, but this time we used only a part of b' set (exactly the same part of b' that we have used for the same enhanced LBA experiment). If the classic LBA with backtracking is successful on the partial b' set, we consider it successful. If the classic LBA with backtracking is successful only on the full set $\bar{b}$, we still consider the experiment successful, though we never observed such a situation in practice. Table 7.1 already accounts for this rule. In fact, none of the failed classic LBA experiments succeeded on partial b', but these additional experiments make Table 7.1 and Table 7.3 comparable.

We tried the enhanced LBA for the same problems that we used for the standard LBA with backtracking. This way you can compare Table 7.1 and Table 7.3.

The idea of using just a part of $\bar{b}'$ set helps to significantly reduce the computational complexity of LBA and it works good for longer automorphisms of $\bar{a}$ and $\bar{b}$. But if these automorphisms are short and LBA starts to struggle with extensive backtracking, ignoring the part of $\bar{b}'$ can theoretically lead to even more backtracking. This additional backtracking can easily overweight the benefits of shrinking the set $\bar{b}'$.

To avoid this problem we suggest yet another way to optimize the use of $\bar{b}'$. We call it *compressing* the set $\bar{b}'$.

Algorithm *Compressing* $\bar{b}'$

```
1. Compute the product $p$ of all elements of $\bar{b}'$.
2. Reduce $p$.
3. Consider $p$ to be the only element of compressed $\bar{b}'$.
```

Compressed set $\bar{b}'$ can be used in any LBA instead of original $\bar{b}'$. On the one hand, it reduces the chance of Alice's private key canceling out with itself in $\bar{b}'$. On the other hand, using the compressed $\bar{b}'$ was about two times faster than using the original set $\bar{b}'$ in our experiments.

7.5 Further improvements to the enhanced LBA

In this section, we try to increase the efficiency of the LBA attacks further. We do it by finding more probable peak-generating patterns to include them in the extended guessing set *Ext*. The enhanced LBA already offers some significant improvement over the classic LBA with backtracking. Still, the enhanced LBA can be improved.

The group $Aut(F)$ has a complex behavior that is often hard to predict. For example, if for two automorphism $x, y \in Aut(F)$ holds $|xy| > |x|$, it does not automatically imply that for every $z \in Aut(F)$ $|zxy| > |zx|$. Symmetrically, if $|xy| < |x|$, it does not automatically imply that for every $z \in Aut(F)$ $|zxy| < |zx|$.

It was observed that the enhanced LBA often fails to guess the Alice's key fragments with conjugates like $a_j^{-1}a_ia_j$ even if $|a_j^{-1}| < |a_j^{-1}a_i| < |a_j^{-1}a_ia_j|$. It happens because when the rank R of the free group F becomes significantly larger than the number of Nielsen automorphisms in each element of $\bar{a}$, there is a chance that when you multiply $a_j^{-1}a_i$ by a_j there will be a lot of reductions going on and final increase in the automorphism's length will be minimal. And when the increase in length is small, it increases the probability of length actually decreasing when it happens inside the Alice's secret key A. We discuss such unpredictable behaviour further in the Chapter 14.

Because of that, it makes a sense to add elements of the form $a_j^{-1}a_ia_j$ to the extended guessing set Ext even if $|a_j^{-1}a_ia_j| = |a_j^{-1}a_i| + C$ where $C \in \mathbb{N}$ is a small positive number. By increasing the parameter C we include more elements in the Ext set, thus making it more powerful. The algorithm for the set extending turns into

Algorithm *Create an extended set with extra elements*

```
1. Initialize a set $Ext = \bar{a}$.
2. Choose a set extension parameter $C \in \mathbb{N}, C \geq 0$.
3. For all $1 \leq i < j \leq N_1$, and all $\varepsilon_i, \varepsilon_j = \pm 1$ for which $|a_i^{\varepsilon_i}a_j^{\varepsilon_j}| \leq |a_i| + C$ add $a_i^{\varepsilon_i}a_j^{\varepsilon_j}$ to the set $Ext$.
4. For all $1 \leq i, j \leq N_1$, $i \neq j$ and all $\varepsilon_i = \pm 1$ for which $|a_i^{\varepsilon_i}a_ja_i^{-\varepsilon_i}| \leq |a_i^{\varepsilon_i}a_j| + C$ add $a_i^{\varepsilon_i}a_ja_i^{-\varepsilon_i}$ to the set $Ext$.
```

It is natural that the introduction of $C > 0$ increases the computational load on the computer. It happens in two ways. C increases the size of the Ext set. On one hand, the

Number of Nielsen transformations	Rank 10	Rank 15	Rank 20
6	95%	80%	50%
7	100%	85%	85%
8	95%	100%	95%
9	100%	100%	95%

Table 7.5: Success rate of enhanced LBA on the further extended ($C = 3$) guessing set

larger is Ext the more conjunctions and reductions should be made every time we deal with Ext. On the other hand, the larger is Ext the more length-decreasing choices the enhanced LBA has and the higher is the chance of making a wrong choice. Wrong choices lead to backtracking and increase the number of steps.

We use this new set-extension algorithm for the the enhanced LBA on partial $\bar{b}'$ set. For our experiments we have chosen the parameter $C = 3$. We used this parameter to rerun the experiments from Table 7.3. The results are presented in Table 7.5.

On average Table 7.5 shows the significant increase in attack's efficiency. However, for rank 20 and length 7 the percentage of success actually decreased. It happens because, as we have already discussed above, the introduction of $C > 0$ increases the computation complexity of the attack. Two experiments (one for rank 20, length 7 and the other one for rank 15 length 5) failed to complete in 30 hours for $C = 3$ even though they completed successfully for $C = 0$. We consider such experiments failures.

7.6 LBA with look-ahead

Here we introduce a modification of the enhanced LBA attack for a better performance on the most LBA-resistant parameters (that is large group rank and short elements of a and b).

While LBA with backtracking shows good results, its efficiency noticeably drops for shorter elements of $\bar{a}$. After inspecting a number of cases where the enhanced LBA had failed, it was noted that the main reason of failures is extensive backtracking. Indeed, each LBA step is computationally intensive. To make things worse, in most cases it is possible to make a few more length-reducing steps ahead after the wrong step, so there is a lot of backtracking involved. The majority of the failed experiments fail not because of the inability of the enhanced LBA to find the next step, not because of peaks, but because the experiment fails to complete in a given time (30 hours in our case). One might speculate that having more computational powers it is possible to successfully complete those experiments but we suggest another approach.

In addition to the original heuristic that the correct element reduces the length of the set $\bar{b}'$ the most, we add one more heuristic property that after picking the wrong element to reduce $\bar{b}'$ it is hard to find the next length-reducing element of $\bar{a}$ and even if such element exists this next-step reduction will be smaller compared to the next-step reduction that happens after the correct initial step.

We use this new heuristic to modify the enhanced LBA even further. Now we evaluate each step by the sum of its length reduction and maximum possible next-step length reduction if this step is taken. Such approach adds weight to the correct steps and decreases the amount of the backtracking. We apply this new rule only to the steps which use the original elements of $\bar{a}$. Also, for computing the maximum length of the next-step reduction we use only the elements of $\bar{a}$. Ignoring the elements that appeared during the set extension helps to greatly reduce the computational complexity of the algorithm. To modify the enhanced algorithm we introduce the metric for pairs (c, x) mentioned in enhanced LBA algorithm.

Metric $|(c, x)|_{LA}$

- if $x, x^{-1} \notin \bar{a}$ then $|(c, x)|_{LA} = |c^x|$
- if $x, x^{-1} \in \bar{a}$ then $|(c, x)|_{LA} = min|c^{xy^{\pm 1}}|$ where $y \in \bar{a}$

Now we can state the new version of LBA

Algorithm *LBA with Look-ahead*

```
1. Run Create an extended set with extra elements algorithm to create the set $Ext$.
2. Initialize a set $S = \{(\bar{b}', 1)\}$, where 1 is the identity element of $Aut(F)$.
3. If $S = 0$ then output FAIL.
4. Choose a pair $(c, x) \in S$ with the smallest $|(c, x)|_{LA}$.
5. For each $\xi \in Ext$ and each $\epsilon = \pm 1$ compute $\Delta_{i,\epsilon} = |c^x| - |c^{x\xi_i^\epsilon}|$.
     • If $c^{x\xi_i^\epsilon} = \bar{b}$ then output the final result $(x\xi_i^\epsilon)^{-1}$.
     • If $\Delta_{i,\epsilon} > 0$ then add $(c, x\xi_i^\epsilon)$ to the set $S$.
6. Delete $(c, x)$ from the set $S$.
7. Go to Step 3.
```

We conducted a series of 100 experiments where the same problems were attacked by

- the enhanced LBA on the extended set with extra elements with the parameter $C = 3$ and the full set $\bar{b}'$
- the LBA with look ahead on the extended set with extra elements with the parameter $C = 3$ and the full set $\bar{b}'$

In addition to registering the increased efficiency for the LBA with look-ahead, we observe one more benefit. When the enhanced LBA and the LBA with look ahead attempted the same problems, the sets of the successfully attacked problems do not fully match. And it means that we can use both approaches on the same problems to increase the rate of success even further. Therefore, in the following Table 7.6 we add the column "joint successes" that shows the total number of successful experiments when attacking by both methods. In these

Number of experiments	enhanced LBA	LBA with look ahead	joint successes
100	30	49	58

Table 7.6: Performance of the LBA with look-ahead compared to the enhanced LBA. The rank of the free group F is 20, each element of Alice's and Bob's set consists of 6 Nielsen transformations.

L_A	enhanced LBA	LBA with look ahead	joint successes
5	3	6	9
6	14	21	25
7	27	29	34

Table 7.7: Performance of look-ahead LBA and enhanced LBA for different L_A. The rank of the free group F is 20.

experiments the rank of the free group F is 20, each element of Alice's and Bob's set consists of 6 Nielsen transformations.

Also, we conducted a number of experiments to see how the precision of the LBA with look-ahead changes when the elements of $\bar{a}$ and $\bar{b}$ become shorter. We conducted 40 experiments where the lengths L_A and L_B of each element of $\bar{a}$ and $\bar{b}$ is 5, 6 and 7 Nielsen transformation. In all those experiments, the rank of F is 20, the number of elements in both $\bar{a}$ and $\bar{b}$ is 10, the number L of elements of $\bar{a}$ in A is 40. The number of experiments in each series is 40.

Chapter 8

Complexity estimation

In this section, we estimate the computational complexity of the LBA attacks. We approach the estimation from a slightly unusual perspective. Our main focus is on the best-case complexity. The reason for it is the following: we design our LBA attacks to stay close to the best-case. As we will show, the worst-case is so bad that there is no hope of solving it with any reasonably available computational powers.

First, lets analyze the classic LBA with backtracking. The Alice's secret key $A = a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \ldots a_{s_L}^{\varepsilon_L}$ consists of L elements. Therefore, the LBA attack should make L successful iterations to find it. Since we want to compute the best-case complexity, we assume that we do only the successful iterations. Each iteration is finding an element $a_i \in \bar{a}$ that yields the product $a_i \tilde{A}_{i-1} \bar{b}' \tilde{A}_{i-1}^{-1} a_i^{-1}$ of the smallest length, where $\tilde{A}_{i-1}$ is the product of the elements of $\bar{a}$ chosen on the previous $i-1$ iterations. To find such an element, we must check all possible candidates, that is all elements of the set $\bar{a}$. To compute the length of $a_i \tilde{A}_{i-1} \bar{b}' \tilde{A}_{i-1}^{-1} a_i^{-1}$ for the single element of $\bar{a}$, we must perform conjugation and reduction of the set $\tilde{A}_{i-1} \bar{b}' \tilde{A}_{i-1}^{-1}$ by a_i. The set $\tilde{A}_{i-1} \bar{b}' \tilde{A}_{i-1}^{-1}$ contains N_2 elements, thus we need to perform N_2 conjugations and reductions for every element of $\bar{a}$. There are N_1 elements in $\bar{a}$. Thus, the number of conjugations $\sharp_{conj}$ and the number of reductions $\sharp_{reduct}$ that we need to perform is

$$\sharp_{conj} = \sharp_{reduct} = LN_1N_2.$$

Considering the fact that none of the parameters should be really large, this estimation does not look bad at all. However, further we will present the actual measurements showing that conjugations and reductions are very expensive operations. Also, we should consider the inability of the classic LBA with backtracking to deal with the peaks in Alice's secret key A a trade-off for this relatively modest estimation.

Next we evaluate the number of conjugations and reductions needed in the best-case scenario for the LBA attack on the extended guessing set. There are the following differences compared to the complexity of the LBA with backtracking. First of all, there is the extended set itself. It replaces the original set $\bar{a}$ when it comes to choosing the conjugation element for the next iteration. Thus, we replace the size N_1 of the set $\bar{a}$ by N_{Ext} – the size of the extended guessing set. The natural question is this: how much larger is N_{Ext} compared to N_1? The answer strongly depends on the parameters been chosen for the AAG protocol. The longer is the length of the elements of $\bar{a}$ and the smaller is the rank R of the free group F the closer is N_{Ext} to the N_1. Subsequently we call such parameters "good" parameters, opposed to "bad" parameters (short elements of $\bar{a}$ and large rank R). For the set $\bar{a}$ with 10 elements each consisting of 9 Nielsen transformations and rank $R = 10$, the set $\bar{a}$ was hardly extending at all so N_1 and N_{Ext} were equal or almost equal. However, for the set $\bar{a}$ with 10 elements each consisting of 6 Nielsen transformations and rank $R = 20$, N_{Ext} was 10–15 times larger than N_1. Obviously, it constitutes 10–15 times growth of the computational complexity compared to the classical LBA, because we try conjugating by every element in our guessing set in order to make the next iteration step.

The next factor that changes complexity is the potentially different number of iteration steps L. Since the extended set contains elements that are made up of a few different elements of $\bar{a}$ (for example, conjugations), we can iterate towards the result a few elements at a time. Therefore, the number of steps can be smaller than L. In practice, we observed a small decrease of the number of iterations of the LBA on the extended set compared to the LBA with backtracking for some parameters. The problem is the parameters that give the minor decrease of the number of iterations are the same parameters that give the major growth of the extended guessing set. That is why it seems reasonable to ignore the potential decrease of the number of iterations and estimate the number of conjugations $\sharp_{conj}$ and the number of reductions $\sharp_{reduct}$ the following way

$$\sharp_{conj} = \sharp_{reduct} = LN_{Ext}N_2$$

If we move to the LBA with look-ahead the best-case computational complexity increases even further. Now for every element of the original set $\bar{a}$ that further decreases the length of $\bar{b}'$, we do the additional N_1 conjugations and reductions, one conjugation and reduction for every element of $\bar{a}$. So we can have up to N_1^2 additional computations and reductions on each iteration. Thus, the best-case estimation of the number of conjugations $\sharp_{conj}$ and the number of reductions $\sharp_{reduct}$ for the LBA with look-ahead is

$$\sharp_{conj} = \sharp_{reduct} \le L(N_{Ext} + N_1^2)N_2$$

The LBA with look-ahead shows the worst best-case complexity. Given the realistic parameters, the numbers themselves do not look large for modern computers. The problem is the operations of conjugation and reduction are really computationally intense. Later we will evaluate conjugation and reduction further, but now we can point out that their computational intensity is the reason we use their numbers as a measure of the computational intensity of LBA algorithms.

In practice, each iteration of the LBA attack takes a lot of time. All the versions of LBA presented above have backtracking in them but the practical cost of backtracking is very high. As if the high cost of each iteration is not bad enough, usually the LBA attack manages to make a few more steps after making a wrong one. Our experiments showed that one wrong choice can take the LBA into the maze of dozens or even hundreds of iterations before it returns on the right track.

This is the reason why we prefer the analysis of the best-case complexity. The idea is to keep the average-case of the attacks as close to the best-case as possible. This is the reason the LBA with look-ahead works better than the LBA with the extended set. We agree to perform a significant amount of additional computations on each step to minimize backtracking.

Evaluating the total number of all possible iterations on all possible paths (both right and wrong) is really hard. Of course, it heavily depends on the parameters but also varies greatly in different cases of the same parameters. Our experiments showed that often the attack can never (within a given time limit for the experiment of course) go back on the right track after making a wrong iteration. It means that each side-tree of iterations that diverts from the correct sequence of iterations can be pretty large. Depending on the parameters, there can be a lot of such side-trees on each iteration. For some "bad" parameters, it was not unusual to see a dozen of possible iterations on a single step and the large part of those iterations was wrong.

In our complexity discussion, we are mainly concerned about the number of conjugations and reductions that is needed to be done. In addition to the general discussion about their complexity, we carried out a number of small experiments to get better understanding of how time consuming conjugations and reductions of automorphisms are in practice. Of course, the specific numbers depend on the particular computer and implementation of the algorithms. The computer we used for those experiments has Intel Core i5-6200U 2.30 GHz processor and 8 GB of RAM. Both conjugations and reductions was implemented without multithreading. We performed 1000 experiments (conjunctions and reductions) for each set of parameters and computed the average time of each. As for the size of the conjugated and reduced automorphisms, we tried to keep it similar to the sizes that we deal with in our experiments. Each conjugation approximately corresponds to the conjugation of a single element of $\bar{b}'$ by an element of $\bar{a}$, each reduction reduces the result of such conjugation. The experiment can be described like this

1. Set the number $Nils$ of Nielsen transformations. It corresponds to the number of Nielsen transformations that make up each element of $\bar{a}$ and $\bar{b}$.
2. Set the rank $Rank$ of the free group F.
3. Repeat 1000 times
    - Generate a random automorphism α of rank $Rank$ made of $Nils$ Nielsen transformations. α has the same size as the elements of $\bar{a}$.
    - Generate a random automorphism β of rank $Rank$ made of $Nils * (2 * 40 + 1)$ Nielsen transformations. β has approximately the same size as the elements $\bar{b}'$ from our experiments. Keep in mind that our experiments use 40 elements of $\bar{a}$ to generate the Alice's secret key.
    - Conjugate β by α and measure the duration of that conjugation.
    - Reduce the result of conjugation of β by α and measure the duration of that reduction.
4. Compute the average duration of the above 1000 conjugations and also average reduction duration.

Below you can see the results.

Nils	Rank 10	Rank 15	Rank 20
6	143	66	35
7	214	102	53
8	289	145	74
9	384	204	106

Table 8.1: Average time to preform conjugation (in milliseconds)

Nils	Rank 10	Rank 15	Rank 20
6	410	155	72
7	668	253	116
8	980	390	174
9	1377	581	262

Table 8.2: Average time to preform reduction (in milliseconds)

Those results give a nice insight in why we measure the complexity of our algorithms in conjugations and reductions. Those operations turn out to be really computationally intensive and time consuming sometimes even to an extent of being humanly perceivable.

Also, we can measure the time to generate the Alice's secret key A. Random generation of A consisting of L elements of $\bar{a}$ requires $L-1$ multiplications and 1 reduction. It also requires the generation of a random sequence of L elements of $\bar{a}$, but it is reasonable to consider that the choice of L random elements to multiply is negligibly fast compared to multiplications and reductions. Below is the table with the actual average times it took to generate the secret Alice's key for different parameters. For each set of parameters there were 100 keys generated and the average generation time computed. For those experiments we used the same computer as for conjugations and reductions (Intel Core i5-6200U 2.30 GHz processor and 8 GB of RAM).

We choose the parameters for the above experiments to generate the keys of the sizes we used in our experiments. One should not try to directly compare these results with the results from the tables 8.1 and 8.2 because those tables deal with much larger automorphisms.

The last measurements we want to present here is time it takes to do all the computational

$Nils$	Rank 10	Rank 15	Rank 20
6	116	81	73
7	164	106	90
8	225	135	109
9	299	171	132

Table 8.3: Average time to generate an Alice's secret key (in milliseconds)

job to obtain the shared secret key as Alice. This computational job includes

1. Generation of a random set $\bar{a}$. Generation of each of N_1 elements of $\bar{a}$ requires the multiplication of L_A Nielsen transformations.
2. Generation of the set $\bar{b}'$. Generation of each of N_2 elements of $\bar{b}'$ requires 1 conjugation (2 multiplications) of an element of $\bar{b}$ by A.
3. Every element of $\bar{b}'$ must be reduced.
4. Constructing the shared secret key K. It requires multiplication of L elements of $\bar{a}'$ and the Alice's secret key A.
5. K must be reduced.

In this list we consider automorphisms inversions instantly available as it happens in our implementation.

The Step 5, reducing K, is a very computationally intensive step. Since the complexity of reduction is $O(|\mathcal{P}|^7)$ and K is very long, this step takes more time than all other steps of K generation combined. Because of it, we've chosen to measure the time of shared secret key generation for smaller parameters. We reduced the number L of elements of $\bar{a}$ in A (and correspondingly the number of elements of $\bar{b}$ in B) to 20.

$Nils$	Rank 10	Rank 15	Rank 20
6	295	89	30
7	524	178	56
8	1255	202	99
9	2949	473	164

Table 8.4: Average time to generate a shared secret key K (in milliseconds)

Chapter 9

Brute-force security

Here we discuss the possibility of replacing the LBA attacks by a straight-forward "try all possible keys" brute-force attack. To demonstrate the impropriety of the brute-force approach we estimate the number of possible keys that such attack should try. As before, we are given a set $\bar{a}$ that contains N_1 elements and the number of the elements of $\bar{a}$ in the Alice's secret key L. If no elements of $\bar{a}$ commute the number $\sharp_A$ of all possible keys is

$$\sharp_A = 2N_1(2N_1 - 1)^{L-1}$$

Such number of possible keys greatly discourages the brute-force attack, however the elements of $\bar{a}$ sometimes commute for "bad" parameters. We conducted the following experiment to evaluate the possible number of pairs of commuting elements in $\bar{a}$. We took one of the worst parameters that we have dealt with in this work (the rank $R = 20$, the length of each element of $\bar{a}$ equals 6, the number of elements in $\bar{a}$ is 10). We generated random sets $\bar{a}$ for those parameters and counted the number of pairs of commuting elements in each set. For 1000 sets, the average number of commuting pairs in each set was 0.534. In other words, we have found one commuting pair in every two sets. Thus, our evaluation of the number of possible Alice's secret keys A should not change a lot even for "bad" parameters.

We can conclude that the brute-force attack is inefficient and cannot replace the LBA attack.

It is important to mention that $\sharp_A$ is so large that even the quadratic speedup offered by the Grover's algorithm does not make the brute-force attack viable.

Chapter 10

Important properties of $Aut(F)$

In this section, we discuss the properties of $Aut(F)$. Those properties offer us the insight in the already observed LBA performance and provide us with a better understanding of $Aut(F)$ for future research.

10.1 Irregular peaks

The LBA attack that was described above has a high percentage of successful applications but in some experiments it still fails to find a key. For high ranks and short elements in Alice's and Bob's sets, the success rate of LBA drops below 60%. Those lower results are caused by the complex nature of the $Aut(F)$ group.

One major reason for LBA failures is the presence of peaks in Alice's secret key A. We teach the LBA to handle peaks by including possible peaks in the extended guessing set Ext. But some peaks in Alice's secret key appear without any predictable patterns and therefore are hard to include in the extended guessing set Ext.

Here is the example of such irregular peak to demonstrate. In this example, neither $\xi^{-1}\phi\xi$ nor $\xi^{-1}\psi\xi$ contain any peaks. Therefore, these conjugations do not seem to belong in Ext.

But the product of these conjugations produces a peak $|(\xi^{-1}\psi\xi) * (\xi^{-1}\phi\xi)| = |\xi^{-1}\psi\phi\xi| \leq |\xi^{-1}\psi\phi|$.

Example 10.1.1. *Consider the following automorphisms from $Aut(F_3)$*

Automorphism ψ :

$$\begin{cases} a \to ca \\ b \to a^{-1}b \\ c \to c \end{cases}$$

Automorphism ϕ :

$$\begin{cases} a \to a \\ b \to ab \\ c \to cbb \end{cases}$$

Automorphism ξ :

$$\begin{cases} a \to ab^{-1}b^{-1} \\ b \to b \\ c \to c \end{cases}$$

In the next example, the product $\psi\phi$ does not produce a peak and $|\psi\phi| > |\psi|$. But when $\psi\phi$ is a fragment of a longer sequence $\xi\psi\phi$ a peak occurs and $|\xi\psi\phi| < |\xi\psi|$. This example shows that the behaviour of automorphism as a part of a larger automorphism depends not only on the original automorphism itself but also on the context in which it is included.

Example 10.1.2. *Consider the following automorphisms from $Aut(F_3)$*

Automorphism ψ :

$$\begin{cases} a \to ab \\ b \to cb \\ c \to c \end{cases}$$

Automorphism ϕ :

$$\begin{cases} a \to ab^{-1} \\ b \to b \\ c \to cb \end{cases}$$

Automorphism ξ :

$$\begin{cases} a \to a \\ b \to b \\ c \to caaaa \end{cases}$$

Finding such irregular peaks in order to include them in Ext is hard because it is not obvious where to look for them. Also, the amount of such peaks is significant while the chance of them appearing in A is very low. Thus, even if we try to account for such low-probability peaks we end up with a very large Ext that slows down the LBA significantly without any reasonable increase of the LBA success rate.

Because of such complex behavior, we have introduced the guessing set with extra elements in Section 7.5. Also, this is one of the reasons why there is a chance of an LBA failure even on LBA-weak parameters.

10.2 Alternative normal form for the automorphisms from $Aut(F)$

So far, we were dealing with automorphisms presented as a set of images of all letters from the alphabet X of a free group F. This presentation of the automorphisms (which we will subsequently call ILP that stands for Images of the Letters Presentation) provides us with an efficiently computable normal form. However, that is not the only available presentation. Another, and rather obvious, way to present automorphisms is a sequence of Nielsen transformations. Since every automorphism from $Aut(F)$ can be presented as a sequence of Nielsen transformations, this presentation is quite possible.

We did not use this presentation before because of the problems with the normal form in it. Often times, the same automorphism can be presented by a few very different sequences so having an efficiently computable normal form is crucial. The example of such different presentations can be two sequences:

Sequence 1: $(a \to ab) * (b \to a^{-1}b) * (a \to ba) * (a \to a^{-1})$

Sequence 2: $a \to b$

The sequences 1 and 2 both define the same automorphism but it is not immediately obvious from the the presentations themselves.

The normal form for the Nielsen transformations presentation (subsequently called NTP) can be obtained from the previous ILP presentation of automorphisms. Indeed, since the images of X can be used as a normal form of an automorphism, we can define an unambiguous rules of converting this normal form to the NTP. For our experiments we choose the following algorithm:

Algorithm *Converting the automorphism ϕ from ILP to NTP*

1. Set Seq to be an initially empty sequence of Nielsen transformations.
2. If the length of ϕ is equal to the rank r of the free group F go to Step 4.
3. From all Nielsen transformations choose transformation n that minimizes $|\phi n|_{ILP}$.
    - If $|\phi|_{ILP} - |\phi n|_{ILP} > 0$, set $\phi = \phi n$ and add n to the beginning of Seq. Go to Step 2.

- If $|\phi|_{ILP} - |\phi n|_{ILP} = 0$, find the automorphism ψ of the same length as ϕ such that

  $n_1 ... n_i \phi = \psi$ and the length of ψ can be reduced by multiplication by one of Nielsen transformations. $n_1 ... n_i$ is the set of Nielsen transformations of types 3, 4 and 5. Add these transformations to the beginning of *Seq*. $\phi = \psi$. Go to Step 2.

4. Multiply ϕ by Nielsen transformations of type 1 to get rid of all inverse letters in alphabet letter's images. Add each such transformation to the beginning of *Seq*.

5. Multiply ϕ by Nielsen transformations of type 2 to bring ϕ to identity. Add each such transformation to the beginning of *Seq*.

6. Replace every transformation in *Seq* by it's inverse. The resulting sequence in *Seq* is the NTP form of the initial ϕ

In step 3 of this algorithm, we try to reduce the length as fast as we can but sometimes we can run into the situation when no single Nielsen transformation can reduce the length of the automorphism. In this case, we know that there is an automorphism that can be obtained from the current automorphism without changing the length and the second automorphism can be reduced further by one of the Nielsen transformations. This fact follows from the Nielsen reduction procedure.

The important thing to mention about the above algorithm of ILP to NTP conversion is its complexity. This algorithm is computationally heavy. Most of the work should be done on Step 3 when we do multiplication and reduction for every Nielsen transformation of type 3,4 and 5.

Thus, if we have an automorphism in NTP and we want to obtain its NTP normal form, we must perform two steps. First, we must convert NTP to ILP. Second, we must convert ILP back to NTP following the unambiguous rules of the conversion algorithm defined above.

The first step is fast. However, the second step is slow, and as of now there is no known way to convert ILP to NTP efficiently. During the LBA attack, we can afford to compute such normal form a few times but doing it all the time for realistically complex automorphisms is not practical.

The computationally expensive normal form of NTP and two more NTP properties that we will discuss next lead us to believe that NTP is not good for use in the LBA attacks. So what are these two NTP properties?

The first property is the small size of an automorphism in the NTP form. The sizes of the images of the letters of X grow exponentially with the linear growth of the number of Nielsen transformations that make up the automorphism. We use SLP to store and process these images because SLP promises the computationally- and memory-efficient way to deal with them. We said before that we expect SLPs to compare in efficiency to NTP presentation. However, in practice, because of the cancellations of the inverse letters in words, SLP presentation of X images is memory consuming and all operations on it are computationally intense. Therefore, if Alice and Bob use large automorphisms in NTP form they must convert them to the expensive normal form just a few times (for shared secret key and sets $\bar{a}'$ and $\bar{b}'$), while attacker got to do it many times. This computational imbalance allows to make LBA practically infeasible.

One might justly argue that LBA does not require computing normal forms often. Indeed, most of the time LBA cares only for the length of the automorphisms. We really need to compare the equality of the automorphisms only to check if we have reached the end of the attack. And it does not look so bad for the attacker. However, there is the second property that does not allow to use NTP form for the LBA easily.

The second property of NTP is instability and unpredictability of an above-discusses normal form. Here is what we mean by "unstable and unpredictable": If we have two automorphisms $\phi, \psi \in Aut(F)$, the normal form of the product $\phi * \psi$ can bare no resemblance to neither ϕ and ψ nor their normal forms. This property has two immediate consequences. First, it becomes unclear how we can deal with automorphisms in NTP during the LBA attack without constantly converting to ILP and potentially back to NTP. Second, this inconvenient property of NTP normal forms translates to the length measurement.

The most natural way to define the length of the automorphism in NTP is by the number of Nielsen transformations that make up its NTP form. However, for the two fully reduced automorphisms $\phi = n_1 * ... * n_{i-1} * n_i$ and $\psi = n_1 * ... * n_{i-1}$ where all n are Nielson transformations, the lengths of their normal forms can differ significantly in any manner. For example, the length of the normal form of ψ can be significantly larger than the length of the normal form of ϕ though one might expect the opposite behaviour.

To illustrate the unpredictable behaviour of the NTP, we conducted the following experiment. We generate the automorphism ϕ_{39} as a sequence of 39 Nielsen transformations of type 3 and 4 (length changing Nielsen transformations). Then we generate the automorphism ϕ_{40} by applying one more random Nielsen transformation of types 3 or 4 to ϕ_{39} that does not cancel out with the existing transformations. Thus, the lengths of ϕ_{39} and ϕ_{40} differ by 1. Then we compute normal NTP forms for both ϕ_{39} and ϕ_{40}. Here is the table showing the difference in lengths $|norm_{NTP}(\phi_{40})| - |norm_{NTP}(\phi_{39})|$. The total number of experiments is 100. The rank R of the base free group F is 10. The length is understood as a number of Nielsen transformations of types 3, 4 and 5 only.

As mentioned above, we expect the difference of lengths of the normal forms to be 1. Instead of it, the difference varies greatly and goes as high as 11. In 11% of the experiments we observe the decrease in length of the nornal form when going from 39 Nielsen transformations to 40. The counterintuitive results in 11 out of 100 experiments should be explained by the properties of the NTP normal form. In 8% of experiments, we observe no change of length and it is also unacceptable from the LBA point of view.

To demonstrate the properties of the NTP presentation further we define the specter of

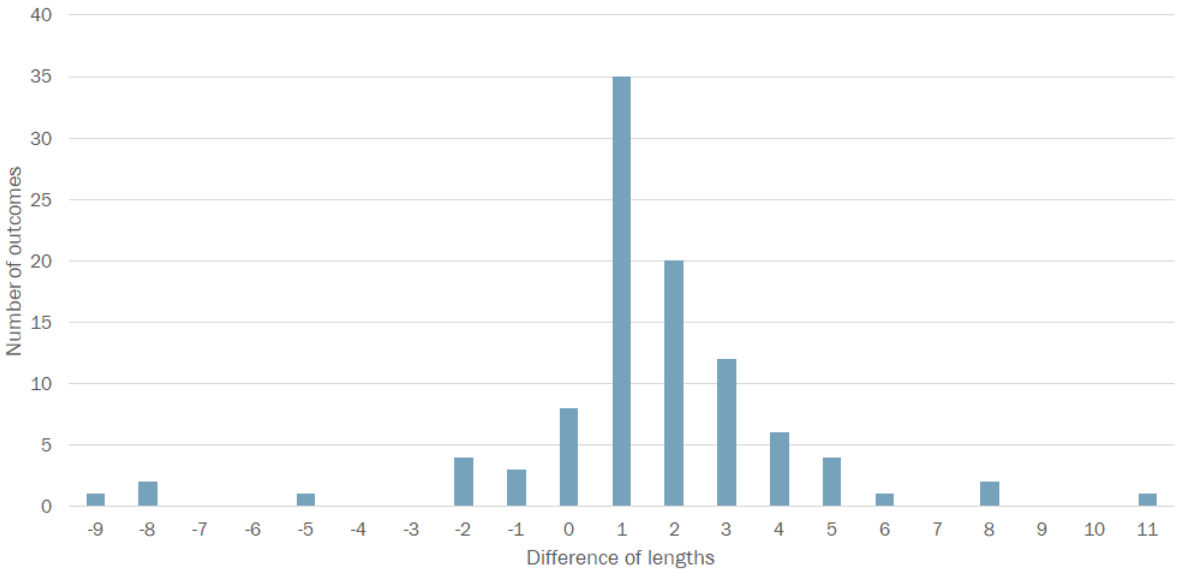

Figure 10.1: NTP length differences

NTP presentation of the automorphism.

Definition 10.2.1. *The specter of the NTP presentation of the automorphism $\phi \in Aut(F)$ is the number of occurrences of each Nielsen transformation in this NTP presentation.*

For the above definition, the number of occurrences of a Nielsen transformation and the number of occurrences of its inverse are counted separately. We used specter to further evaluate the difference of $norm_{NTP}(\phi_{40})$ and $norm_{NTP}(\phi_{39})$. It is intuitive that different specters mean differently looking automorphisms that are hard to compare without converting them to ILP.

We repeated the experiments for $norm_{NTP}(\phi_{40})$ and $norm_{NTP}(\phi_{39})$, but this time instead of comparing the length of $norm_{NTP}(\phi_{40})$ and the length $norm_{NTP}(\phi_{39})$ we compare their specters. We compute the ratio

$$similarity = common/total$$

where $total$ is the total number of Nielsen transformations in $norm_{NTP}(\phi_{39})$ (in other words, *total* is the NTP-length of $norm_{NTP}(\phi_{39})$) and *common* is the number of Nielsen transformations that occur in both $norm_{NTP}(\phi_{40})$ and $norm_{NTP}(\phi_{39})$. For example, if $norm_{NTP}(\phi_{39})$ has three occurrences of $a \to ab$ and $norm_{NTP}(\phi_{40})$ has only two, then $norm_{NTP}(\phi_{40})$ and $norm_{NTP}(\phi_{39})$ have two occurrences of $a \to ab$ in common.

Again, we conducted 100 experiments and for each experiment we computed the *similarity* value. The average value of *similarity* for 100 experiments was 0,456. Such a low level of similarity for the normal forms of the automorphisms that originally differed only by one Nielsen transformation is another demonstration of the unpredictable nature of the NTP normal forms of automorphisms.

All of the above lead us to believe that NTP form is unfitting for the LBA attacks. However, some other uses may be possible. For example, Alice and Bob can use the fact that NTP form is much shorter than ILP form in practice. It allows to share public data in a condensed form when the data channel bandwidth is limited. Also, NTP form allows an easy inversion of the automorphism, whereas there is no known efficient way to invert the automorphism in ILP form.

Chapter 11

LBA-resistant key generation

In this section, we discuss a way to improve the resistance of AAG to the LBA attacks by generating special non-random Alice's secret key A.

The idea comes from the observation of the major weakness of LBA. LBA attacks do not like peaks in the Alice's secret key and subsequently in the elements of $\bar{b}'$. In the enhanced and look-ahead LBA attacks, we extend the initial Alice's public set to fight those peaks. Still, the suggested set extension fights only the peaks of the following types:

- $...a_i a_{i+1}...$ where a part of a_{i+1} cancels out with a part of a_i.
- $...a_i...a_{i+2}...$ where a part of a_{i+2} cancels out with a part of a_i.

The extension allows the enhanced and look-ahead LBAs to efficiently attack randomly generated Alice's secret key A. But what happens if A contains a peak of a different type? The LBA attack fails.

The above mentioned types of peaks are responsible for the majority of peaks appearing in a randomly generated A and the chance of getting random peak of other type is very low. It means that we should move away from completely random key generation in order to reliably introduce such improbable peaks. Also, we introduce some rules for the Alice's public set $\bar{a}$, thus moving away from a completely random $\bar{a}$ as well.

The rules presented below show just one of the possible approaches to building LBA-resistant keys. We do not claim that they are optimal. Even more so, to present the idea we want to build a pretty simple example.

We split the Alice's public set $\bar{a}$ in three parts:

- General purpose elements. They are just random automorphisms.
- "Peak" elements – the elements that are used to create peaks.
- "Fill" elements – used to fill the space between "peak" elements without spoiling their ability to create peaks. Obviously, some restrictions should apply to both "peak" and "fill" elements.

Now, laying down really efficient rules for creating "peak" and "fill" elements can be quite a creative task in itself. To demonstrate how the whole construction can work, let us build a very simple example without claiming that it is optimal. Each element of $\bar{a}$ in the algorithm below initially has L_A Nielsen transformations in it.

Algorithm *Generating the initial "peak" automorphism p of length L_A*

1. $a = x_{R/2+1} \to x_{R/2+1}x_i$ or $a = x_{R/2+1} \to x_i x_{R/2+1}$ where R is the rank of the free group F and $i > R/2+1$
2. Multiply a by $L_A - 1$ Nielsen transformations of from $x_k \to x_k x_l^{\pm}$ or $x_k \to x_l^{\pm} x_k$ where $k, l < R/2+1$.
3. Reduce $p = Reduce(a)$.

Algorithm *Generating the "fill" automorphism f of length L_A*

1. $a = x_i \to x_{R/2+1}x_i$ or $a = x_i \to x_i x_{R/2+1}$ where R is the rank of the free group F and $i > R/2 + 1$
2. Multiply a by $L_A - 1$ Nielsen transformations of from $x_k \to x_k x_l^{\pm}$ or $x_k \to x_l^{\pm} x_k$ where $k, l \geq R/2 + 1$.
3. $f = Reduce(a)$.

It is an important property of "peak" and "fill" elements that they do not fully commute with each other. Using the above algorithms we can generate an Alice's public set

Algorithm *Creating the initial set $\bar{a}$ for LBA-resistant keys*

1. Choose a number N_{gp} of general purpose elements. Generate N_{gp} of the length L_A and place them in $\bar{a}$.
2. Choose a number N_{peak} of "peak" elements. Following the "peak" automorphism generation algorithm, generate N_{peak} "peak" automorphisms p_i and place them in $\bar{a}$.
3. Choose a number N_{fill} of "fill" elements. Following the "fill" automorphism generation algorithm, generate N_{fill} "fill" automorphisms f_i and place them in $\bar{a}$.

From the elements of the LBA-resistant set $\bar{a}$, we can generate an LBA-resistant Alice's private key A using the following algorithm:

Algorithm *Generating an LBA-resistant key A of S_L elements of $\bar{a}$*

1. Choose the numbers $head, construct, mid, end > 1$ such that $head + construct + mid + construct + end = S_L$
2. $A = a_{sh_1} \cdot \ldots \cdot a_{sh_{head}}$, where a_{sh} are the elements of $\bar{a}$ or its inverses and no neighbouring elements cancel each other out.

3. $A = A \cdot p_{sp_1} \cdot ... \cdot p_{sp_{construct}}$, where $p_{sp_1} \cdot ... \cdot p_{s_{construct}}$ is a random sequence of "peak" elements and their inverses, where no neighbouring elements are the same or inverses of each other (including $a_{sh_{head}}$).

4. $A = A \cdot f_{sf_1} \cdot ... \cdot f_{sf_{mid}}$, where $f_{sf_1} \cdot ... \cdot f_{sf_{mid}}$ is a random sequence of "fill" elements and their inverses, where no neighbouring elements are the same or inverses of each other.

5. $A = A \cdot p^{-1}_{sc_{construct}} \cdot ... \cdot p^{-1}_{sc_1}$, where $p^{-1}_{sc_{construct}} \cdot ... \cdot p^{-1}_{sc_1}$ is a inverted sequence of "peak" elements and their inverses from Step 3. After multiplication by each p^{-1}_{sc}, perform reduction. If after the reduction the length of A has increased, increase the p_{sc} by a random Nielsen transformation of from $x_k \to x_k x_l^{\pm}$ or $x_k \to x_l^{\pm} x_k$ where $k, l < R/2+1$, discards the current A and go back to step 2.

6. $A = A \cdot a_{se_1} \cdot ... \cdot a_{se_{end}}$, where $a_{se_1} \cdot ... \cdot a_{se_{end}}$ is a product of a random sequence of elements of $\bar{a}$ and their inverses, where no neighbouring elements cancel each other (including $p^{-1}_{sc_1}$) out.

7. Reduce A

The idea of the above algorithm is this: each inverse of "peak" element added on Step 5 largely cancels out with the corresponding "peak" element added on Step 3. Adding "fill" elements on Step 4 ensures that the "peak" elements do not cancel each other out completely. Thus, we expect each "peak" element added on Step 5 to reduce the length after reduction. It creates a peak of height *construct* meaning that for *construct* multiplications in a row the length of A decreases after reduction.

We conducted the series of experiments trying to break LBA-resistant Alice's secret keys with both enhanced LBA and LBA with look-ahead. We use the rank of a free group $R = 10$ because this lower rank is favorable to LBA. Other parameters: the number of elements of $\bar{a}$ in A is 20 (8 random elements of $\bar{a}$, 5 "stumble" elements then 2 "fill" elements and finally 5

L_A	9	8	7	6
Enchanced LBA	0/20	0/20	0/20	0/20
LBA with look-ahead	0/20	0/20	0/20	0/20

Table 11.1: Success rate of enhanced LBA and LBA with look-ahead for different L_A for a securely generated key A

more "stumble" elements that are the inverses of the first 5). Table 11.1 shows the results of the experiments. Each cell represents the number of successful attacks out of 20 experiments for that set of parameters. One can see that none of the LBA-resistant keys was broken by any version of LBA.

The natural question is, can we further extend the Alice's public set $\bar{a}$ to account for the new type of peaks. The answer is yes, we can. But lets evaluate how large such extension can be.

The constructions that generates the peak in A looks like this

$$peak = a_{t_1} \cdot \ldots \cdot a_{t_{construct}} \cdot a_{f_1} \ldots \cdot a_{f_{mid}} \cdot a_{t_{construct}}^{-1} \cdot \ldots \cdot a_{t_1}^{-1}$$

where a_t are "peak" elements and a_f are "fill" elements. To avoid including in the extended set all such constructions for all possible values of *construct* and *mid*, we can extend $\bar{a}$ with a smaller number of all possible elements of the form

$$a_{ext} = a_{t_1} \cdot \ldots \cdot a_{t_{construct}} \cdot a_f \cdot a_{t_{construct}}^{-1} \cdot \ldots \cdot a_{t_1}^{-1}$$

Indeed, if we multiply *peak* construction by the corresponding a_{ext} the *peak* construction looses one "fill" element.

If the number of possible "peak" elements is N_{peak} and the number of possible "fill" elements is N_{fill} then the total number $N_{a_{ext}}$ of all possible a_{ext} is

$$N_{a_{ext}} > construct^{2N_{peak}-1} \cdot 2N_{fill}$$

The above estimation shows that the number $N_{a_{ext}}$ grows fast, and even small values of *construct*, N_{peak} and N_{fill} extend the set $\bar{a}$ so much that LBA attack becomes computationally too difficult.

On the other hand, generating the LBA-resistant key A is not much more difficult than generating a completely random A. Compared to generating a random A, the LBA-resistant A can require some additional automorphism multiplications and reductions if in Generating the LBA-resistant key algorithm there is jumping from the step 5 to the step 2. In practice, we did not observe any significant delay in Alice's key generation when we switched from random keys to LBA-resistant keys.

Chapter 12

Conclusion

We have analyzed the security properties of the Anshel-Anshel-Goldfeld protocol used with the base group of automorphisms of a free group. We found that the AAG protocol used with $Aut(F)$ is susceptible to different versions of the length-based attack. We showed the efficient way to use the existing versions of the LBA attack and also introduced our own new version of LBA which we call the LBA with look-ahead. The LBA with look-ahead provides us with a significantly increased success rate for the most attack-resistant combinations of parameters. All the proposed versions of LBA can be used for other cryptography protocols based on different versions of the conjugacy search problem (notably the Ko-Lee protocol) used with $Aut(F)$.

However, our experiments and theoretical evaluations showed that there are the parameters that present additional challenges for the LBA attacks. These parameters are a high rank R of the free group F and a low number of Nielsen transformations in the elements that make up $\bar{a}$ and $\bar{b}$, the public sets of Alice and Bob. We showed that the success rate of the LBA attacks drops for such parameters. Therefore, these parameters can be used to develop a secure way to use the AAG protocol with $Aut(F)$.

We experimented with a presentation of automorphisms in a form of the sequence of

Nielsen transformations and suggested to use it as a compact way to exchange automorphisms between Alice and Bob. This presentation that we called NTP allows easy inversion of the automorphism. It is especially useful since ILP presentation does not have an efficient algorithm for automorphism inversion.

We suggested to move away from the random key generation and use the known limitations of the LBA attacks to create private keys that are resistant to such attacks. We suggested one such method that generated secret keys, none of which was broken by any LBA attack in our experiments.

Bibliography

[Anshel et al., 1999] Anshel, I., Anshel, M., and Goldfeld, D. (1999). An algebraic method for public-key cryptography. *Math. Res. Lett.*, 6(3-4):287–291.

[Anshel et al., 2006] Anshel, I., Anshel, M., Goldfeld, D., and Lemieux, S. (2006). Key agreement, the algebraic eraser^{TM}, and lightweight cryptography. In *Algebraic Methods in Cryptography*, volume 418 of *Contemporary Mathematics*, pages 1–34. American Mathematical Society.

[Bigelow, 2001] Bigelow, S. (2001). Braid groups are linear. *J. Amer. Math. Soc.*, 14:471–486.

[Birman et al., 2007] Birman, J. S., Gebhardt, V., and Gonzalez-Meneses, J. (2007). Conjugacy in Garside groups I: Cyclings, powers, and rigidity. *Groups Geom. Dyn.*, 1:221–279.

[Cramer and Shoup, 1998] Cramer, R. and Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – CRYPTO 1998*, volume 1462 of *Lecture Notes Comp. Sc.*, pages 13–25, London, UK. Springer-Verlag.

[D. Atkins, 2016] D. Atkins, D. G. (2016). Addressing the algebraic eraser diffie–hellman over-the-air protocol. *Cryptology ePrint Archive.*

[D. Hart, 2017] D. Hart, D. Kim, G. M. G. P. C. P. Y. Q. (2017). A practical cryptanalysis of walnutdsa. *Cryptology ePrint Archive.*

[Dehn, 1911] Dehn, M. (1911). Uber unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE T. Inform. Theory*, IT-22:644–654.

[E. El-Rifai, 1994] E. El-Rifai, H. M. (1994). Algorithms for positive braids. *Quart. J. Math.*, pages 479–497.

[Eick and Kahrobaei,] Eick, B. and Kahrobaei, D. Polycyclic groups: a new platform for cryptology? Preprint. Available at `http://arxiv.org/abs/math.GR/0411077`.

[ElGamal, 1985] ElGamal, T. (1985). A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE T. Inform. Theory*, IT-31:469–473.

[F.Garside, 1969] F.Garside (1969). The braid group and other groups. *Quart. J. Math.*, pages 235–254.

[Franco and González-Meneses, 2003] Franco, N. and González-Meneses, J. (2003). Conjugacy problem for braid groups and garside groups. *J. Algebra*, 266:112–132.

[Garber et al., 2015] Garber, D., Kahrobaei, D., and Lam, H. (2015). Length-based attacks in polycyclic groups. *J. Math. Crypt.*, z:xx–yy.

[Garber et al., 2005] Garber, D., Kaplan, S., Teicher, M., Tsaban, B., and Vishne, U. (2005). Probabilistic solutions of equations in the braid group. *Adv. Appl. Math.*, 35:323–334.

[Garber et al., 2006] Garber, D., Kaplan, S., Teicher, M., Tsaban, B., and Vishne, U. (2006). Length-based conjugacy search in the braid group. In *Algebraic Methods in Cryptography*, volume 418 of *Contemp. Math.*, pages 75–88. Amer. Math. Soc.

[Gebhardt, 2005] Gebhardt, V. (2005). A new approach to the conjugacy problem in garside groups. *J. Algebra*, 292:282–302.

[Gebhardt, 2006] Gebhardt, V. (2006). Conjugacy search in braid groups from a braid-based cryptography point of view. *Appl. Algebra Eng. Comm.*, 17:219–238.

[Grover, 1996] Grover, L. (1996). A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219.

[Gunnells, 2011] Gunnells, P. (2011). On the cryptanalysis of the generalized simultaneous conjugacy search problem and the security of the algebraic eraser. *Cryptography and Security*.

[Hellman, 2002] Hellman, M. E. (May 2002). An overview of public key cryptography. *IEEE Communications Magazine*, pages 42–49.

[Hofheinz and Steinwandt, 2003] Hofheinz, D. and Steinwandt, R. (2003). A practical attack on some braid group based cryptographic primitives. In *Advances in Cryptology – PKC 2003*, volume 2567 of *Lecture Notes Comp. Sc.*, pages 187–198, Berlin. Springer.

[Hsu, 2018] Hsu, J. (2018). Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy. Institute of Electrical and Electronics Engineers.

[Hughes, 2002] Hughes, J. (2002). A linear algebraic attack on the AAFG1 braid group cryptosystem. In *The 7th Australasian Conference on Information Security and Privacy ACISP 2002*, volume 2384 of *Lecture Notes Comp. Sc.*, pages 176–189, Berlin. Springer.

[I. Anshel, 2017a] I. Anshel, D. Atkins, D. G. P. G. (2017a). Kayawood, a key agreement protocol. *Cryptology ePrint Archive.*

[I. Anshel, 2017b] I. Anshel, D. Atkins, D. G. P. G. (2017b). Walnutdsa(tm): A quantum-resistant digital signature algorithm. *Cryptology ePrint Archive.*

[J. Cheon, 2003] J. Cheon, B. J. (2003). A polynomial time algorithm for the braid diffie-hellman conjugacy problem. *CRYPTO*, pages 212–225.

[Jez, 2015] Jez, A. (2015). Faster Fully Compressed Pattern Matching by Recompression. volume 11, pages 20:1–20:43.

[Kelly, 2018] Kelly, J. (2018). A preview of bristlecone, googles new quantum processor. Google AI Blog.

[Knight, 2017] Knight, W. (2017). IBM raises the bar with a 50-qubit quantum computer. MIT Technology Review.

[Ko et al., 2000] Ko, K. H., Lee, S. J., Cheon, J. H., Han, J. W., Kang, J., and Park, C. (2000). New public-key cryptosystem using braid groups. In *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes Comp. Sc.*, pages 166–183, Berlin. Springer.

[Kotov and Ushakov,] Kotov, M. and Ushakov, A. Analysis of a certain polycyclic-group-based cryptosystem. Submitted to JMC. Available at `http://arxiv.org/abs/1504.05040`.

[Lee and Lee, 2002] Lee, S. J. and Lee, E. (2002). Potential weaknesses of the commutator key agreement protocol based on braid groups. In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes Comp. Sc.*, pages 14–28, Berlin. Springer.

[Lifshits, 2007] Lifshits, Y. (2007). Processing Compressed Texts: A Tractability Border. In *Annual Symposium on Combinatorial Pattern Matching – CPM 2007*, volume 4580 of *Lecture Notes Comp. Sc.*, pages 228–240. Springer.

[Lohrey, 2012] Lohrey, M. (2012). Algorithmics on SLP-compressed strings: A survey. *Groups, Complexity, Cryptology*, 4:241–299.

[Longrigg and Ushakov, 2009] Longrigg, J. and Ushakov, A. (2009). A practical attack on a certain braid group based shifted conjugacy authentication protocol. *Groups Complex. Cryptol.*, 1:275–286.

[Lyndon and Schupp, 2001] Lyndon, R. and Schupp, P. (2001). *Combinatorial Group Theory.* Classics in Mathematics. Springer.

[M. Kotov, 2018] M. Kotov, A. Menshov, A. U. (2018). An attack on the walnut digital signature algorithm. *Cryptology ePrint Archive.*

[Maffre, 2005] Maffre, S. (2005). Reduction of conjugacy problem in braid groups, using two garside structures. *WCC*, pages 214–224.

[Maffre, 2006] Maffre, S. (2006). A weak key test for braid-based cryptography. *Designs, Codes and Cryptography*, pages 347–373.

[Magnus et al., 1976] Magnus, W., Karrass, A., and Solitar, D. (1976). *Combinatorial Group Theory.* Dover Publications, Inc.

[Matucci, 2008] Matucci, F. (2008). Cryptanalysis of the Shpilrain-Ushakov protocol for Thompson's group. *J. Cryptology*, 21:458–468.

[Menezes et al., 1996] Menezes, A. J., van Oorschot, P., and Vanstone, S. (1996). *Handbook of Applied Cryptography.* CRC Press.

[Miasnikov et al., 2005] Miasnikov, A. G., Shpilrain, V., and Ushakov, A. (2005). A practical attack on some braid group based cryptographic protocols. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes Comp. Sc.*, pages 86–96, Berlin. Springer.

[Miasnikov et al., 2006] Miasnikov, A. G., Shpilrain, V., and Ushakov, A. (2006). Random subgroups of braid groups: an approach to cryptanalysis of a braid group based cryptographic protocol. In *Advances in Cryptology – PKC 2006*, volume 3958 of *Lecture Notes Comp. Sc.*, pages 302–314, Berlin. Springer.

[Miasnikov et al., 2011] Miasnikov, A. G., Shpilrain, V., and Ushakov, A. (2011). *Non-Commutative Cryptography and Complexity of Group-Theoretic Problems.* Mathematical Surveys and Monographs. AMS.

[Myasnikov and Ushakov, 2007] Myasnikov, A. D. and Ushakov, A. (2007). Length based attack and braid groups: Cryptanalysis of Anshel-Anshel-Goldfeld key exchange protocol. In *Advances in Cryptology – PKC 2007*, volume 4450 of *Lecture Notes Comp. Sc.*, pages 76–88. Springer.

[Myasnikov and Ushakov, 2009] Myasnikov, A. D. and Ushakov, A. (2009). Cryptanalysis of Anshel-Anshel-Goldfeld-Lemieux key agreement protocol. *Groups Complex. Cryptol.*, 1:263–275.

[Nielsen, 1921] Nielsen, J. (1921). Om regning med ikke-kommutative faktorer og dens anvendelse i gruppeteorien. *Math. Tidsskrift B.*

[Nielsen, 1924] Nielsen, J. (1924). Om regning med ikke-kommutative faktorer og dens anvendelse i gruppeteorien. *Mathematische Annalen.*

[Plandowski, 1994] Plandowski, W. (1994). Testing equivalence of morphisms on context-free languages. In *Algorithms-ESA 1994 (Utrecht)*, volume 855 of *Lecture Notes Comp. Sc.*, pages 460–470. Springer-Verlag.

[Pollard, 1974] Pollard, J. M. (1974). Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76.

[R. Rivest, 1978] R. Rivest, A. Shamir, L. A. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM.*

[Ruinsky et al., 2007] Ruinsky, D., Shamir, A., and Tsaban, B. (2007). Cryptanalysis of group-based key agreement protocols using subgroup distance functions. In *Advances in Cryptology – PKC 2007*, volume 4450 of *Lecture Notes Comp. Sc.*, pages 61–75. Springer.

[S. Blackburn, 2016] S. Blackburn, M. R. (2016). On the security of the algebraic eraser tag authentication protocol. *Applied Cryptography and Network Security*, pages 3–17.

[Schleimer, 2008] Schleimer, S. (2008). Polynomial-time word problems. *Comment. Math. Helv.*, 83:741–765.

[Shor, 1997] Shor, P. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509.

[Shpilrain and Ushakov, 2005] Shpilrain, V. and Ushakov, A. (2005). Thompson's group and public key cryptography. In *Applied Cryptography and Network Security – ACNS 2005*, volume 3531 of *Lecture Notes Comp. Sc.*, pages 151–164. Springer.

[Shpilrain and Ushakov, 2006a] Shpilrain, V. and Ushakov, A. (2006a). A new key exchange protocol based on the decomposition problem. In *Algebraic Methods in Cryptography*, volume 418 of *Contemporary Mathematics*, pages 161–167. American Mathematical Society.

[Shpilrain and Ushakov, 2006b] Shpilrain, V. and Ushakov, A. (2006b). The conjugacy search problem in public key cryptography: unnecessary and insufficient. *Appl. Algebra Engrg. Comm. Comput.*, 17:285–289.

[Tsaban, 2013] Tsaban, B. (2013). Polynomial-time solutions of computational problems in noncommutative-algebraic cryptography. *Journal of Cryptology*, 28:601–622.

[V. Gebhardt, 2008] V. Gebhardt, J. G.-M. (2008). The cyclic sliding operation in garside groups.

[W. Beullens, 2018] W. Beullens, S. B. (2018). Practical attacks against the walnut digital signature scheme. *Cryptology ePrint Archive.*

[Whitehead, 1936] Whitehead, J. (1936). On equivalent sets of elements in a free group. *Ann. of Math.*, 37:782–800.