# GROUPS, COMPLEXITY, CRYPTOLOGY

by

Maggie E. Habeeb

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

2012

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

Delaram Kahrobaei

Date — Chair of Examining Committee

Jozef Dodziuk

Date — Executive Officer

Delaram Kahrobaei

Vladimir Shpilrain

Alexei Miasnikov

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

GROUPS, COMPLEXITY, CRYPTOGRAPHY

By: MAGGIE E. HABEEB

Advisor: Delaram Kahrobaei

The field of non-commutative group based cryptography has flourished in the past twelve years with the increasing need for secure public key cryptographic protocols. This has led to an active line of research called non-abelian group based cryptography.

In this work, I in collaboration with Delaram Kahrobaei and Vladimir Shpilrain introduce a new public key exchange protocol based on a group theoretic problem and propose an appropriate platform group for this protocol. This work can be found in [19] and [20]. In addition, I in collaboration with Delaram Kahrobaei and Vladimir Shpilrain propose two new secret sharing schemes that utilize non-abelian groups. These schemes have some advantages over Shamir's secret sharing scheme (see [21] for the full paper). We propose a class of groups, namely small cancellation groups, to implement these secret sharing schemes.

Choosing the platform groups used in group based cryptographic protocols is vital to their security. D. Kahrobaei and B. Eick proposed in [12] polycyclic groups as a potential platform for these cryptographic protocols. Polycyclic groups were also proposed as

platform groups for group based cryptographic protocols in [28] and [10]. An important feature of polycyclic groups, and hence finitely generated nilpotent groups, is that they are linear. I in collaboration with Delaram Kahrobaei considered the complexity of an embedding of a finitely generated torsion free nilpotent group into a linear group (see [18]). We determined the complexity of an algorithm introduced by W. Nickel in [37] that determined a $\mathbb{Q}$-basis for a finite dimensional faithful $G$-module, which gives a bound on the dimension of the matrices produced. In [18] we also modified Nickel's algorithm for building a $\mathbb{Q}$-basis in order to improve the running time of the algorithm.

# Acknowledgements

I would like to thank Delaram Kahrobaei, my thesis advisor, for all of her support and encouragement. I am extremely grateful for all of the guidance she has given me. I greatly appreciate all the stimulating mathematical discussions had with Vladimir Shpilrain, my second advisor, which had a large impact on my research. I am grateful to have had the opportunity to collaborate with him. I would also like to thank Alberto Baider and John Loustau for their guidance throughout my graduate career, both at CUNY Hunter College and CUNY Graduate Center. Many thanks are due to Ben Fine and Gerhard Rosenberger for giving me the opportunity to collaborate with them early in my graduate career. As executive officer, Jozef Dodziuk provided support and guidance throughout my academic career. Lastly, I would like to thank Alexei Miasnikov for serving on my defense committee.

I acknowledge the support of NSF-LSAMP and PSC CUNY award through a grant by Delaram Kahrobaei.

New York M.E.H.
2012

**To my husband, Ibrahim Habeeb,**

**and my mom, Susan Elkordy,**

**for their support.**

# Preface

The field of non-abelian group based cryptography has become an active topic of research in the recent years. The non-abelian key exchange protocols by Anshel-Anshel-Goldfeld in [1] and Ko-Lee et. al in [29] inspired many mathematicians and cryptographers to design new key exchanges based on non-abelian groups. These new key exchanges have their security based on group theoretic problems. In order to know more about the security of these new protocols it is useful to know the computational complexity of algorithms that solve these underlying problems. This work concerns non-abelian group based cryptography as well as computational complexity.

We begin, in **Chapter 1**, by giving an introduction to cryptography since traditional key exchange protocols serve as a motivation for the new group based protocols. We then give a few examples of new group based key exchange protocols that have been developed over the last twelve years. Lastly, we briefly describe the notion of generic complexity; that is, the idea to consider how

efficiently an algorithm works on most inputs rather than simply considering the worst case scenario.

We give a brief background on computational complexity in **Chapter 2**. We review Turing machines, complexity classes, reductions and complete problems.

Next, in **Chapter 3**, we give a few examples of proposed platform groups for these new cryptographic protocols. We first define each group, and then state the properties that make these groups good candidates for platform groups. The collection of groups listed in this chapter is no where near comprehensive. We refer the reader to [35] and [13] for a more comprehensive list of proposed platform groups.

In **Chapter 4** we describe new key exchange protocols based on the endomorphism search problem in [19] and [20]. We give the proposed platform groups for these protocols and describe the difficulty of the endomorphism search problem in these groups. We discuss a cryptanalysis of the key exchange presented in [19] due to [5]. We then provide an alternate platform group to thwart the attack due to [5]. Lastly, two simplified protocols based on the protocol in [19] are proposed.

We then describe a different kind of cryptographic protocol in

**Chapter 5**; namely, we discuss secret sharing schemes. We provide an introduction to this type of protocol, and then give the traditional example of a secret sharing scheme due to Shamir. Then, we give an example of a secret sharing scheme using group presentations and the word problem presented in [38]. We propose two new secret sharing schemes that also use group presentations and the word problem. We discuss the efficiency and security of the new schemes we propose. Both schemes we propose have some advantages over Shamir's secret sharing scheme.

Lastly, in **Chapter 6**, we describe an algorithm provided by W. Nickel in [37] that provides an embedding of a finitely generated torsion free nilpotent group into a matrix group. This algorithm allows one to get matrix representations of elements of a finitely generated torsion free nilpotent group. We determine the worst case complexity of the crux of this algorithm, which is computing a $\mathbb{Q}$-basis for a $G$ module generated by certain polynomials. In addition, we were able to find an upper bound on the dimension of the matrices produced that depends on the Hirsch length of the group. Furthermore, we provide a new algorithm to provide the basis needed for these matrix representations that is slightly more efficient.

# Table of Contents

## List of Tables

**List of Figures**

# Chapter 1

# Group based cryptography

Here we follow the exposition of [13] and [35].

## 1.1 Introduction

Cryptology is the combination of two fields: Cryptography and Cryptanalysis. Cryptography is the science of devising and implementing secret codes or cryptosystems while cryptanalysis is the science of breaking cryptosystems. Due to the use of internet shopping, online banking, etc. in today's society there is a high demand for secure cryptosystems.

The public key cryptosystems and public key exchange protocols currently in use, such as the RSA algorithm, Diffie-Hellman, and elliptic curve methods, theoretically depend on the structure of abelian groups since they are number theory based. Although there have been no successful attacks on the standard protocols

there is a belief that the increased strength of computing machinery has made these protocols less secure. This has led to an active line of research, called non-commutative group based cryptography, to develop and analyze new cryptosystems and key exchange protocols based on non-commutative cryptographic platforms [35].

Most non-commutative cryptographic platforms proposed have been non-commutative groups. In these cryptosystems the algebraic properties of the platforms are used in both creating cryptosystems and in cryptanalysis of the suggested protocol. The difficulty, in terms of complexity, of certain algorithmic problems in finitely presented groups, such as the conjugacy search problem, plays a crucial role in encryption and decryption.

## 1.2 Public Key Cryptography

Encryption is the process of putting a plaintext message (uncoded message) into a ciphertext message (coded message), while the reverse process is called decryption. The plaintext and the ciphertext messages are divided into message units, $k$-tuples of letters, by the encryption algorithm. The transformations are then performed on these message units. Mathematically, we may write this as follows.

Let

$$\mathcal{P} = \{\text{set of plaintext message units}\} \text{ and}$$

$$\mathcal{C} = \{\text{set of ciphertext message units}\}.$$

The encryption algorithm is then an invertible function

$$f : \mathcal{P} \to \mathcal{C}$$

called the encryption map, and the decryption algorithm is

$$f^{-1} : \mathcal{C} \to \mathcal{P}$$

called the decryption map. The triple $\{\mathcal{P}, \mathcal{C}, f\}$ is called a cryptosystem.

As mentioned above, most cryptosystems in use today are number theory based. Generally, the plaintext message and the ciphertext message are written in an $N$-letter alphabet. In order to encrypt and decrypt messages, the alphabet is considered as $N$ integers modulo $N$ and a number theoretical function is performed on them. Typically, one uses a sequence of $k$ letters rather than a single letter when encrypting and decrypting messages. Here, the $k$ letters are a message unit and encryption is a function

$$f : \mathbb{Z}_N^k \to \mathbb{Z}_N^k.$$

There is a fundamental difference between *symmetric key* cryptography and *public key cryptography*. In symmetric key (classical) cryptography the method of encoding and decoding messages are known only to the sender and receiver, while in public key cryptography the method of encoding messages is public and only the receiver knows how to decode messages. Classical cryptosystems have the feature that if the method of encryption is known then the method of decryption is also known. Hence, in the classical setting once the encryption algorithm is known the decryption algorithm can be carried out in roughly the same amount of time as the encryption algorithm.

Public key cryptography is of great importance in today's society since internet banking and shopping have become mainstream. It is imperative to have secure public key cryptosystems to keep all internet transactions secure. While encryption is easy in public key cryptography, the decryption method is difficult. The idea of public key cryptography is based on a one-way function; that is, a function that is easy to implement but difficult to invert. Hence, if the encryption algorithm is a one-way function it becomes easy

to encrypt messages but difficult to decrypt them.

The standard model for a public key cryptosystem can be described as follows. Suppose Alice wants to send a message, $\mathcal{M}$, to Bob. The encryption maps for Alice and Bob, $f_A$ and $f_B$ respectively, are pubic information, while the decryption maps $f_A^{-1}$ and $f_B^{-1}$ are known only to Alice and Bob, respectively. Alice begins by sending $f_B f_A^{-1}(\mathcal{M})$ to Bob. Bob then decodes this message by first applying $f_B^{-1}$ to get $f_B^{-1}(f_B f_A^{-1}(\mathcal{M})) = f_A^{-1}(\mathcal{M})$. Then since $f_A$ is public knowledge Bob can apply $f_A$ to $f_A^{-1}(\mathcal{M})$ to get $f_A(f_A^{-1}(\mathcal{M})) = \mathcal{M}$, recovering Alice's message.

It is important for Bob to know that he is, in fact, receiving a message from Alice. For this reason Alice sends Bob $f_B f_A^{-1}(\mathcal{M})$ rather than simply $f_B(\mathcal{M})$. By applying $f_B f_A^{-1}$ to her message, Alice is providing a method of authentication for Bob. Otherwise, anyone can send Bob a message $\mathcal{N}$ by sending $f_B(\mathcal{N})$. Since $f_A^{-1}$ is known only to Alice, getting a reasonable message from $f_A(f_B^{-1} f_B f_A^{-1}(\mathcal{M}))$ would verify that the message came from Alice.

Since most public key cryptosystems in use today are number theory based, the one-way functions they are based on are number theoretic functions. Hence, the security of these cryptosystems is

based on the difficulty of inverting these functions. The first public key protocol was developed in 1976 by Diffie and Hellman. This protocol was based on the difficulty of the discrete log problem, which can be stated as follows:

*Discrete Log Problem*: Let $G$ be a finite group. Given an element $g \in G$ and $y = g^n$ for some $n \in \mathbb{Z}$, find such an $n$. We say the discrete log of $y$ with base $g$ is any number $j$ with $y = g^j$.

The Diffie-Hellman protocol is as follows:

Let $p$ be a prime and let $g \in \mathbb{Z}_p^*$ be a primitive root mod $p$. Both $g$ and $p$ are public information. Suppose Alice and Bob want to agree on a key $k$ with $1 < k < p - 1$.

1. Alice begins by randomly choosing an element $a \in \mathbb{Z}_p^*$, which she keeps secret. She then computes $g^a \mod p$ and makes this element public.

2. Similarly, Bob randomly chooses an element $b \in \mathbb{Z}_p^*$, which he keeps secret. He then computes $g^b \mod p$ and makes this element public.

3. Then the element $g^{ab} \mod p$ is computable for both Bob and Alice since $(g^b)^a \mod p$ is computable for Alice and $(g^a)^b \mod p$ is computable for Bob. Hence, the element $g^{ab}$ is the shared key $k$.

The security of this cryptosystem relies on the difficulty of the discrete log problem. If an adversary can solve the discrete log problem, then he or she can determine either Alice's secret element $a$ or Bob's secret element $b$ and thus recover the key since the elements $g$, $g^a$, and $g^b$ are public information. We will see later that Ko, Lee et al. created a non-commutative analog of this protocol using conjugation of elements rather than taking powers of elements.

Rivest, Adelman and Shamir developed the RSA algorithm in 1977, which is presently the public key cryptosystem most commonly implemented. The RSA algorithm relies on the difficulty of factoring large integers. The RSA algorithm is as follows.

Suppose Alice wants to send a message $\mathcal{P}$ to Bob.

1. Alice begins by randomly choosing two large primes $p_A, q_A$ and an integer $e_A$ that is relatively prime to $\Phi(p_A q_A) = (p_A - 1)(q_A - 1)$, which she keeps secret.

2. Alice then computes $n_A = p_A q_A$ and the multiplicative inverse $d_A$ of $e_A \mod \Phi(n_A)$. She makes public $K_A = (n_A, e_A)$ and the encryption algorithm

$$f_A(\mathcal{P}) = \mathcal{P}^{e_A} \mod n_A$$

where $\mathcal{P} \in \mathbb{Z}_{n_A}$ is a message unit. The decryption algorithm is then

$$f_A^{-1}(\mathcal{C}) = \mathcal{C}^{d_A} \mod n_A.$$

3. Similarly, Bob randomly chooses two large primes $p_B, q_B$ and an integer $e_B$ that is relatively prime to $\Phi(p_B q_B) = (p_B - 1)(q_B - 1)$, which he keeps secret.

4. Bob then computes $n_B = p_B q_B$ and the multiplicative inverse of $e_B \mod \Phi(n_B)$. Bob makes his public key $K_B = (n_B, e_B)$.

5. Alice can now send $f_B(f_A^{-1}(\mathcal{P}))$ to Bob. Bob can recover the message $\mathcal{P}$ by applying $f_A f_B^{-1}$ to the element $f_B(f_A^{-1}(\mathcal{P}))$ .

One possible attack an adversary can mount on this cryptosystem is to determine $f_B^{-1}$ or $f_A^{-1}$ based on the public information $K_B$, $K_A$, $n_B$ and $n_A$. If the adversary can factor $n_B$ or $n_A$ he or she would be able to recover the private keys. Since factoring a number is known to be difficult, this attack is not feasible. Currently, it is not known whether or not breaking this cryptosystem is equivalent to factoring a product of primes. There has been some evidence that breaking the RSA cryptosystem is not equivalent to factoring (see [6]). We would like to note that there are no non-commutative analogs of the RSA algorithm.

The last public key cryptosystem we would like to mention is the El-Gamal cryptosystem (see [15]). This cryptosystem modifies the Diffie-Hellman key exchange to perform encryption.

The protocol is as follows.

Let $p$ be a prime, $g$ a primitive root mod $p$, and $x \in \mathbb{Z}_p^*$.

1. Bob's public key is the triple $(p, g, z)$, where $z = g^x \mod p$, and his private key is $x$.

2. Alice encrypts a message $m$ with $0 \leq m \leq p-1$ by first choosing $y \in \mathbb{Z}_p^*$ and computing $k = g^y \mod p$ and $d = mz^y$. She then sends Bob $(k, d)$. $k$ is called the header.

3. Bob decrypts using his private key by computing $k^{-x}d \mod p = m$.

Decryption works since
$k^{-x}d = g^{-xy}d = g^{-xy}mz^y = g^{-xy}mg^{xy} = m \mod p$.

Kahrobaei and Khan created a non-commutative analog of the El-Gamal cryptosystem by replacing multiplication and taking powers by conjugation.

After a key is established between the two parties Alice and Bob, say using the Diffie-Hellman key exchange, symmetric key

cryptography is then used for encryption of messages for efficiency purposes. Following implementation of a key exchange protocol Alice and Bob have a shared key $k$, known only to them and would like to communicate openly using this key. Let $f_k$ be an encryption algorithm based on the key $k$. Suppose now that Alice wants to send the message $m$, given as a binary bit string, to Bob. Alice sends to Bob

$$f_k(m) \oplus k$$

where $k$ is a bit string for the key $k$ and $\oplus$ is addition modulo 2.

Bob knows the key $k$ and hence can compute it as a binary string. He now computes

$$f_k(m) \oplus k \oplus k$$

Since addition modulo 2 has order 2 we have

$$f_k(m) \oplus k \oplus k = f_k(m).$$

Bob now applies the decryption algorithm $f_k^{-1}$ to decode the message. In practice a *hash function* is usually applied to $k$ (see [8]).

## 1.3 Group based Public Key Exchanges

The study of group based non-abelian cryptography sprouted with the schemes proposed by Anshel-Anshel-Goldfeld [1] and Ko-Lee

et al. [29]. These schemes proposed using non-abelian groups and combinatorial group theory to create new public key exchanges. The security of these schemes is based on "hard" group theoretic problems.

If $G$ is a group and $g, h \in G$ we let $g^h$ denote the conjugate of $g$ by $h$; that is $g^h = h^{-1}gh$. By simply noting that conjugation behaves like ordinary exponentiation in that $(g^{h_1})^{h_2} = g^{h_1 h_2}$, one can mimic the Diffie-Hellman key exchange protocol by using conjugation. This idea is the basis of the Ko-Lee protocol.

The Anshel-Anshel-Goldfeld protocol and the Ko-Lee protocol use a platform group $G$ given by a group presentation. In both protocols it is assumed that the elements of $G$ have efficiently computable unique normal forms. In addition, it is assumed that given normal forms for $x, y \in G$ the normal form for the product $xy$ does not reveal $x$ or $y$. That is, there is an algorithm that transforms $xy$ into $NF(xy)$, where $NF(xy)$ denotes the normal form of $xy$, such that $xy =_G NF(xy)$, but this is hard to detect. This assumption is required to prevent an adversary from obtaining information about the group elements by inspection. The restriction of the normal forms allows for a good method of disguising group elements. Before describing these protocols we need the following definition.

**Definition 1.3.1** (Word Problem)**.** Given a finitely presented group $G$, does there exist an algorithm to decide whether or not a word in the generators is the trivial word?

By requiring that group elements have an efficiently computable normal form, we ensure that there is a solution to the word problem. One can solve the word problem by putting the word $w$ in normal form and checking whether or not $NF(w) =_G 1$.

We will begin by describing the protocol introduced by Ko, Lee et al. (see [29]). The protocol developed mimics the Diffie-Hellman key exchange protocol in a non-abelian group setting. The protocol is as follows.

Let $G$ be a group with efficiently solvable word problem. Recall that $g^h$ means the conjugate of $g$ by $h$. The group $G$, an element $g \in G$, and commuting subgroups $A, B \leq G$ are made public.

1. Alice begins by choosing an element $a \in A$, which she keeps secret. She then computes $g^a$ and sends this to Bob.

2. Similarly, Bob chooses an element $b \in B$, which he keeps secret. He then computes $g^b$ and sends this to Alice.

3. Since the elements $a$ and $b$ were chosen from commuting subgroups $(g^a)^b = (g^b)^a = g^{ab}$ is computable for both Alice and Bob. Hence, $g^{ab}$ is the shared key.

The security of this protocol relies on the following group theoretic problem.

**Definition 1.3.2** (Search Conjugacy Problem)**.** Let $G$ be a recursively presented group. Given elements $g, h \in G$ with $h = g^x$ find such a conjugator.

If an adversary can solve the search conjugacy problem, then he or she would recover the shared key. To see this we proceed as follows. Suppose the adversary recovers an element $x \in G$ such that $g^x = g^a$ and $xb = bx$ for all $b \in B$. The adversary can then compute the shared key by computing $(g^b)^x = (g^x)^b = g^{ab}$. Hence, the security of this protocol relies on the difficulty of the search conjugacy problem in the proposed platform group. We would like to note that the search conjugacy problem is always recursively solvable in a recursively presented group since one can recursively enumerate all conjugates of an element (see [40]).

Next we will describe the Anshel-Anshel-Goldfeld public key exchange protocol (see [1]). This protocol, unlike the Ko-Lee protocol, doesn't require commuting subgroups or elements. The protocol is as follows.

Let $G$ be a group with efficiently solvable word problem. The group $G$, two finitely generated subgroups $A, B \leq G$ and their

respective generators $a_1, \cdots, a_n$ and $b_1, \cdots, b_m$ are made public.

1. Alice begins by choosing an element $a \in A$, which she keeps secret. She then sends $b_1^a, \cdots, b_m^a$ to Bob.

2. Similarly, Bob chooses an element $b \in B$ which he keeps secret. He then sends $a_1^b, \cdots, a_n^b$ to Alice.

3. The shared key is then $[a, b] = (b^{-1})^a b = a^{-1}a^b$.

Alice can determine $a^b$ since she knows $a$ in terms of the generators of her subgroup $A$ and she knows the conjugates of the generators by $b$ from the public information. Hence, Alice can determine $[a, b] = a^{-1}a^b$. Similarly Bob can determine $[a, b] = (b^a)^{-1}b$.

At first glance it seems the security of this scheme relies on the difficulty of solving the following group theoretic problem.

**Definition 1.3.3** (Simultaneous Conjugacy Search Problem)**.** [35] Given $u_i, v_i \in G$ with $u_i^x = v_i$ for $1 \leq i \leq n$, find such a conjugator.

If an adversary were to solve the simultaneous conjugacy search problem, this would not be enough to obtain the shared key. One should note that not only does the adversary need to find such a conjugator, but he needs to find the conjugator in the generators of one of the subgroups. Hence, in addition the adversary would also have to solve the following problem.

**Definition 1.3.4** (Membership Search Problem)**.** [35] Given elements $x, a_1, \cdots, a_n$ in a group $G$, find an expression (if it exists) of $x$ as a word in $a_1, \cdots, a_n$.

Hence, the security of the Anshel-Anshel-Goldfeld scheme relies on the difficulty of the following problem.

**Definition 1.3.5** (Simultaneous Conjugacy Search Problem relative to a subgroup)**.** [35] Given $u_i, v_i \in G$ and a finitely generated subgroup $A$ of $G$ such that the system $u_i^x = v_i$, $i = 1, \cdots, n$ in $A$, find a solution.

Anshel, Anshel, Goldfeld and Ko and Lee suggested Braid Groups as potential platforms which led to the development of braid group cryptography. There have been various attacks on braid group cryptosystems that, in turn, led to the discovery of some interesting properties of braid groups. For more on braid groups and their applications to group based cryptography see [35].

The last non-commutative protocol we will describe is a non-commutative analog of the El Gamal public key cryptosystem based on the search conjugacy problem proposed by Kahrobaei and Khan (see [28]). The protocol is as follows.

Let $G$ be a group with efficiently solvable word problem, and let $S, T \leq G$ be two commuting subgroups of $G$.

Suppose Alice wants to send $x \in G$ as a session key to Bob. Then,

1. Bob begins by choosing an element $s \in S$, which he keeps secret, and an arbitrary element $b \in G$. Bob publishes $b$ and $c = b^s$.

2. Alice chooses a random $t \in T$ and sends $E = x^{(c^t)}$ to Bob along with the header $h = b^t$.

3. Bob then calculates $(b^t)^s = (b^s)^t = c^t$.

4. Now, Bob may calculate $E' = (c^t)^{-1}$, allowing him to decrypt the session key since $(x^{(c^t)})^{E'} = (x^{(c^t)})^{(c^t)^{-1}} = x$.

In order for this scheme to be feasible one must be able to compute products and inverses in $G$ efficiently. The security of this protocol relies on the difficulty of the search conjugacy problem. If an adversary can solve the search conjugacy problem for $G$ he will be able to determine Bob's private key $s$ or Alice's private key $t$. With this information an adversary can recover the secret. Hence, in order for this cryptosystem to be secure the search conjugacy problem must be "hard."

In Chapter 4 another key exchange protocol based on the following group theoretic problem is proposed.

**Definition 1.3.6** (Endomorphism Search Problem)**.** ([41]) Given elements $\phi(g)$ and $g$ in a group $G$, where $\phi$ is an endomorphism of $G$, find $\phi$.

The security of the protocol proposed in Chapter 4 is based on the fact that in the appropriate setting this problem is, in fact, NP-hard (see Chapter 2 for background on computational complexity).

## 1.4 Generic Complexity

When discussing the cryptanalysis of these group theoretic cryptographic protocols the computational complexity of the algorithm used to solve the problem at hand is important. The problem may be hard (exponential time) or undecidable on certain inputs while it is actually easy (polynomial time) on most inputs. For example, the conjugacy search problem in braid groups is hard on some inputs but is easy for many inputs. Typically in computer science one measures the worst case complexity of an algorithm. The worst case complexity gives an upper bound for the running time of the algorithm. Hence, although the worst case complexity of an algorithm could be exponential time the algorithm could be run in polynomial time on most inputs. Generic complexity refers to the computational complexity of an algorithm over most inputs.

We refer the reader to [35] for more on generic complexity and its use in cryptanalysis.

The reason some braid group cryptographic protocols are deemed insecure is because random subgroups of $B_n$ are generically free. Before we can describe the notion of generically free, we must review the notion of asymptotic density. Asymptotic density is a method to compute probabilities on infinite discrete sets in which each outcome is assumed to be equally likely. This method can also be used if there is a probability distribution on the elements. We refer the reader to the paper by Borovik, Myasnikov and Shpilrain [7] for a general description of this method in group theory.

Let $\mathcal{P}$ be a group property and let $G$ be a finitely generated group. The goal is to determine the measure of the set of elements that satisfy the property $\mathcal{P}$. For every $n \in \mathbb{N}$ let $B_n$ denote the $n$-ball in $G$. Let $|B_n|$ denote the size of $B_n$ or the measure of $|B_n|$ if a distribution has been placed on the elements of $G$. Let $S$ be the set of elements in $G$ satisfying $\mathcal{P}$. The asymptotic density of $S$ is then

$$\lim_{n\to\infty} \frac{|S \cap B_n|}{|B_n|}$$

provided this limit exists.

**Definition 1.4.1.** [35] The property $\mathcal{P}$ is called generic in $G$ if the

asymptotic density of the set $S$ of elements satisfying $\mathcal{P}$ is one.

**Definition 1.4.2.** [35] The property $\mathcal{P}$ is called an asymptotically visible property if the asymptotic density of the set $S$ of elements satisfying $\mathcal{P}$ is strictly between 0 and 1.

**Definition 1.4.3.** [35] The property $\mathcal{P}$ is called negligible if the asymptotic density of the set $S$ of elements satisfying $\mathcal{P}$ is 0.

If $\mathcal{P}$ is a group property and $G$ is a group then we say that subgroups of $G$ are generically $\mathcal{P}$ if a generic randomly chosen subgroup $H$ of $G$ has property $\mathcal{P}$. Equivalently this means that the asymptotic density of subgroups $H$ of $G$ that have property $\mathcal{P}$ is one.

We are now in a position to describe the notion of a group being generically free.

**Definition 1.4.4.** [4] A group $G$ has the generic free group property if a finitely generated subgroup is generically a free group.

**Definition 1.4.5.** [4] A group $G$ has the strong generic free group property if given randomly chosen elements $g_1, ..., g_n$ in $G$ then generically they are a free basis for the free subgroup they generate.

In [36] Myasnikov and Ushakov showed that pure braid groups $P_n$ with $n \geq 3$ have the strong generic free group property. In [9]

Carstensen, Fine and Rosenberger showed that all Fuchsian groups of finite co-volume and all braid groups $B_n$ with $n \geq 3$ have the strong generic free group property. The result of Myasnikov and Ushakov on pure braid groups has applications to the cryptanalysis of both the Ko-Lee cryptosystem and the Anshel-Anshel-Goldfeld cryptosystem (see [36]). These cryptosystems were susceptible to length based attacks because random choices of subgroups of braid groups are actually free groups, and free groups are susceptible to these attacks.

# Chapter 2

# Computational Complexity

Here, we will review basic facts about computational complexity.

## 2.1 Turing Machines and Complexity Classes

In this section we follow the exposition of [2].

When concerned with the notion of computational efficiency, one must consider what model of computation is being used since "efficiently computable" can depend on the hardware of the computer being used. The mathematical model used to study questions about computational complexity is the Turing machine because it simulates all realistic computation methods with a minor loss of efficiency.

Informally a Turing machine can be described as follows. Let $f$ be a function that takes a bit string as an input and outputs either 0 or 1. An algorithm for computing $f$ is a set of rules that

allow one to compute $f(x)$ given any input $x \in \{0,1\}^*$. Each of the rules consists of one or more of the following operations:

1. Read a bit of the input.
2. Read a bit from the scratch pad or workspace the algorithm is allowed to use.

Depending on the values that are read,

1. Write a bit on the scratch pad.
2. Either stop and output 0 or 1, or choose a new rule from the set that will be applied next.

The running time is the number of basic operations performed. We say that a Turing machine runs in time $T(n)$ if it performs at most $T(n)$ basic operations on inputs of length $n$.

Before formally defining a Turing machine, we must begin with some definitions.

**Definition 2.1.1.** [2] A tape is an infinite one directional line of cells. Each cell can hold a symbol from a finite set $\Gamma$ called the alphabet of the machine.

**Definition 2.1.2.** [2] A tape head can read or write symbols to the tape one cell at a time. Each tape comes with a tape head.

**Definition 2.1.3.** [2] A scratch pad simply consists of $k$ tapes.

The Turing machine's computation is divided into time steps in which the head can move to the left or right one cell in each time step. The machine's head can only read symbols from the first tape of the machine, which is designated the input tape. This is referred to as the machine's read-only head. The $k-1$ remaining read-write tapes are referred to as work tapes, and the last tape is designated the output tape. This is the tape in which the machine writes its final answer before halting. The machine has a register that holds a single element from the finite set of states, $Q$, associated with the machine. The state determines the next action to be taken by the machine, which consists of

1. read the symbols in the cells directly under the $k$ heads
2. replace each symbol with a new symbol or re-write the old symbol for the $k-1$ read-write tapes
3. change the register to contain another state from $Q$ or choose the same state again
4. move each head one cell to the left, right, or stay in place.

We are now in a position to formally define a deterministic $k$-tape Turing machine.

**Definition 2.1.4.** [2] A $k$-tape Turing machine $M$ is a tuple $(\Gamma, Q, \delta)$ consisting of:

- A finite set $\Gamma$ of the symbols that $M$'s tape can contain. $\Gamma$ contains a dedicated blank symbol, a designated start symbol, and the numbers 0 and 1. $\Gamma$ is called the alphabet of $M$.
- A finite set $Q$ of possible states that $M$ can be in. $Q$ contains a designated start state, $q_{start}$, and a designated halting state, $q_{halt}$.
- A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, called the transition function of $M$, where $k \geq 2$ that describes the rules $M$ uses in each step.

If the machine is in state $q \in Q$ and $(\sigma_1, \cdots, \sigma_k)$ are being read in the $k$ tapes, and $\delta(q, (\sigma_1, \cdots, \sigma_k)) = (q', (\sigma_1', \cdots, \sigma_k'), z)$, where $z \in \{L, S, R\}^k$, then in the next step the symbols $(\sigma_1, \cdots, \sigma_k)$ will be replaced by $(\sigma_1', \cdots, \sigma_k')$, the state will be $q'$, and the $k$ heads will move **L**eft, **R**ight, or **S**tay in place as is given by $z$.

Since we are concerned with running times of algorithms, we must formalize this notion. When counting the number of basic steps required for a computation, we must consider this as a function of the input length.

**Definition 2.1.5.** [2] Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let $M$ be a Turing machine. We say that $M$ computes $f$ if for every $x \in \{0,1\}^*$, whenever $M$ is initialized to the start configuration on input $x$, then it halts with $f(x)$ written on its output tape. We say $M$ computes $f$ in $T(n)$-time if its computation on every input $x$ takes at most $T(|x|)$ steps.

**Definition 2.1.6.** [2] A complexity class is a set of functions that can be computed within given resource bounds.

**Definition 2.1.7.** [2] A Boolean function is a function with only one bit of output. These define decision problems or languages.

**Definition 2.1.8.** [2] A machine decides a language $L \subseteq \{0,1\}^*$ if it computes the function $f_L : \{0,1\}^* \to \{0,1\}$, where $f_L(x) = 1 \Leftrightarrow x \in L$.

**Definition 2.1.9.** [2] Let $T : \mathbb{N} \to \mathbb{N}$ some function. A language $L$ is in $\mathbf{DTIME}(T(n))$ if and only if there is a deterministic Turing machine that runs in time $c \cdot T(n)$ for some constant $c > 0$ and decides $L$.

We are now in a position to formally define the notion of "efficiently computable." Generally when one thinks of "efficiently computable" one thinks of polynomial running time; that is, the

running time is at most $n^c$ for some constant $c > 0$. The class **P** (**P** for polynomial) captures this notion.

**Definition 2.1.10.** [2] $\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c)$.

We would also like to formally define the notion of "efficiently verifiable solutions". The class **NP** formalizes this notion.

**Definition 2.1.11.** [2] A language $L \subseteq \{0,1\}^*$ is in **NP** is there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time Turing machine (called the verifier for $L$) such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \, s.t. M(x,u) = 1.$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call $u$ a certificate for $x$.

## 2.2 Reductions and Complete Problems

In this section we follow the exposition of [35].

Here we will describe what it means for one problem to be as hard as another by using the notion of a reduction. The intuitive idea of a reduction is as follows: A problem $D_1$ is reducible to a problem $D_2$ if a decision algorithm for $D_2$ gives a decision algorithm for $D_1$. This notion of a reduction allows one to estimate

the difficulty of one problem by comparing it to the difficulty of another.

**Definition 2.2.1.** [35] Let $F$ be a set of functions from $\mathbb{N}$ to $\mathbb{N}$, and $D_1$ and $D_2$ be decision problems. The problem $D_1$ is called reducible to $D_2$ under $F$ if there exists a function $f \in F$ such that

$$x \in D_1 \Leftrightarrow f(x) \in D_2.$$

We write $D_1 \leq_F D_2$.

**Definition 2.2.2.** [35] A problem $C$ is called hard for a set of decision problems $S$ if for any $D \in S$ we have $D \leq_F C$.

**Definition 2.2.3.** [35] A problem $C$ is called comlplete for a set of decision problems $S$ if it is hard for $S$ and $C \in S$.

**Definition 2.2.4** (Turing Reducible)**.** [35] A problem $D_1$ is Turing reducible to a problem $D_2$, written as $D_1 \leq_T D_2$, if $D_1$ is computable by a Turing machine with an oracle for $D_2$.

**Definition 2.2.5** (Ptime Reduction)**.** [35] Let $D_1$ and $D_2$ be decision problems. We say that $f : D_1 \to D_2$ Ptime reduces $D_1$ to $D_2$, written $D_1 \leq_P D_2$, if

- $f$ is polynomial time computable and
- $x \in D_1$ if and only if $f(x) \in D_2$.

We are now in a position to define the classes NP-hard and NP-complete.

**Definition 2.2.6** (NP-hard)**.** [35] A problem $D$ is called NP-hard if for every problem $D^{'} \in NP$ there exists a polynomial time reduction of $D^{'}$ to $D$.

**Definition 2.2.7** (NP-complete)**.** [35] A problem is called NP-complete if it belongs to NP and is NP-hard.

NP-hard problems are important in cryptography as some NP-hard problems are the basis for the security of certain cryptographic protocols (see[20]).

# Chapter 3

# Platform Groups

## 3.1 Introduction

As new group based cryptographic protocols are being proposed it is imperative that these protocols are implemented in the appropriate platform group. A problem can be difficult to solve in one group but hard to solve in another. For example, although all cyclic groups of the same order are isomorphic the complexity of the discrete log problem depends on the presentation of the group. While there are no known polynomial time algorthims to solve the discrete log problem in multiplicative cyclic groups, there is a polynomial time algorithm to solve the discrete log problem in additive cyclic groups. To see this, suppose we are given an additive cyclic group, $G$, of order $n$ with generator $g$. The discrete log problem written additively is as follows. Given an element $y = kg \in G$ find

an element $h$ such that $y = hg \mod n$. This is easily solved by computing $g^{-1}y = h \mod n$, which is efficiently computable [44].

Here we describe a few potential platform groups for these new cryptographic protocols. Polycyclic groups have been proposed in [28], [10] and [12] since it is conjectured that there is no subexponential time algorithm that solves search conjugacy problem in these groups. We review basic facts about nilpotent groups simply because finitely generated nilpotent groups are polycyclic. Free metablelian groups are proposed as a platform group in [20] due to the difficulty of the endomorphism search problem in these groups. Small cancellation groups are discussed due to the efficiency of solving the word problem using Dehn's algorithm. This property is essential in the secret sharing scheme proposed in [21]. Lastly, matrix groups are discussed as the elements in these groups are easily diffused by performing matrix multiplication.

## 3.2 Polycyclic Groups

Here we follow the exposition of [24].

**Definition 3.2.1.** [24] A group is called polycyclic if it admits a finite subnormal series

$$G = G_1 \triangleright G_2 \triangleright G_3 \triangleright \cdots \triangleright G_{n+1} = 1$$

where each $G_i/G_{i+1}$ is cyclic. The chain of subgroups

$$G = G_1 \triangleright G_2 \triangleright G_3 \triangleright \cdots \triangleright G_{n+1} = 1$$

is called a polycyclic series.

**Definition 3.2.2.** [24] Since each factor is cyclic there exists an $x_i \in G$ such that $\langle x_i G_{i+1} \rangle = G_i/G_{i+1}$. We call the sequence $X = [x_1, x_2, \cdots, x_n]$ a polycyclic sequence for $G$.

**Definition 3.2.3.** [24] Let $X$ be a polycyclic sequence for a polycyclic group $G$. The sequence of relative orders of $X$ is the sequence $R(X) = (r_1, \cdots, r_n)$ where $r_i = [G_i : G_{i+1}] \in \mathbb{N} \cup \infty$. We denote the set of indices in which $r_i$ is finite by $I(X)$.

**Definition 3.2.4.** [24] The number of infinite entries in the sequence $R(X)$ is called the Hirsch length of the group; that is, the Hirsch length is the number of infinite factors in the polycyclic series. The Hirsch length is independent of the polycyclic series chosen.

**Example 3.2.1.** *[24] Let* $G := \langle (1,2,3,4), (1,3) \rangle \cong D_8$*, the dihedral group of order 8.*

- *The subnormal series*

$$G = G_1 \triangleright G_2 \triangleright G_3 = 1,$$

*where* $G_2 := \langle (1,2,3,4) \rangle \cong \mathbb{Z}_4$, *is a polycyclic series for* $G$. *Some polycyclic sequences defining this polycyclic series are* $X := [(1,3),(1,2,3,4)]$ *and* $Y := [(2,4),(1,4,3,2)]$. *The sequences of relative orders are* $R(X) = R(Y) = (2,4)$ *and the indices in which* $r_i$ *is finite are* $I(X) = I(Y) = \{1,2\}$.

- *The subnormal series*

$$G = G_1 \triangleright G_2 \triangleright G_3 \triangleright G_4 = 1,$$

*where* $G_2 := \langle (1,2)(3,4),(1,3)(2,4) \rangle \cong V_4$ *and*
$G_3 := \langle (1,3)(2,4) \rangle \cong Z_2$, *is a polycyclic series for* $G$. *Some polycyclic sequences defining this polycyclic series are*
$X := [(2,4),(1,2)(3,4),(1,3)(2,4)]$ *and*
$Y := [(1,2,3,4),(1,2)(3,4),(1,3)(2,4)]$. *The sequences of relative orders are* $R(X) = R(Y) = (2,2,2)$ *and the indices in which* $r_i$ *is finite are* $I(X) = I(Y) = \{1,2,3\}$.

**Example 3.2.2.** *[24] Let* $G := \langle a, b \rangle$ *where* $a = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ *and* $b = \begin{pmatrix} -1 & -1 \\ 0 & 1 \end{pmatrix}$. *We then have that* $G \cong D_\infty$, *the infinite dihedral group. Here we give some polycyclic sequences for* $G$:

- $X := [a, ab]$ *is a polycyclic sequence for* $G$ *with sequence of*

*relative orders* $R(X) = (2, \infty)$ *and the indices in which* $r_i$ *is finite is* $I(X) = \{1\}$.

- *We would like to note that* $G$ *has infinitely many polycyclic sequences by observing the following. Let* $l \in \mathbb{N}$ *and* $n_i \in \mathbb{N}$ *with* $n_i \neq 1$ *for* $1 \leq i \leq l$. *Then* $Y = [a, ab, (ab)^{n_1}, \cdots, (ab)^{n_1 n_2 \cdots n_l}]$ *is a polycyclic sequence for* $G$ *with sequence of relative orders* $R(Y) = (2, \infty, n_1, \cdots, n_l)$.

**Definition 3.2.5.** [24] Let $X = [x_1, \cdots, x_n]$ be a polycyclic sequence for $G$ and $R(X) = (r_1, \cdots, r_n)$ be its sequence of relative orders. Then every $g \in G$ can be written in the form $g = x_1^{e_1} \cdots x_n^{e_n}$ with $e_i \in \mathbb{Z}$ and $0 \leq e_i < r_i$. This expression is called the normal form of $G$ with respect to $X$.

**Definition 3.2.6.** [24] A presentation of the form

$$\left\langle a_1, \cdots, a_n | a_i^{a_j} = w_{ij}, a_i^{a_j^{-1}} = v_{ij}, a_k^{s_k} = u_{kk} \text{ for } k \in I \right\rangle$$

for $1 \leq j < i \leq n$ where $s_i \in \mathbb{N} \cup \infty$, $s_i < \infty$ if $i \in I \subseteq \{1, 2, \cdots, n\}$ and $w_{ij}, v_{ij}, u_{jj}$ are words in the generators $a_{j+1}, \cdots, a_n$ is called a polycyclic presentation. Relations of the form $a_i^{a_j^{-1}} = v_{ij}$ and $a_i^{a_j} = w_{ij}$ are called conjugacy relations and relations of the type $a_k^{s_k} = u_{kk}$ are called power relations. The sequence $S =$

$(s_1, \cdots, s_n)$ is called the sequence of power exponents of the presentation.

**Theorem 3.2.3.** *[24] Every polycyclic sequence defines a unique polycyclic presentation. Therefore every polycyclic group can be defined by a polycyclic presentation.*

**Theorem 3.2.4.** *[24] Let $\langle a_1, \cdots, a_n | R\rangle$ be a polycyclic presentation and let $G$ be the group defined by this presentation. Then $G$ is polycyclic and $X = [a_1, \cdots, a_n]$ is a polycyclic sequence for $G$. Furthermore its sequence of relative orders $R(X) = (r_1, \cdots, r_n)$ satisfy $r_i \leq s_i$ for $1 \leq i \leq n$.*

**Example 3.2.5.** *[24] Let $G$ be the group defined by the following polycyclic presentation:*

$$\left\langle a_1, a_2, a_3 | a_1^3 = a_3, a_2^2 = a_3, a_2^{a_1} = a_2 a_3, a_2^{a_1^{-1}} = a_2 a_3 \right\rangle.$$

By convention the trivial conjugacy relations $a_3^{a_1^{-1}} = a_3^{a_1} = a_3^{a_2^{-1}} = a_3^{a_2} = a_3$ are omitted from the presentation although they hold. From *Theorem 3.2.4* we know that $X = [a_1, a_2, a_3]$ is a polycyclic sequence for $G$ and that $R(X) \leq (3, 2, \infty)$.

**Definition 3.2.7.** [24] A polycyclic presentation $\langle X | R\rangle$ is called consistent if the sequence of power exponents $S = (s_1, \cdots, s_n)$ is the sequence of relative orders $R(X) = (r_1, \cdots, r_n)$; that is,

the presentation is consistent if $s_i = [G_i : G_{i+1}]$ for each $i \in \{1, 2, \cdots, n\}$.

**Theorem 3.2.6.** *[24] Every polycyclic sequence determines a consistent polycyclic presentation. Thus every polycyclic group can be defined by a consistent polycyclic presentation.*

The normal form for an element in a group given by a consistent polycyclic presentation can be determined via the *collection algorithm.* Thus, the collection algorithm gives a solution to the word problem for a group $G$ given by a consistent polycyclic presentation. The collection algorithm works by iteratively applying the power and conjugacy relations of the presentation until the normal form for a word is obtained (see Figure 3.1). The nature of the power and conjugacy relations give rise to the following theorem.

**Theorem 3.2.7.** *[24] The collection algorithm terminates.*

The collection algorithm is currently the best known way to obtain the normal form of an element in a polycyclic group.

**Definition 3.2.8.** [24] Let $X$ be a generating set, and $w = x_1^{e_1} \cdots x_k^{e_k}$ for $e_i \in \mathbb{Z}/\{0\}$. Let $R(I) = \{r_1, \cdots, r_n\}$ be the sequence of relative orders.

1. A word $w$ is collected if $w = x_{i_1}^{e_1} \cdots x_{i_n}^{e_n}$ with $i_1 < i_2 < \cdots < i_n$ and $e_j < r_j$; that is, if $w$ is in normal form. Otherwise $w$ is *uncollected*.

2. A subword $v$ of $w$ in the generators is a *minimal uncollected subword* if

   - $v = x_j^{e_j} x_{i_{j+1}}$ or $v = x_j^{e_j} x_{i_{j+1}}^{-1}$ for $i_j > i_{j+1}$.
   - $v = x_{i_j}^{e_j}$ for $r_{i_j} \neq \infty$ and $e_j > r_{i_j}$.

Figure 3.1: Collection Algorithm, [24]

**Input:** A polycyclic presentation $\langle X|R\rangle$ and a word $w$ in $X$.

**Output:** A collected word equivalent to $w$.

1. **while** there is a minimal uncollected subword of $w$
2. **do** choose a minimal uncollected subword $v$ in $w$
3. **if** $v = x_{i_j}^{e_j} x_{i_{j+1}}$ with $i_j > i_{j+1}$
4. **then** replace $v$ by $x_{i_{j+1}} w_{i_j,i_{j+1}}^{e_j}$ in $w$;
5. **if** $v = x_{i_j}^{e_j} x_{i_{j+1}}^{-1}$ with $i_j > i_{j+1}$
6. **then** replace $v$ by $x_{i_{j+1}}^{-1} w_{i_{j+1},i_j}^{e_j}$ in $w$;
7. **if** $v = x_{i_j}^{e_j}$ with $e_j > r_j$
8. **then** replace $v$ by $x_{i_j}^{m} w_{i_j,i_j}^{q}$ in $w$ where $e_j = qr_{i_j} + m$ with $0 \leq m < r_{i_j}$;
9. **return** $w$;

Polycyclic groups were suggested as platform groups for crytographic protocols by D. Kahrobaei and B. Eick (see [12]), D. Karobaei and M. Anshel (see [10]), and D. Kahrobaei and B. Khan (see [28]). Polycyclic groups have been suggested as a potential platform group since the word problem in groups given by a consistent presentation is efficiently solvable and the search conjugacy problem is conjectured to be exponential time. We note that the search conjugacy problem in a polycyclic group is always decidable due to the following facts.

- It is a well known fact, due to L. Auslander (see [30]), that every polycyclic group is linear; that is, every polycyclic group $G$ can be embedded into $GL_n(\mathbb{Z})$ for an appropriate $n$.
- In $GL_n(\mathbb{Z})$ group multiplication and the word problem are efficiently solvable (see [28]).
- The search conjugacy problem in any subgroup of $GL_n(\mathbb{F})$ is solvable (see [28]).

By utilizing an embedding, $\rho : G \to GL_n(\mathbb{Z})$, from a polycyclic group $G$ into $GL_n(\mathbb{Z})$, one may solve the search conjugacy problem for $G$ in the group $GL_n(\mathbb{Z})$. Implementing an algorithm that provides an embedding of a polycyclic group into $GL_n(\mathbb{Z})$ with

an algorithm to solve the search conjugacy problem in $GL_n(\mathbb{Z})$ provides an algorithm to solve the search conjugacy problem for polycyclic groups.

## 3.3 Nilpotent Groups

Here we follow the exposition of [24]. For more on nilpotent groups see [22].

**Definition 3.3.1.** [24] A normal series

$$G = G_0 \triangleright G_1 \triangleright G_2 \triangleright \cdots \triangleright G_r = 1$$

of a group $G$ is called a central series if $G_{i-1}/G_i \leq Z(G/G_i)$ for $1 \leq i \leq r$, where $Z(H)$ denotes the center of a group $H$.

**Definition 3.3.2.** [24] A group $G$ is called nilpotent if it has a central series, and the length $r$ of the shortest central series of $G$ is called the nilpotentency class of $G$.

**Definition 3.3.3.** [24] The lower central series of a group $G$ is the series

$$G = \gamma_1(G) \triangleright \gamma_2(G) \triangleright \gamma_3(G) \triangleright \cdots \triangleright \gamma_i(G) \triangleright \gamma_{i+1}(G) \triangleright \cdots ,$$

where $\gamma_{i+1}(G) := [G, \gamma_i(G)]$.

**Definition 3.3.4.** [24] A group $G$ is nilpotent if and only if it has a finite lower central series.

**Definition 3.3.5.** [24] The upper central series of a group $G$ is the series

$$1 = Z_0(G) \triangleleft Z_1(G) = Z(G) \triangleleft Z_2(G) \triangleleft \cdots Z_i(G) \triangleleft Z_{i+1}(G) \triangleleft \cdots ,$$

where each $Z_{i+1}(G)$ for $i \geq 1$ is $Z_{i+1}/Z_i(G) = Z\left(G/Z_i(G)\right)$.

**Definition 3.3.6.** [24] A group $G$ is nilpotent if and only if its upper central series terminates with $Z_r(G) = G$.

**Example 3.3.1.** *[27] The Heisenberg group $H$, a finitely generated nilpotent group of class 2, is the group of matrices of the form*

$$\begin{pmatrix} 1 & r & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix},$$

*where $r, s, t \in \mathbb{Z}$. If we set*

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

*then one can show that*

$$A^k = \begin{pmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, B^l = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & l \\ 0 & 0 & 1 \end{pmatrix}, C^m = \begin{pmatrix} 1 & 0 & m \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

*for any $k, l, m \in \mathbb{Z}$. Hence, each element of the group $H$ is equal to one of the matrices*

$$A^k B^l C^m = \begin{pmatrix} 1 & k & m+kl \\ 0 & 1 & l \\ 0 & 0 & 1 \end{pmatrix}.$$

*In addition we have the relations,*

$$[A, B] = C, [C, B] = [C, A] = I.$$

*From this, we get the following presentation for the Heisenberg group $H$:*

$$\langle a, b, c | [a, b] = c, [c, b] = [c, a] = 1 \rangle .$$

For our purposes, we are interested in a specific type of nilpotent group: finitely generated nilpotent groups. We are interested in finitely generated nilpotent groups since they are in fact polycyclic, and admit a polycyclic presentation. Hence, the same properties mentioned in the previous section hold.

**Definition 3.3.7.** [24] A polycyclic presentation $\langle x_1, \cdots, x_n | R \rangle$ with power exponents $S$ is called a nilpotent presentation if the conjugacy relations are of the form:

$$x_i^{x_j} = x_i x_{i+1}^{b_{i,j,i+1}} \cdots x_n^{b_{i,j,n}} \text{ for } 1 \leq j < i \leq n,$$

$$x_i^{x_j^{-1}} = x_i x_{i+1}^{c_{i,j,i+1}} \cdots x_n^{c_{i,j,n}} \text{ for } 1 \leq j < i \leq n$$

where $b_{i,j,k}, c_{i,j,k} \in \mathbb{Z}$ for $1 \leq j < i < k \leq n$.

**Lemma 3.3.2.** *[24]*

1. *Every finitely generated nilpotent group has a consistent nilpotent presentation defining it.*

2. *Every consistent nilpotent presentation defines a finitely generated nilpotent group.*

3. *If $G$ is a finitely generated torsion free nilpotent group, then there exists a consistent nilpotent presentation defining $G$ whose power exponents are all infinite. The number of generators in such a presentation is minimal for a polycyclic presentation of $G$.*

**Example 3.3.3.** *[24] Let $G : \langle (1,2,3,4), (1,3) \rangle \cong D_8$. Then $G$ is nilpotent with nilpotent presentation*

$$\left\langle x_1, x_2, x_3 | x_1^2 = x_3^2 = 1, x_2^2 = x_3, x_2^{x_1} = x_2 x_3, x_2^{x_1^{-1}} = x_2 x_3 \right\rangle.$$

## 3.4 Free Metabelian Groups

Here we follow the exposition of [35] and [42].

**Definition 3.4.1.** [35] Let $F$ be a free group and let $R$ be a normal subgroup of $F$. The factor group $F/R$ is called relatively free if $R$ is fully invariant; that is, if $\alpha(R) \leq R$ for any endomorphism $\alpha$ of $F$. If $x_1, \ldots, x_n$ are free generators of $F$, then $x_1R, \ldots, x_nR$ are called relatively free generators of $F/R$.

**Definition 3.4.2.** [35] Let $F_n$ be the free group of rank $n$. The relatively free group $F_n/F_n''$, where $F_n''$ denotes the second commutator subgroup, is called the free metabelian group of rank $n$. This group is denoted $M_n$.

If no ambiguity arises, we denote the relatively free generators, $x_1R, \ldots, x_nR$, of $F/R$ by $x_1, \ldots, x_n$. Let $\mathcal{F}_n = F_n/R$ for some fully invariant subgroup $R$. We would like to note that any map on the generators into $\mathcal{F}_n$ can be extended to an endomorphism of $\mathcal{F}_n$.

Before discussing the normal form for this proposed platform group, we would like to introduce *Nielson automorphisms*. Let $X = \{x_1, \cdots, x_n\}$ be a set of generators of $F_n$ and let $\alpha_j, \beta_{jk} : X \to F_n$ be the maps given by

$$\alpha_j : x_i \mapsto \begin{cases} x_i^{-1} & \text{if } i = j \\ x_i & \text{if } i \neq j \end{cases}$$

and

$$\beta_{jk} : x_i \mapsto \begin{cases} x_i x_j & \text{if } i = k \\ x_i & \text{if } i \neq k, \end{cases}$$

where $1 \leq i, j, k \leq n$. The $\alpha_j$'s and the $\beta_{jk}$'s are called Nielson automorphisms, and they generate the automorphism group $Aut(F_n)$. Nielsen automorphisms can be defined in the same fashion for any relatively free group $\mathcal{F}_n$. The $\alpha_j$'s and the $\beta_{jk}$'s of a relatively free group generate a subgroup of the automorphism group of $\mathcal{F}_n$ called the group of tame automorphisms.

The free metabelian group of rank $n$, $M_n$, has a normal form as well as a seminormal form which is not unique if $n > 2$. The seminormal form can be used for transmissions as it can easily be converted back to a word representing the transmitted element. Since the seminormal form is not unique for $n > 2$ it cannot be used as a shared key in a public key exchange protocol. The normal form, a $2 \times 2$ matrix, can be used for this purpose. We will begin by describing the seminormal form.

Let $u \in M_n$. We denote the image of $u$ under the natural epimorphism $\alpha : M_n \to M_n/[M_n, M_n]$ by $u_{ab}$. This is also called the abelianization of $u$. We note that the abelianization of $u$ is an element of a factor group of $F_n$. When no ambiguity arises we use

$u_{ab}$ to denote an element in the ambient free group $F_n$ representing $u_{ab}$. We would also like to note that we can identify $M_n/[M_n, M_n]$ with $F_n/[F_n, F_n]$.

Let $u, v \in M_n$. Suppose that $v$ acts on $u$ by conjugation. If $u \in [M_n, M_n]$, then this action can be extended to the group ring $\mathbb{Z}(M_n/[M_n, M_n])$ which we denote by $\mathbb{Z}A_n$, where $A_n = M_n/[M_n, M_n]$ is the free abelian group of rank $n$. Let $W \in \mathbb{Z}A_n$ be expressed in the form $W = \sum a_i v_i$, where $a_i \in \mathbb{Z}$, $v_i \in A_n$. Then $u^W$ denotes the product $\prod (u^{a_i})^{v_i}$. This product is well-defined since any two elements of $[M_n, M_n]$ commute in $M_n$.

Now let $u \in M_n$. Then $u$ can be written in the following seminormal form:

$$u = u_{ab} \cdot \prod_{i<j} [x_i, x_j]^{W_{ij}}$$

where $W_{ij} \in \mathbb{Z}A_n$.

In order to obtain the seminormal form of a word one uses a collection process, which utilizes the following identities:

$$[y, x] = [x, y]^{-1}$$

$$xy = yx[x, y]$$

$$xy^{-1} = y^{-1}[y,x]^{y^{-1}x^{-1}}x$$

$$x^{-1}y = y[y,x]^{y^{-1}x^{-1}}x^{-1}$$

$$[x,y]z = z[x,y]^{z}.$$

The collection process is as follows:

1. From left to right along the word $u$ begin by collecting all "non-commutator" occurrences of $x_1$ on the left by using the identities above. Repeat this with $x_2$, $x_3$, etc. In the end of this process, $u$ will be written in the form
   $u = u_{ab} \cdot c$, where $c \in [M_n, M_n]$ is a product of expressions of the form $[x_i, x_j]^g,\ g \in M_n$.

2. Since any two elements of $[M_n, M_n]$ commute in $M_n$, one can now re-group the expressions $[x_i, x_j]^g$ so that $u$ takes the form $u = u_{ab} \cdot \prod_{i<j}[x_i, x_j]^{W_{ij}}$, where $W_{ij} \in \mathbb{Z}A_n$.

The collection process is an efficient way to put a word in seminormal form. It takes quadratic time with respect to the length of $u$. The seminormal form of a word is already a word. Hence, to

convert the seminormal form to a word is trivial. Unfortunately, the seminormal form is not unique if $n > 2$.

We will now introduce a normal form which is unique and efficiently computable. Unfortunately, it is not easy to convert the normal form back to a word. We begin by defining Fox derivatives.

**Definition 3.4.3.** (see [35] or [33]) Let $\mathbb{Z}F$ be the group ring of a free group $F$ generated by $x_1, x_2, ....$ A Fox derivation with respect to $x_i$ is a map $\partial_{x_i} : \mathbb{Z}F \to \mathbb{Z}F$ such that $\partial_{x_i}(x_j) = \delta_{ij}$ and $\partial_{x_i}(vw) = \partial_{x_i}(v) + v \cdot \partial_{x_i}(w)$ for any $v, w \in F$. This map can be extended to the whole $\mathbb{Z}F$ by linearity.

**Example 3.4.1.** *[35]* Let $g \in F$ and let 1 be the identity of $F$. Since $\partial(1) = \partial(1) + \partial(1)$, it follows that $\partial(1) = 0$. Therefore $\partial(gg^{-1}) = \partial(g) + g\partial(g^{-1}) = 0$, which implies $\partial(g^{-1}) = -g^{-1}\partial(g)$.

**Example 3.4.2.** *[35]* Let $x$ and $y$ be generators of the free group $F(x, y)$. Then

$$\begin{aligned}
\partial_x([x,y]) &= \partial_x(x^{-1}y^{-1}xy) \\
&= \partial_x(x^{-1}) + x^{-1}\partial_x(y^{-1}) + x^{-1}y^{-1}\partial_x(x) + x^{-1}y^{-1}x\partial_x(y) \\
&= -x^{-1} + x^{-1}y^{-1} = x^{-1}y^{-1}(1-y)\,.
\end{aligned}$$

$$\begin{aligned}\partial_y([x,y]) &= \partial_y(x^{-1}y^{-1}xy) \\ &= \partial_y(x^{-1}) + x^{-1}\partial_y(y^{-1}) + x^{-1}y^{-1}\partial_y(x) + x^{-1}y^{-1}x\partial_y(y) \\ &= -x^{-1}y^{-1} + x^{-1}y^{-1}x = -x^{-1}y^{-1}(1-x)\,.\end{aligned}$$

Let $F_{ab} = F/F^{'}$ be the abelianization of a free group $F$. Let $\alpha : F \to F_{ab}$ be the natural epimorphism. By linearity this map can be extended to the map $\alpha : \mathbb{Z}F \to \mathbb{Z}F_{ab}$.

**Proposition 3.4.3.** *[35] Let $w \in F_n$. Then $w \in F_n''$ if and only if $\alpha(\partial_{x_i}(w)) = 0$ for each generator $x_i$ of $F_n$.*

This proposition gives an efficient algorithm to solve the word problem in a free metabelian group, $M_n$. The algorithm is as follows: Suppose you are given $w \in M_n$ as a word in relatively free generators $x_i$. One considers $w$ as an element of the free group $F_n$ with the same set of free generators, and then computes $\partial_{x_i}(w)$ for each $x_i$. One then checks whether or not all of them abelianize to 0. This is straightforward since the word problem in the free abelian group $F_{ab}$ is easily solvable.

**Proposition 3.4.4.** *[35] The algorithm for solving the word problem in $M_n$ based on Proposition 3.4.3 has at most quadratic time complexity with respect to the length of the input word.*

Now, we can give the normal form of $u \in M_n$ based on Fox derivatives.

Each $u \in M_n$ has normal form as a $2 \times 2$ matrix with the following entries:

- The entry in the lower left corner is 0.
- The entry in the lower right corner is 1.
- The entry in the upper left corner is the abelianization of $u$; that is, it is an element of the free abelian group $M_n/[M_n, M_n]$.
- The entry in the upper right corner is a vector of $n$ abelianized partial Fox derivatives of the word $u$.

Free metabelian groups have been suggested as platform groups in cryptographic protocols since the word problem in $M_n$ is efficiently solvable and $M_n$ has exponential growth, providing a large key space. The presence of a normal form is also a desired property of platform groups, as they provide a good method of diffusion and provide a solution to the word problem.

## 3.5 Small Cancellation Groups

We refer the reader to [33] and [35] for more information on small cancellation groups. Here we follow the exposition of [35].

Before defining small cancellation groups, we must begin with a few basic definitions. Let $F(X)$ be the free group on generators $X = \{x_i : i \in I\}$, where $I$ is an indexing set.

**Definition 3.5.1.** [35] A word $w(x_1, \cdots, x_n) = x_{i_1}^{\epsilon_1} \cdots x_{i_n}^{\epsilon_n}$ where $\epsilon_i = \pm 1$ for $1 \leq i \leq n$ and $x_{i_j}$ not necessarily distinct is called a reduced X-word if $x_{i_j}^{\epsilon_j} \neq x_{i_{j+1}}^{-\epsilon_{j+1}}$ for $1 \leq j \leq n-1$.

**Definition 3.5.2.** [35] The word $w(x_1, \cdots, x_n) = x_{i_1}^{\epsilon_1} \cdots x_{i_n}^{\epsilon_n}$ is cyclically reduced if it is a reduced $X$-word and $x_{i_1}^{\epsilon_1} \neq x_{i_n}^{-\epsilon_n}$.

**Definition 3.5.3.** [35] A set $R$ containing cyclically reduced words is called symmetrized if it is closed under taking cyclic permutations and inverses.

**Definition 3.5.4.** [35] Given a group $G = \langle X; R \rangle$, a non-empty word $w \in F(X)$ is called a piece if there exists two distinct relators $r_1, r_2 \in R$ such that $w$ is an initial segment of $r_1$ and $r_2$; that is, $r_1 = wv_1$ and $r_2 = wv_2$ for some $v_1, v_2 \in F(X)$ and there is no cancellation between $w$ and $v_1$ or $w$ and $v_2$.

**Definition 3.5.5.** [33] Let $R$ be a symmetrized set of relators and $\lambda > 0$. A group $G = \langle X; R \rangle$ is said to satisfy the *small cancellation condition* if for every $r \in R$ such that $r = wv$ and $w$ is a piece, one has $|w| < \lambda |r|$. $G$ is said to belong to the class $C'(\lambda)$.

Groups that satisfy the small cancellation property were proposed in [21] (see Chapter 5) because groups belonging to the class $C'(\frac{1}{6})$ have efficiently solvable word problem when using Dehn's algorithm. Dehn's algorithm is straightforward: given a word $w$, look for a piece of a relator whose length is more than half of the length of the whole relator. If no such piece exists, then $w \neq 1$ in $G$. If there is such a piece, say $u$, then $r = uv$ for some $r \in R$ where the length of $v$ is smaller than the length of $u$. Replace $u$ by $v^{-1}$ and the length of the resulting word is smaller than that of $w$. Thus, the algorithm must terminate in a finite number of steps. Dehn's algorithm has quadratic time complexity with respect to the length of $w$.

## 3.6 Matrix Groups

We refer the reader to [35] for more information on matrix groups as cryptographic platforms.

Matrix groups over finite commutative rings have been proposed as a "canonical" cryptographic platform in [35]. The fact that matrix multiplication is non-commutative while the entries of the matrix are from a commutative ring provides one with a good method of diffusion. By utilizing matrix multiplication, one can

easily hide factors in a group product.

Finite commutative rings, $R$, are proposed for the entries because they will provide periodicity; that is, for any $u \in R$ there exists $k, m \in \mathbb{Z}$ such that $u^k = u^m$. This contributes to the diffusion of elements in the group. This periodicity gives rise to a dynamical system, which typically exhibits complex behavior.

In [35] the ring $R = \mathbb{F}_p[x]/(f(x))$, where $f(x)$ is an irreducible polynomial of degree $n$, was proposed as a ground ring for a matrix group to be used in a cryptographic protocol. While this ring is isomorphic to $\mathbb{F}_{p^n}$ its structure as a quotient provides a large key space while parameters in a cryptographic protocol remain small.

# Chapter 4

# A New Key Exchange Protocol

## 4.1 Introduction

As mentioned earlier the public key exchange protocols currently in use are based on the structure of abelian groups. With improved computing power, the security of these protocols is in question. Many new public key exchanges based on group theoretic problems have been proposed (see, for example, [1], [29], [28], [43]). Here, we propose a new public key exchange problem based on the search endomorphism problem and the semidirect product. We will present the new key exchange, discuss a known attack on this key exchange when its platform group is a semidirect product of an additive abelian $p$ group and a $p$ group, and then propose a new platform group that will thwart this attack. For implementation purposes we propose parameters and methods for key generation in

this new platform. We will also introduce two simplified protocols based on the semidirect product key exchange.

## 4.2 Semidirect Product Key Exchange Protocol

We begin by recalling the definition of a semidirect product.

**Definition 4.2.1.** [25] Let $H, Q$ be two groups, and let $\phi : Q \to Aut(H)$ be a homomorphism from $Q$ into the automorphism group of $H$, $Aut(H)$. Then the semidirect product of $H$ and $Q$ is the set $\Gamma = H \rtimes_\phi Q = \{(h, q) : h \in H,\ q \in Q\}$, with the group operation given by

$$(h, q)(h', q') = (h\phi(q)(h'),\ qq').$$

For the key exchange protocol presented in [19] and [20] $n^{th}$ powers of elements in the semidirect product are needed. We begin by inductively defining the product of $n$ elements of $\Gamma$:

$$\begin{aligned}(h_1, q_1)(h_2, q_2)...(h_n, q_n) &= (h_1\phi(q_1)(h_2), q_1q_2)(h_3, q_3)...(h_n, q_n)\\ &= (h_1\phi(q_1)(h_2)\phi(q_1q_2)(h_3), q_1q_2q_3)(h_4, q_4)...(h_n, q_n)\\ &= \cdots\\ &= (h_1\phi(q_1)(h_2)\phi(q_1q_2)(h_3)...\phi(q_1q_2...q_{n-1})(h_n), q_1q_2...q_n)\end{aligned}$$

From this we can see that

$$(h, q)^n = (h\phi(q)(h)\phi(q^2)(h)\cdots\phi(q^{n-1})(h), q^n).$$

We are now in a position to present the key exchange protocol proposed in [19] and [20].

Let $A$ and $B$ be groups, $(b, a) \in B \times A$, and $n \in \mathbb{N}$. We require $B$ to be an abelian group, with the additional condition that its automorphism group, $Aut(B)$, must contain a sufficiently large abelian subgroup. In our key exchange protocol, $A, B, Aut(B), (b, a)$, and $n$ are public information.

Bob begins by choosing a homomorphism $\phi : A \to Aut(B)$, which he keeps secret. He then computes

$$x = (b, a)^n = (b\phi(a)(b)\phi(a^2)(b) \cdots \phi(a^{n-1})(b), a^n) \in B \rtimes_\phi A.$$

Bob then sends $x$ to Alice.

Alice also chooses a homomorphism $\psi : A \to Aut(B)$, which she keeps secret, and computes

$$y = (b, a)^n = (b\psi(a)(b)\psi(a^2)(b) \cdots \psi(a^{n-1})(b), a^n) \in B \rtimes_\psi A.$$

Alice sends $y$ to Bob.

Bob can find the element

$$\psi(a)(b)\psi(a^2)(b) \cdots \psi(a^{n-1})(b) \in B$$

by multiplying the first coordinate of $y$ by $b^{-1}$, while Alice can find the element

$$\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b) \in B$$

by multiplying the first coordinate of $x$ by $b^{-1}$. We will denote $\phi(a)$ by $\phi_a$ and $\psi(a)$ by $\psi_a$ for ease of notation.

Bob can now compute

$$\begin{aligned}\prod_{i=1}^{n-1} \phi_{a^i}(\psi_a(b)\psi_{a^2}(b)\cdots\psi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) \\ &= \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b).\end{aligned}$$

Similarly, Alice can compute

$$\begin{aligned}\prod_{i=1}^{n-1} \psi_{a^i}(\phi_a(b)\phi_{a^2}(b)\cdots\phi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \psi_a^i(\phi_a(b)\phi_a^2(b)\cdots\phi_a^{n-1}(b)) \\ &= \prod_{i=1}^{n-1} \psi_a^i \circ \phi_a(b)\psi_a^i \circ \phi_a^2(b)\cdots\psi_a^i \circ \phi_a^{n-1}(b)).\end{aligned}$$

If in addition to $B$ being abelian, we require that $\phi_a$ and $\psi_a$

commute in $Aut(B)$, then we have the following equality:

$$\begin{aligned}
\prod_{i=1}^{n-1} \phi_{a^i}(\psi_a(b)\psi_{a^2}(b)\cdots\psi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) \\
&= \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b) \\
&= \prod_{i=1}^{n-1} \psi_a \circ \phi_a^i(b)\psi_a^2 \circ \phi_a^i(b)\cdots\psi_a^{n-1} \circ \phi_a^i(b) \\
&= \prod_{j=1}^{n-1} \psi_a^j \circ \phi_a(b)\psi_a^j \circ \phi_a^2(b)\cdots\psi_a^j \circ \phi_a^{n-1}(b) \\
&= \prod_{j=1}^{n-1} \psi_a^j(\phi_a(b)\phi_a^2(b)\cdots\phi_a^{n-1}(b)) \\
&= \prod_{j=1}^{n-1} \psi_{a^j}(\phi_a(b)\phi_{a^2}(b)\cdots\phi_{a^{n-1}}(b))
\end{aligned}$$

If we denote this element by $k$, then $k$ is computable by both Bob and Alice as shown above, giving them a shared key. The security of this protocol relies on the difficulty of recovering the homomorphisms $\phi$ and $\psi$ from $n$, $(b, a) \in B \times A$, $x$, and $y$.

The most obvious way for an adversary to find the shared secret key is to determine either the homomorphism $\phi$ or $\psi$ based on the public information. Since both $x$ and $y$ are public, an adversary can easily compute

$$\psi(a)(b)\psi(a^2)(b)\cdots\psi(a^{n-1})(b) \in B$$

and

$$\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b) \in B.$$

If the adversary determines $\phi$, then he knows $\phi_a \in Aut(B)$ and can compute

$$\prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) = \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b))$$

which is the shared key, $k$. Similarly, if the adversary determines $\psi$, he can also find the shared secret key.

## 4.3 Proposed Platform

In [19] it was proposed that the group $B$ should be an additive abelian $p$ group of order $p^m$ with the additional property that $pb = 0$ for every $b \in B$. This group was chosen so that $B$ can be considered an $m$ dimensional vector space over $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$. Under these conditions the group of automorphisms of $B$ is $GL_m(\mathbb{F}_p)$. The order of $GL_m(\mathbb{F}_p)$ is $(p^m-1)(p^m-p)\cdots(p^m-p^{m-1})$. Since we require a homomorphism $\phi : A \to Aut(B)$, the order of $\phi(a)$ must divide the order of $a$. We know that $Aut(B)$ has an element of order $p$, and so to ensure that we can find a non-trivial homomorphism $\phi$, we require that the group $A$ also be a $p$ group of order $p^l$. Now $|B \rtimes A| = p^{m+l}$, and so $1 < n < p^{m+l}$ [11].

Recall that in order for an adversary to find the key, one must be able to determine either the homomorphism $\phi$ or $\psi$ based on the public information. Since both $x$ and $y$ are public, an adversary can easily compute

$$\psi(a)(b)\psi(a^2)(b)\cdots\psi(a^{n-1})(b) \in B$$

and

$$\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b) \in B.$$

If the adversary determines $\phi$ then he knows $\phi_a$ and can compute

$$\prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) = \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b))$$

which is the shared key, $k$. Similarly if the adversary determines $\psi$, he can also find the secret key.

If we require $B$ and $A$ to be as above, the homomorphisms $\phi_a$ and $\psi_a$ have matrix representations $J$ and $K$ respectively, and the element $b$ and shared key $k$ can be represented as column vectors.

We would like to ensure that it is easy to find homomorphisms $\phi_a, \psi_a$ that commute. We begin by choosing an arbitrary matrix $M \in GL_m(\mathbb{F}_p)$ and a natural number $s$ with $s = \left\lfloor \frac{m}{2} \right\rfloor$ which are made public. Bob chooses an automorphism of the form $H = \begin{pmatrix} Q & 0 \\ 0 & I \end{pmatrix}$ and Alice chooses an automorphism of the form $S =$

$\begin{pmatrix} I & 0 \\ 0 & R \end{pmatrix}$, where $Q, R$ are $s \times s$ block matrices and $I$ is the $m - s$ identity matrix. The matrices $H$ and $S$ are kept secret. Bob and Alice form the matrices $J = P_B(MHM^{-1}) = \sum_{i=1}^{d} c_i MH^i M^{-1}$ and $K = P_A(MSM^{-1}) = \sum_{i=1}^{m} c_i MS^i M^{-1}$ respectively, where $c_i \in \mathbb{F}_p$. The matrices $J$ and $K$ commute, and can be used as $\phi_a$ and $\psi_a$. Although these matrices are not necessarily invertible, it will not affect the computations above. Now computing the shared key in this platform takes the form

$$\begin{aligned} &\sum_{i=1}^{n-1} J^i(K \cdot b + K^2 \cdot b + \cdots + K^{n-1} \cdot b) \\ &= (J + J^2 + \cdots + J^{n-1})(K \cdot b + K^2 \cdot b + \cdots + K^{n-1} \cdot b) \\ &= (K + K^2 + \cdots + K^{n-1})(J \cdot b + J^2 \cdot b + \cdots + J^{n-1} \cdot b) \\ &= k \end{aligned}$$

Recall that Bob can find the element

$$\psi(a)(b)\psi(a^2)(b) \cdots \psi(a^{n-1})(b) \in B$$

by multiplying the first coordinate of $y$ by $b^{-1}$, while Alice can find the element

$$\phi(a)(b)\phi(a^2)(b) \cdots \phi(a^{n-1})(b) \in B.$$

In this platform, this takes the following form. Bob can recover

the element

$$K \cdot b + K^2 \cdot b + \cdots + K^{n-1} \cdot b$$

by subtracting $b$ from the first coordinate of $y$, while Alice can recover

$$J \cdot b + J^2 \cdot b + \cdots + J^{n-1} \cdot b$$

by subtracting $b$ from the first coordinate of $x$. We will denote these elements by $g$ and $h$ respectively.

Finding the matrix $K$ from the public information is equivalent to solving the matrix equation

$$(K + K^2 + \cdots + K^{n-1}) \cdot b = g. \tag{4.3.1}$$

Similarly if one wanted to find the matrix $J$, he or she must solve the equation

$$(J + J^2 + \cdots + J^{n-1}) \cdot b = h. \tag{4.3.2}$$

We may write either of these matrix equations as a system of $m$ equations. Since we would like to know the entries of $K$(or $J$) $\in GL_m(\mathbb{F}_p)$ and our matrix equation involves powers of $K$(or $J$), these equations are non-linear with integer solutions. Hence, the most obvious way to determine the matrix $K$(or $J$) is to solve $m$

distinct non-linear equations in $m$ variables over a finite field. The search problem for polynomial equations over finite fields is known to be NP-hard [3]. We would like to note that equations (4.3.1) and (4.3.2) can be simplified by adding $b$ to both sides. Equations (4.3.1) and (4.3.2) become

$$(I + K + K^2 + \cdots + K^{n-1}) \cdot b = b + g$$

and

$$(I + J + J^2 + \cdots + J^{n-1}) \cdot b = b + h,$$

which are equivalent to

$$(I - K^n) \cdot b = (I - K) \cdot (b + g)$$

and

$$(I - J^n) \cdot b = (I - J) \cdot (b + h),$$

respectively. Although this simplifies the equation, it does not simplify the problem at hand.

To illustrate how the scheme works in this setting, we give a simple example.

**Example 4.3.1.** *Let* $H = \mathbb{Z}_p \times \mathbb{Z}_p = \langle h_1 \rangle \times \langle h_2 \rangle$ *and* $K = \mathbb{Z}_p = \langle x \rangle$. $Aut(H) \cong GL_2(\mathbb{F}_p)$. *Take n=3. Let* $\phi : x \mapsto \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ *and let*

$\psi : x \mapsto \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$. *Each element in* $H$ *can be represented as* $\begin{pmatrix} a \\ b \end{pmatrix}$.

*Bob computes* $\left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right)^3 \in H \rtimes_\phi K$.

$$\left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right) \left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right) = \left( \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}, x \right)$$
$$= \left( \begin{pmatrix} 2a \\ a+2b \end{pmatrix}, x^2 \right)$$

$$\left( \begin{pmatrix} 2a \\ a+2b \end{pmatrix}, x^2 \right) \left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right) = \left( \begin{pmatrix} 2a \\ a+2b \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}, x^3 \right)$$
$$= \left( \begin{pmatrix} 3a \\ 3a+3b \end{pmatrix}, x^3 \right).$$

*Bob sends* $\left( \begin{pmatrix} 3a \\ 3a+3b \end{pmatrix}, x^3 \right)$ *to Alice.*

*Alice computes* $\left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right)^3 \in H \rtimes_\psi K$.

$$\left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right) \left( \begin{pmatrix} a \\ b \end{pmatrix}, x \right) = \left( \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}, x \right)$$
$$= \left( \begin{pmatrix} 2a \\ 2a+2b \end{pmatrix}, x^2 \right).$$

$$\left(\begin{pmatrix} 2a \\ 2a+2b \end{pmatrix}, x^2\right)\left(\begin{pmatrix} a \\ b \end{pmatrix}, x\right) = \left(\begin{pmatrix} 2a \\ 2a+2b \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix}, x^3\right)$$
$$= \left(\begin{pmatrix} 3a \\ 6a+3b \end{pmatrix}, x^3\right).$$

*Alice sends* $\left(\begin{pmatrix} 3a \\ 6a+3b \end{pmatrix}, x^3\right)$ *to Bob.*

*Bob computes the secret as follows. He begins by first computing*
$$\begin{pmatrix} 3a \\ 6a+3b \end{pmatrix} - \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 2a \\ 6a+2b \end{pmatrix}.$$
*He then computes*
$$\left(\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 2a \\ 6a+2b \end{pmatrix}\right) + \left(\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 2a \\ 6a+2b \end{pmatrix}\right) = \begin{pmatrix} 4a \\ 18a+4b \end{pmatrix}.$$

*Alice computes the secret as follows. She begins by first computing*
$$\begin{pmatrix} 3a \\ 3a+3b \end{pmatrix} - \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 2a \\ 3a+2b \end{pmatrix}.$$
*She then computes*
$$\left(\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 2a \\ 3a+2b \end{pmatrix}\right) + \left(\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 2a \\ 3a+2b \end{pmatrix}\right) = \begin{pmatrix} 4a \\ 18a+4b \end{pmatrix}.$$

*So* $\begin{pmatrix} 4a \\ 18a+4b \end{pmatrix}$ *is the shared key.*

## 4.4 Cryptanalysis

Although the underlying problem of the protocol when implemented in the proposed platform in Section 4.3 is NP-hard, Blackburn, Cid, and Mullan provided a cryptanalysis that proved the scheme is not secure under this platform in [5]. The cryptanalysis in [5] is as follows:

Suppose the adversary Eve recovers $g$ and $h$ from the public information. Let $X$ be any matrix that commutes with $L+L^2+\cdots+L^{n-1}$ for any matrix $L$ that Bob can produce with the additional property that $X \cdot b = g$. We know a matrix satisfying these conditions exists since $X = K+K^2+\cdots+K^{n-1}$ satisfies these conditions. These conditions on $X$ are linear conditions as we have the equation $X \cdot b = g$. The condition of commuting with $L+L^2+\cdots+L^{n-1}$ is expressed as $X \cdot (L+L^2+\cdots+L^{n-1}) = (L+L^2+\cdots+L^{n-1}) \cdot X$. In order to find a matrix $X$ that satisfies this commuting condition, the adversary Eve can randomly generate suitable matrices $L + L^2 + \cdots + L^{n-1}$ and then impose the necessary condition $X \cdot (L + L^2 + \cdots + L^{n-1}) = (L + L^2 + \cdots + L^{n-1}) \cdot X$ on $X$. The number of matrices produced by Eve required for the necessary condition to become sufficient to imply the commuting condition is very small since the conditions are linear. The matrix $X$ can be

computed efficiently since all the conditions are linear.

Upon finding $X$, the secret key is computed by $k = X \cdot h$. To see this, note that:

$$\begin{aligned} X \cdot h =& X \cdot (J + J^2 \cdots + J^{n-1}) \cdot b \\ =& (J + J^2 \cdots + J^{n-1}) \cdot X \cdot b \\ =& (J + J^2 \cdots + J^{n-1}) \cdot g \\ =& k. \end{aligned}$$

Hence the adversary can recover the shared key.

### 4.4.1 New Platform Group

In order to thwart this attack the holomorph of a free metabelian group, $M_k \rtimes Aut(M_k)$, was proposed as an alternative platform in [20]. Since $M_k$ is not abelian, the protocol should be slightly modified when choosing a public element $(b, a) \in M_k \times Aut(M_k)$. Since the group $M_k$ is metabelian the element $b \in M_k$ chosen should be an element of the commutator subgroup $M_k' = [M_k, M_k]$. This will give the commutativity required for the protocol. The holomorph of a free metabelian group was also suggested since Roman'kov [39] showed that the endomorphism problem in $M_n$ is NP-hard. As mentioned in Chapter 3 free metabelian groups were

suggested since the word problem in $M_n$ is efficiently solvable. In addition, $M_n$ exhibits exponential growth.

## 4.5 A Simplified Protocol

Upon investigating the protocol presented in Section 4.2 one can see that the semidirect product structure is not required. After making this observation, we can simplify the protocol as follows. For the full paper, see [20].

Let $B$ be a group, with the additional condition that its automorphism group contains a sufficiently large abelian subgroup. An element $b \in B$ is chosen and made public as well as an arbitrary $n \in \mathbb{N}$. Bob begins by choosing an automorphism $\phi : B \to B$, which he keeps secret. He then computes

$$x = \phi(b)\phi^2(b) \cdots \phi^n(b) \in B.$$

Bob then sends $x$ to Alice.

Alice also chooses an automorphism $\psi : B \to B$, which she keeps secret, and computes

$$y = \psi(b)\psi^2(b) \cdots \psi^n(b) \in B.$$

Alice sends $y$ to Bob.

Bob can now compute

$$\begin{aligned}\prod_{i=1}^{n}\phi^i(y) &= \prod_{i=1}^{n}\phi^i(\psi(b)\psi^2(b)\cdots\psi^n(b)) \\ &= \prod_{i=1}^{n}\phi^i\circ\psi(b)\phi^i\circ\psi^2(b)\cdots\phi^i\circ\psi^n(b).\end{aligned}$$

Similarly, Alice can compute

$$\begin{aligned}\prod_{i=1}^{n}\psi^i(x) &= \prod_{i=1}^{n}\psi^i(\phi(b)\phi^2(b)\cdots\phi^n(b)) \\ &= \prod_{i=1}^{n}\psi^i\circ\phi(b)\psi^i\circ\phi^2(b)\cdots\psi^i\circ\phi^n(b).\end{aligned}$$

If we require that $\phi$ and $\psi$ commute in $Aut(B)$ and $\phi^i(b)$ and $\psi^i(b)$ commute (for $i \in \{1,\cdots n\}$) then we have the following equality:

$$\begin{aligned}\prod_{i=1}^{n}\phi^i(\psi(b)\psi^2(b)\cdots\psi^n(b)) &= \prod_{i=1}^{n}\phi^i\circ\psi(b)\phi^i\circ\psi^2(b)\cdots\phi^i\circ\psi^n(b)\\ &= \prod_{i=1}^{n}\psi\circ\phi^i(b)\psi^2\circ\phi^i(b)\cdots\psi^n\circ\phi^i(b)\\ &= \prod_{j=1}^{n}\psi^j\circ\phi(b)\psi^j\circ\phi^2(b)\cdots\psi^j\circ\phi^n(b)\\ &= \prod_{i=1}^{n}\psi^i(\phi(b)\phi^2(b)\cdots\phi^n(b))\end{aligned}$$

If we denote this element by $k$, then $k$ is computable by both Bob and Alice as shown above, giving them a shared key. The most obvious way for an adversary to find the shared secret key is to determine either the endomorphism $\phi$ or $\psi$ based on the public information $b \in B$, $x$ and $y$. If an adversary can recover either endomorphism $\phi$ or $\psi$ based on the public information, he or she will recover the shared key. Hence, the security of this protocol is based on the difficulty of the endomorphism search problem in the proposed platform group, the free metabelian group of rank $r$, $M_r$. This problem is known to be NP-hard due to a result by Roman'kov (see [39]). Although the security of this protocol is based on an NP-hard problem, we would like to note that it is not known whether or not there are other attacks on this protocol that do not require recovering either of the endomorphisms $\phi$ or $\psi$.

## 4.6 Parameters and Key Generation

The parameters and method for key generations suggested in [20] are as follows. For the simplified protocol in the previous section, the suggested platform is the free metabelian group $M_r$ of a fairly small rank. We suggest that the rank is $r = 6$ for purposes of efficiency. Suppose the group $M_r$ is generated by $x_1, \ldots, x_r$.

The element $b \in M_r'$ is selected in the form $b = \prod_{i<j}[x_i, x_j]^{\pm u_{ij}}$, where $u_{ij}$ are words in the generators $x_1, \ldots, x_r$, such that $1 \leq |u_{ij}| \leq 3$. Thus, if $r = 6$, then the length of $b$ is at most 120.

Bob and Alice to need a way to efficiently obtain commuting endomorphisms $\phi$ and $\psi$, respectively. We suggest the following.

Suppose that we have $x_1, \ldots, x_{2k}$, generators of $M_{2k}$, $k \geq 2$. Bob may choose

$$\sigma : x_i \mapsto y_i = y_i(x_1, \ldots, x_k, 1, \ldots, 1) \text{ for } 1 \leq i \leq k,$$
$$\sigma : x_i \mapsto x_i \text{ for } k+1 \leq i \leq 2k,$$

while Alice may choose

$$\tau : x_i \mapsto x_i \text{ for } 1 \leq i \leq k,$$
$$\tau : x_i \mapsto z_i = z_i(1, \ldots, 1, x_{k+1}, \ldots, x_{2k}) \text{ for } k+1 \leq i \leq 2k.$$

These endomorphisms commute, and so will their conjugates $\alpha^{-1}\tau\alpha$ and $\alpha^{-1}\sigma\alpha$, for any automorphism $\alpha$ of $M_{2k}$.

Bob (Alice) also need a method to construct the words $y_i$ ($z_i$) for each $i$. They can do this by first selecting the length of $y_i$ ($z_i$) uniformly randomly from the range $[5, 30]$. They then build a word of selected length letter by letter. Each time he/she chooses a letter from the set of relevant generators and their inverses, ensuring that there are not any consecutive $x_j$ and $x_j^{-1}$.

Now Alice selects her endomorphism $\psi$ in the form $\alpha^{-1}\tau\alpha$ for some $\alpha \in Aut(M_{2k})$, and Bob selects his $\phi$ in the form $\alpha^{-1}\sigma\alpha$ for the same automorphism $\alpha$. This automorphism $\alpha$ should be public. $\alpha$ is chosen as a product of $N$ Nielsen automorphisms, where $N$ is selected uniformly randomly from the range $[10, 20]$.

We also need to specify the parameter $n$, which is the maximum number of iterations of $\psi$ or $\phi$ to be computed by Alice/Bob. By choosing $n$ to be very small (say $n = 2$) the length of relevant endomorphisms will be prevented from being too big. Since the length of $\phi^n$, in general, grows exponentially in $n$ it is important to keep this parameter small.

## 4.7 Another Simplified Protocol

The following protocol was also suggested in [20].

This key exchange uses a free metabelian group as the platform,

and endomorphisms of this group as the private keys. We would like to note that a similar idea was explored in [34], where automorphisms of finite $p$-groups were used. Using automorphisms instead of arbitrary endomorphisms substantially decreases the size of key space. Finite $p$-groups were most likey chosen due to the fact that the group of automorphisms of such a group is abelian. In our case arbitrary endomorphisms of a free metabelian group do not necessarily commute. Hence, extra care in choosing the endomorphisms must be taken when using a free metabelian group as a platform. However, since a free metabelian group is a relatively free group any map on the (standard) generators extends to an endomorphism of this group. Hence, we have a large key space even if we make the restriction that Alice and Bob choose their endomorphisms to commute.

The platform group we suggest is the free metabelian group $M_{2k}$, with $k \geq 2$, which is made public. In addition, there is an element $g \in M_{2k}$ which is made public. The protocol is as follows:

1. Alice chooses an endomorphism $\phi$ of $M_{2k}$ as her secret key and sends $\phi(g)$ to Bob.

2. Bob chooses another endomorphism $\psi$ as his secret key and sends $\psi(g)$ to Alice.

3. Assuming that $\phi$ and $\psi$ commute, the shared public key is

$$k_A = \phi(\psi(g)) = \psi(\phi(g)) = k_B.$$

We suggest the same parameters and key generation for this protocol as in the previous section.

The security is based on the fact that the endomorphism search problem in a free metabelian group is hard; that is, it is computationally hard to recover an endomorphism $\phi$ from $g$ and $\phi(g)$. This assumption is supported by a result of Roman'kov [39] that states that the endomorphism search problem in a free metabelian group is at least as hard as the Diophantine search problem, which is NP-hard (see [14]). We would like to note that although the security assumption is based on an NP-hard problem generically this problem could be easy; that is, there could be an algorithm that runs in polynomial time on most inputs. For example, the subset sum problem and the 3-satisfiability problem are NP-complete problems with low generic case complexity (see [35]). In addition, there could be other attacks on the protocol that do not require recovering the endomorphisms $\phi$ or $\psi$. We would like to further investigate potential attacks on this protocol that do not require recovering the endomorphisms $\phi$ or $\psi$.

# Chapter 5

# Secret Sharing Schemes

## 5.1 Introduction

Secret sharing schemes are a method of distributing a secret among a number of participants in such a way that a certain number of participants is required to recover the secret. More formally, the definition is as follows.

**Definition 5.1.1.** [44] Let $t, n$ be positive integers with $t \leq n$. A $(t, n)$-threshold scheme is a method of distributing a secret $K$ among $n$ participants in such a way that any of the $t$ participants can recover the secret, but any group of $t-1$ (or less) participants cannot.

Shamir created a $(t, n)$-threshold scheme in 1979 that utilizes polynomial interpolation. In Shamir's scheme the secret is an element $K \in \mathbb{Z}_p$, where $p$ is a prime larger than $n$. In order to

distribute the secret, the dealer begins by choosing a polynomial $f$ of degree $t-1$ such that $f(0) = K$. Then he sends the value $y_i = f(x_i)$, where $x_i \in \mathbb{Z}_p$, secretly to participant $P_i$. The scheme is presented in Figure 5.1.

Figure 5.1: Shamir $(t, n)$-Threshold Scheme, [44]

**Initialization Phase**

(1) The dealer chooses $n$ distinct non-zero elements $x_i \in \mathbb{Z}_p$, $1 \leq i \leq n$. The dealer then distributes (publicly) the values $x_i$ to participant $P_i$ for $1 \leq i \leq n$.

**Share Distribution**

(2) Suppose the dealer wants to distribute a secret $K \in \mathbb{Z}_p$. The dealer randomly chooses elements $a_i \in \mathbb{Z}_p$, $1 \leq i \leq t-1$, which he keeps secret.
(3) The dealer then computes $y_i = f(x_i)$, for $1 \leq i \leq n$, where $f(x)$ is the polynomial

$$f(x) = K + \sum_{j=1}^{t-1} a_j x^j \mod p.$$

(4) The dealer then secretly distributes the share $y_i$ to participant $P_i$.

In order for the participants to recover the secret, they use polynomial interpolation to recover $f$ and hence the secret $f(0)$. This can be done as follows. Suppose participants $P_{i_1}, \cdots, P_{i_t}$ want to recover the secret $K$. When the participants pool their shares, they will have $t$ points $(x_{i_j}, y_{i_j})$, $1 \leq j \leq t$ on the polynomial. The polynomial $f$ must be of the form $f(x) = a_0 + a_1 x + \cdots a_{t-1} x^{t-1}$ since its degree is $t-1$. By recovering the coefficients of the polynomial, the participants will recover the secret $a_0 = K$. Since the

participants know the values $y_{i_j} = f(x_{i_j})$, they will obtain $t$ linear equations in the unknowns $a_0, \cdots, a_{t-1}$ over $\mathbb{Z}_p$. This system of equations will always have a unique solution. One can see this by observing the following theorem.

**Theorem 5.1.1** (Lagrange interpolation formula)**.** *[44] Suppose $p$ is a prime, $x_1, \cdots, x_t$ are distinct elements in $\mathbb{Z}_p$, and $a_1, \cdots, a_t$ are (not necessarily distinct) elements in $\mathbb{Z}_p$. Then there is a unique polynomial $A(x) \in \mathbb{Z}_p[x]$ having degree at most $t-1$, such that $A(x_i) = a_i$ for $1 \leq i \leq t$. The polynomial $A(x)$ is as follows:*

$$A(x) = \sum_{j=1}^{t} a_j \prod_{1 \leq h \leq t, h \neq j} \frac{x - x_h}{x_j - x_h}$$

Hence, in order to determine the secret the participants must solve a system of linear equations over $\mathbb{Z}_p$ or use the Lagrange interpolation formula. If the participants use the Lagrange interpolation formula to recover the secret, they can determine only the constant term rather than the entire polynomial by entering $x = 0$ in the formula. In this situation, no $t-1$ (or less) participants can gain any information about the secret while any $t$ of them can.

In the remainder of this chapter, we will propose two new secret sharing schemes based on group presentations and the word problem. Before proposing these new schemes, we discuss the notion of proactive secret sharing. This notion allows us to provide extra

security for the proposed secret sharing schemes. We also review a secret sharing scheme based on group presentations and the word problem that was proposed in [38].

## 5.2 Proactive Secret Sharing

Here we follow the exposition of [23] and [26].

Suppose an adversary would like to gain information about a secret that has been distributed using a secret sharing scheme. Since secret sharing schemes distribute the secret over various locations, the adversary must attack several locations in order to gain information about the secret. Hence, if a $(t, n)$-threshold scheme was used to distribute the secret an adversary must attack at least $t$ locations in order to obtain any information about the secret. In addition, if an adversary would like to destroy the secret he must corrupt at least $n - t + 1$ shares in order to do so. Although the adversary must attack several locations to gain information about the secret or destroy the secret, the adversary has the entire lifetime of the secret to mount his attack. Hence for long term secrets, the protection provided by simply distributing the secret may be insufficient.

While this problem can be solved by refreshing the secrets, this is not always possible (consider, for example, if the secret is a will). The solution to this problem is given by proactive secret sharing. The basic idea of proactive secret sharing schemes is that the individual shares are periodically modified in such a way that leaves the secret unchanged, and renders the old shares useless. Proactive secret sharing must also protect the corruption of shares by periodically recovering corrupted shares without leaking any information about the recovered shares. Secret sharing schemes involve two processes: a method of distributing shares of a secret and a method of recovering the secret. The proactive approach deals with the time in between these two processes.

When using the proactive approach the lifetime of the secret is divided into time periods. Each time period begins with an update phase in which the participants receive new shares of the same secret. The old shares become useless and are safely erased. Using this approach limits the amount of time an adversary has to attack a minimum of $t$ shares that are required to obtain information about the secret. Instead of having the entire lifetime of the secret to mount attacks, the adversary must attack multiple

locations within a certain time period. In addition, if the adversary wishes to destroy the secret he must corrupt at least $n-t+1$ shares within a specific time period rather than having the entire lifetime of the secret. This enhances the security of the traditional secret sharing scheme.

The definition of security of a proactive secret sharing scheme given here is relative to the mobile adversary model. The mobile adversary model assumes that the adversary can corrupt any of the servers (participants) at any moment in time. The mobile adversary cannot control a server forever, but can break into each server multiple times. In addition, it is assumed that the adversary can corrupt no more than $t-1$ servers in any given time period. In order to define security of a proactive secret sharing scheme we must begin with the following. Let $\kappa$ be a function on the space of secrets $x$. Let $p_0^{(\kappa)}$ be the probability that an adversary correctly computes $\kappa(x)$ with only the knowledge of the initial public information about the secret, and let $p_1(x)^{(\kappa)}$ be the probability that the adversary correctly computes $\kappa(x)$ after the protocol has run.

**Definition 5.2.1.** [23] A proactive secret sharing scheme is semantically secure if for any function $\kappa$ computable on the secret,

the difference between the probabilities $p_0^{(\kappa)}$ and $p_1^{(\kappa)}$ is negligible.

**Definition 5.2.2.** [23] A proactive secret sharing scheme that guarantees the correct reconstructibility of the secret at any time is called robust.

If a scheme is robust, one must ensure that at any time the honest participants have correct shares and that the other participants can verify the correctness of the shares.

In [23] and [26] a semantically secure and robust proactive secret sharing scheme based on Shamir's secret sharing scheme is presented. The scheme presented in [23] and [26] utilizes the notion of verifiable secret sharing. The shares in Shamir's scheme can easily be corrupted by a dishonest dealer. In order to prevent this, verifiable secret sharing, a protocol to verify consistent dealing, is used. For the purposes of this paper, we will only discuss the share renewal protocol proposed in [23] and [26]. This is the basic component of the proactive secret sharing scheme presented in [23] and [26]. The security of this share renewal protocol is relative to a passive adversary; that is, the adversary can learn information from a corrupted server, but all participants still follow the predetermined protocol.

The scheme begins with an initialization phase where a secret $x \in \mathbb{Z}_p$ for a prime $p$ is distributed among the $n$ participants using Shamir's secret sharing protocol. Rather than the dealer sending $y_i = f(x_i)$ for some randomly chosen $x_i \in \mathbb{Z}_p$, the dealer will simply send participant $P_i$ the value $f(i)$. Following the initialization phase is an update phase in which the participants perform a share renewal protocol. In order to renew the shares, the polynomial $f$ is modified by adding to it a polynomial $g$ of degree $t-1$ with the property that $g(0) = 0$. Then we have $x = f(0) = f(0) + g(0)$. The new shares are then $x_i = f(i) + g(i)$. Thus, the shares are modified while the secret remains unchanged and the old shares are now useless. The shares computed in time period $l$ are denoted $x_i^{(l)}$ and the polynomial corresponding to these shares is denoted $f^{(l)}$.

The share renewal protocol (in the presence of a passive attacker) for each participant $P_i$ at time $l$ is as follows:

1. $P_i$ picks $t-1$ random numbers $\{g_{im}\}_{m \in \{1,\cdots,t-1\}} \in \mathbb{Z}_p$ which define a polynomial $g_i(z) = g_{i1}z + g_{i2}z^2 + \cdots + g_{it-1}z^{t-1}$ with $g_i(0) = 0$.

2. $P_i$ secretly sends $u_{ij} = g_i(j) \mod p$ to $P_j$ for $j \neq i$.

3. After receiving $u_{ij}$ for $j \in \{1, \cdots, n\}$, $P_i$ computes his new share $x_i^{(l)} = x_i^{(l-1)} + u_{1i} + u_{2i} + \cdots + u_{ni}$. $P_i$ then erases all variables used to obtain his new share.

**Theorem 5.2.1.** *[23] If all participants follow the above share renewal protocol then:*

- *Robustness: The new shares computed at the end of the update phase correspond to the secret $x$; that is, any subset of $t$ participants can reconstruct the secret $x$.*
- *Secrecy: An adversary that at any time period knows no more than $t-1$ shares learns nothing about the secret.*

## 5.3 Secret Sharing Scheme using Group Presentations

Recently the field of non-commutative group-based cryptography has flourished, resulting in many new non-commutative cryptographic protocols. For the interested reader we refer [35] and [13]. In particular, D. Panagopoulos created a $(t, n)$-threshold scheme using group presentations and the word problem [38]. He proposed polycylic groups and Coxeter groups as potential platform groups. The group theoretic scheme created by Panagopoulos has

one main advantage over Shamir's scheme: the secret need not be determined before each individual receives his or her share of the secret.

The scheme presented in [38] is as follows. Suppose that one would like to distribute a binary sequence $a_1 \cdots a_l$ among $n$ persons in such a way that at least $t$ participants are required to recover the sequence.

1. A group $G = \langle x_1, \cdots, x_k | r_1, \cdots, r_m \rangle$ with solvable word problem is chosen, and $m = \binom{n}{t-1}$. The set of generators $x_1, \cdots, x_k$ is public.

2. Let $A_1, \cdots, A_m$ be an enumeration of the subsets of $\{1, \cdots, n\}$ with $t-1$ elements. Let $R_1, \cdots, R_n$ be subsets of $\{r_1, \cdots, r_m\}$ with $r_j \in R_i$ if and only if $i \in A_j$, for $j = 1, \cdots, m$ and $i = 1, \cdots, n$. Note that for each $j \in \{1, \cdots, m\}$ $r_j$ is not contained in exactly $t-1$ of the subsets $R_1, \cdots, R_n$. Therefore, $r_j$ is contained in any union of $t$ of the subsets $R_1, \cdots, R_n$ and if we take any union of $t-1$ subsets of $R_1, \cdots, R_n$ there exists a $j$ such that $r_j$ is not in their union.

3. The dealer then secretly distributes to each participant one of the sets $R_1, \cdots, R_n$.

4. The dealer constructs and distributes publicly a sequence of elements of $G$ such that $w_i =_G 1$ if and only if $a_i = 1$ for $i = 1, \cdots, l$. The word $w_i$ must involve most of the relations $r_1, \cdots, r_m$ if $w_i = 1$. In addition, all of the relations must be used in the construction of at least one element.

Any $t$ of the participants can obtain the binary sequence $a_1 \cdots a_l$ by taking the union of the subsets $R_i$ that they obtained from the dealer. By doing this, they will obtain the group presentation $G = \langle x_1, \cdots, x_k | r_1, \cdots, r_m \rangle$ and can therefore solve the word problem for each $w_i$, $i = 1, \cdots, l$, distributed. Fewer than $t$ participants cannot correctly recover the secret since any union with fewer than $t$ of the subsets $R_1, \cdots, R_n$ will be missing some of the relations $r_1, \cdots, r_m$.

One method of attack suggested by [38] is to search the pool of group presentations used in Step 1 and then try to decode $w_1, \cdots, w_l$. In order to thwart this type of attack, the author suggests that the pool of group presentations be large. The author also notes that if the adversary has obtained some of the sets $R_1, \cdots, R_n$ this method of attack becomes more efficient. This is why in Step 4 it is required that most of the relations must be used in a word $w = 1$. For more on possible attacks, methods of

constructing a word equal to 1, and proposed platform groups of this protocol see [38].

## 5.4 New Secret Sharing Schemes using the Word Problem

Here we present two secret sharing schemes that are proposed in [21]. The first scheme presented is an $(n, n)$-threshold scheme in which all participants are needed to recover the secret. This scheme utilizes group presentations and the word problem. Just as in [38], it has the advantage over Shamir's scheme that the secret does not need to be determined before each individual receives his or her share. We also propose to use the notion of a secure sum in this scheme to give the additional advantage that no information about the individual shares needs to be revealed in order to recover the secret. The second scheme proposed is a hybrid scheme which utilizes Shamir's scheme and the idea of the $(n, n)$-threshold scheme we proposed. The scheme has the same secret as Shamir's scheme, but rather than sending $f(i) = x_i$ over secure channels the integers are sent in disguise. The participants then use group theoretic methods to recover the integers, and polynomial interpolation to recover the secret. This scheme has two advantages

over Shamir's scheme. The first is that the secret does not need to be determined before the individual shares are distributed. The second advantage is that the integers $f(i) = x_i$ are sent over open channels.

### 5.4.1 A $(n, n)$-threshold Scheme

Suppose we would like to distribute a $k$-column vector, $C = \begin{pmatrix} c_1 \\ c_2 \\ \cdot \\ \cdot \\ c_k \end{pmatrix}$ consisting of 0's and 1's among $n$ people in such a way that the vector can be retrieved only when all $n$ participants cooperate. We begin by making a set of generators $X = \{x_1, \cdots, x_m\}$ public. The scheme is as follows:

1. The dealer distributes over a secure channel to each participant $P_j$ a set of relators $R_j$ such that each group $G_j = \langle x_1, \cdots, x_m | R_j \rangle$ has efficiently solvable word problem.

2. The dealer then distributes words $w_{1j}, \cdots, w_{kj}$ in the generators $x_1, \cdots, x_m$ over an open channel to participant $P_j$ for $1 \leq j \leq n$. The words are chosen so that the number of words $w_{ij}$ such that $w_{ij} =_{G_j} 1$ is even if $c_i = 0$ and odd if $c_i = 1$.

3. Participant $P_j$ then checks if each word $w_i =_{G_j} 1$ or not. From this each participant can make a column of 0's and 1's,

$C_j = \begin{pmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{kj} \end{pmatrix}$, by setting $c_{ij} = 1$ if $w_i =_{G_j} 1$ and 0 otherwise.

4. The participants then construct the secret by forming the column vector $C = \sum_{j=1}^{n} C_j$, where the sum of the entries is taken modulo 2.

By the dealer's choice of words, it is clear that the participants will construct the secret correctly if we assume that the shares have not been corrupted. Hence, this scheme is robust.

We propose that the participants use the protocol of computing a secure sum as suggested in [17]. This will give our scheme the additional advantage over Shamir's scheme that the participants need not reveal any information about their individual shares to recover the secret. The protocol for computing a secure sum is as follows:

1. $P_1$ begins the process by choosing a random column vector $N_1$. He then sends to $P_2$ the sum $N_1 + C_1$.

2. Each $P_i$, for $2 \leq i \leq n-1$, does the following. Upon receiving a column vector $C$ from participant $P_{i-1}$, each participant $P_i$ chooses a random column vector $N_i$ and adds $N_i + C_i$ to $C$ and sends the result to $P_{i+1}$.

3. Participant $P_n$ chooses a random column vector $N_n$ and adds $N_n + C_n$ to the column he has received from $P_{n-1}$. Note that $P_n$ has the column vector $\sum_{i=1}^{n}(N_i + C_i)$.
4. The participants then pool together to recover the secret. The participants do this by each subtracting his random column vector $N_i$ from the sum $\sum_{i=1}^{n}(N_i + C_i)$.

Since it is not necessary for the group $\langle x_1, \cdots, x_m | R_1, \cdots, R_n \rangle$ to have solvable word problem, each participant can have a different group as long as every group has efficiently solvable word problem. This will ensure that the dealer can distribute words that are "random looking," which makes it difficult for an adversary to determine the shares (relators) distributed. By using groups in this scheme, rather than numbers, we have many ways to write a word as one since if $g = 1$ in a group $G$ then any product of the form $\prod_{i=1}^{m} h_i^{-1} g h_i$ for $h_i \in G$ is also equal to 1.

In order for this scheme to be practical, the dealer must be able to choose the words $w_{1j}, \cdots, w_{kj}$ efficiently for $1 \leq j \leq n$. The dealer can effectively build a word $w \in \mathrm{gp}_{F(X)}(R_i)$, where $\mathrm{gp}_{F(X)}(R_i)$ denotes the normal closure of $R_i$ in $F(X)$ in order to have $w =_{G_i} 1$ since $\mathrm{gp}_{F(X)}(R_i) \leq F(X)$. If, in addition, we

require that each participant's group is infinite, then when the dealer chooses $w \in \mathrm{gp}_{F(X)}(R_i)$ with probability close to 1 we have $w \neq_{G_k} 1$ for $k \neq i$ (see [35]). To see more about how to generate random words in such groups, we refer the reader to [7] and [31].

### 5.4.2 A $(t, n)$-threshold Scheme

The following scheme is a hybrid of the $(n, n)$-threshold scheme presented above and Shamir's secret sharing protocol. As in Shamir's scheme, the secret is an element $x \in \mathbb{Z}_p$ and the dealer chooses a polynomial $f$ of degree $t-1$ such that $f(0) = x$. In addition the dealer determines integers $y_i = f(i)$ that need to be distributed to participant $P_i$ for $1 \leq i \leq n$. A set of generators $\{x_1, \cdots, x_m\}$ is made public. The scheme is as follows.

1. The dealer distributes over a secure channel to each participant $P_j$ a set of relators $R_j$ such that each group $G_j = \langle x_1, \cdots, x_m | R_j \rangle$ has efficiently solvable word problem.

2. The dealer then distributes over open channels $k$-column vectors $b_j = \begin{pmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{kj} \end{pmatrix}$ of words in $x_1, \cdots, x_m$ to participant $P_j$ for $1 \leq j \leq n$ . The column vectors $b_j$ are chosen so that in

participant $P_j$'s group precisely $y_j$ of the words are equal to 1.

3. Participant $P_j$ then checks if each word $b_{ij} = 1$ is his or her group $G_j$. The total number of words $b_{ij} = 1$ in $G_j$ is the number $y_j$.

4. Each participant now has a point $f(i) = y_i$ of the polynomial. Using polynomial interpolation any $t$ participants can now recover the polynomial $f$, and hence the secret $f(0)$.

Again, by the dealer's choice of words, it is clear that the participants will construct the secret correctly if we assume that the shares have not been corrupted. Hence, this scheme is robust.

This scheme has the advantage over Shamir's scheme that the shares (relators) do not depend on the secret. The scheme has the additional advantage that the values $x_i$ can be sent over open channels rather than secure channels.

### 5.4.3 Cryptanalysis

In order to protect long term secrets, we suggest the idea of proactive secret sharing. By utilizing this method an adversary or dishonest participant has only a short period of time to recover a participants share rather than an unlimited amount of time, enhancing

the security of the scheme. To implement the notion of proactive secret sharing, we suggest utilizing Tietze transformations. By employing Tietze transformations we can replace the relators $R_i$ with a new set of relators $R_i^{'}$ as long as $\mathrm{gp}_{F(X)}(R_i) = \mathrm{gp}_{F(X)}(R_i^{'})$. This will ensure that $w =_{G_i} 1$ iff $w =_{G_i^{'}} 1$ where $G_i$ is the original group of participant $P_i$ and $G_i^{'}$ is the modified group of participant $P_i$. Hence, we will change each participants share without changing the secret.

Each participant will renew his or her shares using a Tietze transformation that introduces a new generator $y$ and a new relator $y = w(x_1, \cdots, x_m)$. The method in which this is done modifies the share renewal protocol presented in [23] to use Tietze transformations. The protocol to renew shares is as follows:

1. Participant $P_i$ chooses Tietze transformations $\phi_{ij}$ of the type mentioned above, where $1 \leq j \leq n$ and $j \neq i$.

2. Participant $P_i$ secretly sends the Tietze transformation $\phi_{ij}$ to participant $P_j$ for each $j$.

3. Participant $P_i$ now has a sequence of Tietze transformations $\phi_{ji}$ where $1 \leq j \leq n$ and $j \neq i$. From the $\phi_{ji}$, participant $P_i$ can form a Tietze transformation $\phi_i$ by taking the composition of the Tietze transformations $\phi_{ji}$.

4. Participant $P_i$ will then compute his or her new share by applying the Tietze transformation $\phi_i$ to the group $G_i$, and will then delete his or her old share.

To ensure that the word problem will remain efficiently solvable after the share renewal is performed, we suggest that each participant $P_i$ stores the Tietze transformation $\phi_i$.

### 5.4.4 Security Assumptions

The $(n, n)$-threshold scheme and the $(t, n)$-threshold schemes proposed are secure in the presence of an eavesdropping attacker. In both schemes we suppose that the relators are transmitted to each participant in complete secrecy, and hence an eavesdropping attacker cannot gain any information about the relators.

#### $(n, n)$-threshold Scheme

Under the assumption that the relators are distributed in complete secrecy, an eavesdropping adversary can gain no information about the secret from the public information. Brute force attacks are infeasible for sufficiently large $k$ since there are $2^k$ choices for the secret. The size of the column vector can easily be increased to

prevent brute force attacks by having the dealer increase the number of words distributed. We would like to note that after repeated applications of this protocol without using the secure sum, there may be a leak of information. This is due to the fact that each participant knows the secret and the words sent by the dealer, the participants may be able to determine some information about the others relators.

#### $(t, n)$-threshold Scheme

Under the assumption that the relators are distributed in complete secrecy an eavesdropping adversary can gain no information about the secret from the public information. Since the relators are sent in complete secrecy and the words published are "random-lookinh" due to the nature of the protocol, the adversary cannot determine whether a published word $w_{ij} =_{G_j} 1$. Hence, the adversary cannot determine any number $y_j = f(j)$ needed to determine the secret from the public information. In addition, for large primes $p$ brute force attacks used to recover the secret $x \in \mathbb{Z}_p$ are infeasible.

# Chapter 6

# Matrix Representations

In this chapter we will determine the computational complexity of an algorithm that determines a $\mathbb{Q}$-basis for a finite dimensional faithful $G$-module. The basis produced gives matrix representations of elements in a finitely generated torsion free nilpotent group (see [37]). The complexity of this algorithm gives an upper bound on the dimension of the matrices produced. In addition to determining the complexity of this algorithm, we also provide an algorithm based on the one proposed in [37] that provides a $\mathbb{Q}$-basis for the $G$-module generated by certain polynomials. We determine the computational complexity of this new algorithm.

## 6.1 Matrix Representations of Finitely Generated Torsion-free Nilpotent Groups

For the remainder of the chapter we will follow the exposition of [18].

Recall that polycyclic groups were proposed as a potential platform group for certain cryptographic protocols (see [12], [28], [10]) since they have solvable word problem and the search conjugacy problem is conjectured to be exponential time. We know that the search conjugacy problem is solvable in a polycyclic group if one utilizes an embedding $\rho : G \to GL_n(\mathbb{Z})$, from a polycyclic group $G$ into $GL_n(\mathbb{Z})$, and then solves the search conjugacy problem for $G$ in the group $GL_n(\mathbb{Z})$. Since the search conjugacy problem for $GL_n(\mathbb{Z})$ is solvable in polynomial time, the complexity of the search conjugacy problem in polycyclic groups relies on the complexity of the embedding. For the remainder of this chapter, we will describe an algorithm presented in [37] that determines matrix representations of finitely generated torsion free nilpotent groups and then determine the complexity of the crux of this algorithm. In addition, we will provide a modification of the algorithm to provide a $\mathbb{Q}$-basis for a finite dimensional faithful $G$-module presented in [37] that runs in linear time. We will begin by recalling some facts

about finitely generated torsion free nilpotent groups.

Let $G$ be a finitely generated torsion-free nilpotent group. Since $G$ is torsion-free and finitely generated nilpotent, there exists a central series

$$G = G_1 \triangleright G_2 \triangleright G_3 \triangleright \cdots \triangleright G_{n+1} = \{1\}$$

such that for each $1 \leq r \leq n$ the factor $G_r/G_{r+1}$ is infinite cyclic generated by $x_r G_{r+1}$ for some $x_r \in G_r$. Given such $x_1, \cdots, x_n$ each element $x \in G$ has unique normal form

$$x = x_1^{e_1} \cdots x_n^{e_n},$$

where $e_i \in \mathbb{Z}$. The $n$-tuple $(e_1, \cdots, e_n)$ is called the vector of exponents of $x$.

As mentioned earlier all finitely generated nilpotent groups are polycyclic, and hence admit a polycyclic presentation. Recall that every finitely generated nilpotent group admits a polycyclic presentation with conjugacy relations of the form:

$$x_i^{x_j} = x_i x_{i+1}^{b_{i,j,i+1}} \cdots x_n^{b_{i,j,n}} \text{ for } 1 \leq j < i \leq n,$$

$$x_i^{x_j^{-1}} = x_i x_{i+1}^{c_{i,j,i+1}} \cdots x_n^{c_{i,j,n}} \text{ for } 1 \leq j < i \leq n$$

where $b_{i,j,k}, c_{i,j,k} \in \mathbb{Z}$ for $1 \leq j < i < k \leq n$. A polycyclic presentation with these conjugacy relations is called a *nilpotent presentation*. Since $G$ is torsion free, there exists a consistent nilpotent

presentation with each $r_i = \infty$. Hence, for any finitely generated torsion-free nilpotent group we may find a consistent nilpotent presentation of the form:

$$\left\langle x_1, \cdots, x_n; x_i^{x_j} = x_i x_{i+1}^{b_{i,j,i+1}} \cdots x_n^{b_{i,j,n}}, x_i^{x_j^{-1}} = x_i x_{i+1}^{c_{i,j,i+1}} \cdots x_n^{c_{i,j,n}} \text{ for } 1 \leq j < i \leq n \right\rangle.$$

Let $G$ be a finitely generated torsion free nilpotent group with a presentation as above. Then each $x \in G$ may be written uniquely in the form $x = x_1^{e_1} \cdots x_n^{e_n}$. Given two elements $x = x_1^{e_1} \cdots x_n^{e_n}$, $y = x_1^{y_1} \cdots x_n^{y_n}$, their product can be written uniquely as:

$$xy = x_1^{e_1} \cdots x_n^{e_n} x_1^{y_1} \cdots x_n^{y_n} = x_1^{z_1} \cdots x_n^{z_n}.$$

Philip Hall showed that there are rational polynomials $f_1, \cdots, f_n$ that describe the multiplication of elements in $G$, a finitely generated torsion free nilpotent group (see [22]). In particular, if $(e_1, \cdots, e_n)$, $(y_1, \cdots, y_n)$, and $(z_1, \cdots, z_n)$ are the vectors of exponents for $x, y$, and $xy$, respectively, then for $1 \leq r \leq n$ we have

$$z_r = f_r(e_1, \cdots, e_n, y_1, \cdots, y_n).$$

In [32] Leedham-Green and Soicher show how to compute polynomials in variables corresponding to the $e_i, y_i, c_{i,j,k}$ such that they describe the multiplication of elements in the group $G$ by an algorithm referred to as "Deep Thought". In [37] W. Nickel presents an algorithm for computing a representation of a finitely generated

torsion free nilpotent group given by a polycyclic presentation by unitriangular matrices over $\mathbb{Z}$. The algorithm by Nickel uses the polynomials computed by "Deep Thought." There is another algorithm due to DeGraaf and Nickel that computes a faithful unitriangular representation of finitely generated torsion-free nilpotent groups in [16], but the algorithm presented by Nickel in [37] is more efficient. The crux of the algorithm in [37] computes a $\mathbb{Q}$-basis for a finite dimensional faithful $G$-module, where $G$ is a finitely generated torsion-free nilpotent group given by a nilpotent presentation.

In [37], a faithful finite dimensional $G$-module is constructed as follows. Let $G$ be a finitely generated torsion free nilpotent group with nilpotent generating sequence $a_1, \cdots, a_n$ and multiplication polynomials $q_1, \cdots, q_n$. Let $G$ act on the dual of the group ring $(\mathbb{Q}G)^*$ by defining $f^g(h) = f(hg^{-1})$ for $f \in (\mathbb{Q}G)^*$ and $g, h \in G$. By identifying $a_1^{x_1} \cdots a_n^{x_n}$ with $x_1, \cdots, x_n$ the image of $f \in (\mathbb{Q}G)^*$ under the action of $G$ can be described using the multiplication polynomials $q_1, \cdots, q_n$; that is, $f^g = f(q_1, \cdots, q_n)$. With this action one can construct a finite dimensional faithful $G$-submodule of $(\mathbb{Q}G)^*$ by using the following lemma.

**Lemma 6.1.1.** *[37] The submodule $M$ of $(\mathbb{Q}G)^*$ generated by $t_i$ :*

*$G \to \mathbb{Z}$ with $t_i(a_1^{x_1} \cdots a_n^{x_n}) = x_i$ is a finite dimensional faithful $G$-module.*

To construct a basis a method called *Insert* is used. *Insert* adds a polynomial to a given basis of polynomials if that polynomial is not in the span of the basis; that is, if the polynomial cannot be written as a linear combination of the basis elements. This is done by first placing an ordering on the monomials. The leading monomial of a polynomial is then the monomial that is largest with respect to the ordering. *Insert* takes a basis of polynomials in ascending order and a polynomial $f$, and determines if $f$ is in the span of the basis. If $f$ is not in the span of the basis, then it will be added to the basis. *Insert* determines if $f$ is in the span of the basis by subtracting the appropriate multiple of each polynomial in the basis, starting with the polynomial that is largest with respect to the ordering and continuing in descending order. If $f \neq 0$ after this process is completed, then $f$ is not in the span of the basis and is added to the original basis.

In order to determine a $\mathbb{Q}$-basis for a finite dimensional $G$-module generated by polynomials $f_1, \cdots, f_k$, the algorithm first uses *Insert* to build up a basis from $f_1, \cdots, f_k$. Then the algorithm works up the central series of $G$ beginning with $G_n$. For

each $j$ the algorithm will compute a basis for the $G_j$-module $M_j$ from a basis of the $G_{j+1}$-module $M_{j+1}$ generated by $f_1, \cdots, f_k$. In order to obtain a generating set for $M_j$ it is enough to close the module $M_{j+1}$ under the action of powers of $a_j$ (since $G_j/G_{j+1}$ is cyclic) and to apply powers of $a_j$ to each basis element of $M_{j+1}$. The algorithm for computing a $\mathbb{Q}$-basis is presented in Figure 6.1.

The matrix representation for each generator $a_j$ of $G$ can be calculated by decomposing the image of each basis element under $a_j$ in terms of the basis. The ordering utilized in *Insert* is the reverse lexicographic ordering; that is, $x_1^{k_1} \cdots x_n^{k_n} < x_1^{l_1} \cdots x_n^{l_n}$ if there is an $i$ $(1 \leq i \leq n)$ such that $k_j = l_j$ for $i < j \leq n$ and $k_i < l_i$. Suppose that $b_1, \cdots, b_m$ is the basis produced using this ordering. Let $q_1^{(j)}, \cdots q_n^{(j)}$ be the polynomials that describe the multiplication of an arbitrary element of the group with $a_j^{-1}$:

$$a_1^{x_1} \cdots a_n^{x_n} a_j^{-1} = a_1^{q_1^{(j)}} \cdots a_n^{q_n^{(j)}}$$

The polynomials $q_i^{(j)}$ are obtained from the multiplication polynomials $q_i$ by setting $y_i = 0$ if $i \neq j$ and $y_i = -1$ if $i = j$, which we denote $y_i = -\delta_{ij}$. For each basis element $b_k$, one can compute $b_k(q_1^{(j)}, \cdots, q_n^{(j)})$ and decompose this element into a linear combination $c_{k_1} b_1 + \cdots + c_{k_n} b_m$ of basis vectors. The coefficients $c_{k_1}, \cdots, c_{k_n}$

are the $k^{th}$ row of the matrix representing $a_j$. For more information on this algorithm see [37].

This algorithm was implemented in the GAP-package polycyclic. The run times of this algorithm as well as the algorithm in [16] are illustrated in Table 6.1. The times listed are in milliseconds and HL denotes the Hirsch length of the group. Note that $U_n(\mathbb{Z})$ denotes the unitriangular group of dimension $n$ over $\mathbb{Z}$ and $F(k, c)$ denotes the free nilpotent group with $k$ generators and nilpotentcy class $c$.

Table 6.1: Experimental Results [37]

| | | | Algorithm in [16] | | Algorithm in [37] | |
|---|---|---|---|---|---|---|
| Group | Class | HL | Dimension | time | Dimension | time |
| $U_2(\mathbb{Z})$ | 1 | 1 | 2 | 10 | 2 | 10 |
| $U_3(\mathbb{Z})$ | 2 | 3 | 3 | 10 | 4 | 20 |
| $U_4(\mathbb{Z})$ | 3 | 6 | 7 | 220 | 8 | 90 |
| $U_5(\mathbb{Z})$ | 4 | 10 | 16 | 6220 | 16 | 390 |
| $U_6(\mathbb{Z})$ | 5 | 15 | 35 | 101810 | 28 | 1310 |
| $F(2,2)$ | 2 | 3 | 3 | 20 | 4 | 30 |
| $F(2,3)$ | 3 | 5 | 6 | 130 | 7 | 70 |
| $F(2,4)$ | 4 | 8 | 10 | 1460 | 11 | 170 |
| $F(2,5)$ | 5 | 14 | 20 | 65290 | 21 | 1130 |
| $F(3,2)$ | 2 | 6 | 6 | 120 | 7 | 80 |
| $F(3,3)$ | 3 | 14 | 17 | 8090 | 18 | 520 |

Figure 6.1: Matrix Representation Algorithm (building a basis), [37]

**Input:** A finitely generated torsion free nilpotent group $G$ with a nilpotent generating sequence $a_1, \cdots, a_n$ and corresponding multiplication polynomials $q_1, \cdots, q_n$; a list of polynomials $f_1, \cdots, f_k$.

**Output:** A $\mathbb{Q}$-basis $B$ for the $G$-module generated by $f_1, \cdots, f_k$.

**Algorithm:**
$B := []$;
**for** $j$ **in** $[1 \cdots k]$ **do** Insert$(B, f_j)$;**od**;
  **for** $j$ **in** $[n, n-1 \cdots 1]$ **do**
    #B is a basis for $M_{j+1}$
    # Exponents after multiplication by $a_j^{-1}$ from the right:
    $q^{(j)} := [q_1(x_1, \cdots, x_n, y_i = -\delta_{ij}), \cdots, q_n(x_1, \cdots, x_n, y_i = -\delta_{ij})]$;
    **for** $f$ **in** Copy$(B)$ **do**
      # Add to $B$ images of $f$ under powers of $a_j$
      **repeat**
          # Compute $f^{a_j}$
          $f^{a_j} := f(q_1^{(j)}, \cdots, q_n^{(j)})$;
          r:=Insert$(B, f^{a_j})$;
          $f := f^{a_j}$;
      **until** r=0;
    **od**;
  **od**;

**return** $B$;

## 6.2 Complexity Analysis

The algorithm for finding matrix representations of torsion free finitely generated nilpotent groups formulated in [37] was implemented in the GAP package polycyclic. The running times of the algorithm in Table 6.1 include the computation time of determining the multiplication polynomials by "Deep Thought", the

building of a $\mathbb{Q}$-basis, and the construction of the matrix. The algorithm was implemented using the coordinate functions $t_i : G \to \mathbb{Z}$ given by $a_1^{x_1} \cdots a_n^{x_n} \mapsto x_i$ for $1 \leq i \leq n$ as the input polynomials $f_1, \cdots, f_k$ . In order to analyze the complexity of the algorithm utilized for building a $\mathbb{Q}$-basis (see Figure 6.1), we must analyze how the action of $a_j \in G$ affects each coordinate function $t_i$. Recall that $t_i^{a_j} := t_i(q_1^{(j)}, \cdots, q_n^{(j)})$ where $q_i^{(j)} = q_i(x_1, \cdots, x_n, y_i = -\delta_{ij})$ and the $q_i$ are the multiplication polynomials computed via "Deep Thought". From the nature of the polynomials $q_i$ it follows that $q_i^{(j)} = x_i$ for $1 \leq i < j$, $q_j^{(j)} = x_j - 1$, and $q_i^{(j)} = x_i + \overline{q_i}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$. To see this, we will follow the exposition of [37]. We may rewrite the product, $a_1^{x_1} \cdots a_n^{x_n} a_1^{y_1} \cdots a_n^{y_n}$, of two elements $x = a_1^{x_1} \cdots a_n^{x_n}$, $y = a_1^{x_1} \cdots a_n^{x_n} \in G$ as

$$(a_1^{x_1} \cdots a_i^{x_i})(a_{i+1}^{x_{i+1}} \cdots a_n^{x_n} a_1^{y_1} \cdots a_i^{y_i})(a_{i+1}^{y_{i+1}} \cdots a_n^{y_n}),$$

which is equal to

$$(a_1^{x_1} \cdots a_i^{x_i} a_1^{y_1} \cdots a_i^{y_i})(a_{i+1}^{'x_{i+1}} \cdots a_n^{'x_n} a_{i+1}^{y_{i+1}} \cdots a_n^{y_n})$$

since $G_{i+1}$ is normal in $G$. The expression $a_{i+1}^{'x_{i+1}} \cdots a_n^{'x_n} a_{i+1}^{y_{i+1}} \cdots a_n^{y_n}$ can be computed in $G_{i+1}$ and does not involve $a_1 \cdots a_i$. Hence, $q_i$ is determined by $a_1^{x_1} \cdots a_i^{x_i} a_1^{y_1} \cdots a_i^{y_i}$ and $q_i$ only depends on $x_1, \cdots, x_i, y_1 \cdots, y_i$. Since $a_i$ is central in $G/G_{i+1}$ we have that

$q_i = x_i + y_i + \overline{q_i}$ with $\overline{q_i} \in \mathbb{Q}[x_1, \cdots, x_{i-1}, y_1, \cdots, y_{i-1}]$. Since multiplying $a_1^{x_1} \cdots a_n^{x_n}$ by $a_j^{-1}$ from the right does not affect $a_1, \cdots, a_{j-1}$ we have that $q_i^{(j)} = x_i$ for $1 \leq i < j$. Moreover, we have $q_j^{(j)} = x_j - 1$ and by entering $y_i = -\delta_{ij}$ we have $q_i^{(j)} = x_i + \overline{q_i}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$.

In order to understand the action of each $a_j$ on each $t_i$ we look at three cases: $1 \leq i < j$, $i = j$, and $j < i \leq n$.

1. $1 \leq i < j$:

$$\begin{aligned} t_i^{a_j} &:= t_i(q_1^{(j)}, \cdots, q_n^{(j)}) \\ &= t_i(x_1, \cdots, x_j - 1, x_{j+1} + \overline{q_{j+1}}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_n}(x_1, \cdots, x_{n-1})) \\ &= x_i \\ &= t_i \end{aligned}$$

2. $i = j$:

$$\begin{aligned} t_j^{a_j} &:= t_j(q_1^{(j)}, \cdots, q_n^{(j)}) \\ &= t_i(x_1, \cdots, x_j - 1, x_{j+1} + \overline{q_{j+1}}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_n}(x_1, \cdots, x_{n-1})) \\ &= x_j - 1 \\ &= t_j - 1 \end{aligned}$$

3. $j < i \leq n$:

$$\begin{aligned} t_i^{a_j} &:= t_i(q_1^{(j)}, \cdots, q_n^{(j)}) \\ &= t_i(x_1, \cdots, x_j - 1, x_{j+1} + \overline{q_{j+1}}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_n}(x_1, \cdots, x_{n-1})) \\ &= x_i + \overline{q_i}(x_1, \cdots, x_{i-1}) \\ &= t_i + \overline{q_i}(x_1, \cdots, x_{i-1}) \end{aligned}$$

Figure 6.2:

**for** $j$ **in** $[n, n-1 \cdots 1]$ **do**
  #B is a basis for $M_{j+1}$
  # Exponents after multiplication by $a_j^{-1}$ from the right:
  $q^{(j)} := [q_1(x_1, \cdots, x_n, y_i = -\delta_{ij}), \cdots, q_n(x_1, \cdots, x_n, y_i = -\delta_{ij})]$;
  **for** $f$ **in** Copy($B$) **do**
    # Add to $B$ images of $f$ under powers of $a_j$
    **repeat**
      # Compute $f^{a_j}$
      $f^{a_j} := f(q_1^{(j)}, \cdots, q_n^{(j)})$;
      r:=Insert($B, f^{a_j}$);
      $f := f^{a_j}$;
    **until** r=0;
  **od**;
**od**;

By utilizing this information on the action of $a_j$ on each $t_i$ we are able to determine the complexity of the algorithm presented in Figure 6.1 when the coordinate functions are used. We begin by finding a bound on the size of the matrices produced. It is clear from the algorithm presented in [37] that the size of the matrix representation produced depends on the size of the $\mathbb{Q}$-basis constructed using the algorithm presented in Figure 6.1. To determine a bound, we begin by analyzing a single loop of the algorithm in Figure 6.1. We present this loop in Figure 6.2. The number of iterations the loop in Figure 6.2 undergoes throughout the algorithm is directly related to the size of the basis produced.

**Theorem 6.2.1.** *[18] Let $G$ be a finitely generated torsion-free*

*nilpotent group. The $\mathbb{Q}$-basis of the $G$-module generated by $t_1, \cdots, t_n$ has at most $2n$ elements, where $n$ is the Hirsch length of the group $G$.*

*Proof.* We will prove the basis will have cardinality at most $2n$ using induction on the number of times the loop in Figure 6.2 is repeated. Recall that $t_i^{a_j} := t_i(q_1^{(j)}, \cdots, q_n^{(j)})$ where $q_i^{(j)} = x_i$ for $1 \leq i < j$, $q_j^{(j)} = x_j - 1$, and $q_i^{(j)} = x_i + \overline{q_i}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$.
First note that for $2 \leq i \leq n$ we may write $\overline{q_i} = r_{n-i+2} + \sum_{j=1}^{n} c_j t_j$ for appropriate $c_j \in \mathbb{Q}$, $r_i \in \mathbb{Q}[x_1, \cdots, x_{i-1}]$, where $r_i = 0$ or cannot be written as a linear combination of the coordinate functions $t_k$ for $1 \leq k \leq n$.

The algorithm presented in [37] (see Figure 6.1) begins by building a basis from the coordinate functions $t_1, \cdots, t_n$ using *Insert*. By definition $t_i(x_1, \cdots, x_n) = x_i$; hence, *Insert* will begin by forming the basis $B = \{t_1, \cdots, t_n\}$.

Since we are considering the worst case scenario, we will assume that each polynomial $r_i$ is not a linear combination of $t_1, \cdots, t_n, r_j$ for $j \neq i$.

*Base case*: The loop in Figure 6.2 is iterated one time.

The loop begins by adding images of each $t_i$ under the action of $a_n$. We may assume that the algorithm begins by adding images of $t_n$ under the action of $a_n$ to the basis. We know that

$$\begin{aligned} t_n^{a_n} =& t_n(x_1, \cdots, x_{n-1}, x_n - 1) \\ =& x_n - 1 \\ =& t_n - 1, \end{aligned}$$

which clearly is not in the span of the basis. *Insert*$(B,\ t_n^{a_n})$ will begin by subtracting $t_n$ from $t_n - 1$, leaving $r = -1$ which will be added to the basis. The algorithm then defines $t_n := t_n^{a_n}$ and $t_n^{a_n} = (t_n - 1) - 1 = t_n - 2$, which is the span of the basis; that is $r = 0$ and the process terminates. For $1 \leq i \leq n-1$, we have that $t_i^{a_n} = t_i$. Hence, for $t_i$ with $1 \leq i \leq n-1$ no new polynomials will be added to the basis. After this process is completed for $a_n$ the resulting basis will be $B = \{t_1, \cdots, t_n, r_1 = -1\}$. Note that in order to close the module under the action of $a_n$ the loop that adds images of each $t_i$ in Copy$(B)$ under powers of $a_n$ is repeated $n+1$ times.

In order to ensure a clear understanding of the algorithm, we will also consider the case where the loop is repeated twice.

The second repetition of the loop repeats the process for $a_{n-1}$. We may assume that the process begins by adding images of $t_n$ under

the action of $a_{n-1}$ to the basis. From above we know that

$$\begin{aligned} t_n^{a_{n-1}} =& t_n(x_1, \cdots, x_{n-1} - 1, x_n + \overline{q_n}(x_1, \cdots, x_{n-1})) \\ =& x_n + \overline{q_n}(x_1, \cdots, x_{n-1}) \\ =& t_n + \overline{q_n} \\ =& t_n + r_2 + \sum_{j=1}^{n} c_j t_j. \end{aligned}$$

$Insert(B, t_n^{a_{n-1}})$ will subtract appropriate multiples of $t_1, \cdots, t_n$ from $t_n^{a_{n-1}}$ leaving $r = r_2$ to be added to the basis. The algorithm then defines
$t_n := t_n^{a_{n-1}}$, and we have $t_n^{a_{n-1}} = (t_n + \overline{q_n}) + \overline{q_n} = t_n + 2\overline{q_n} = t_n + 2r_2 + 2\sum_{j=1}^{n} c_j t_j$, which is in the span of the basis. For $1 \leq i \leq n-2$ we have $t_i^{a_{n-1}} = t_i$ and $t_{n-1}^{a_{n-1}} = x_{n-1} - 1 = t_{n-1} + r_1$, which are in the span of the basis. Hence, after two iterations of the loop the resulting basis is $B = \{t_1, \cdots, t_n, r_1, r_2\}$. Note that in order to close the module under the action of $a_{n-1}$ the loop that adds images of each $t_i$ in Copy$(B)$ under powers of $a_{n-1}$ is repeated $n+1$ times.

*Inductive assumption:* Suppose for $k < n$ iterations of the loop the resulting basis is $B = \{t_1, \cdots, t_n, r_1, r_2, \cdots, r_k\}$.

$(k+1)^{st}$ *iteration of the loop*: The loop in Figure 6.2 will be

performed for $a_{n-k}$.

We begin by noting that

$$\begin{aligned} t_i^{a_{n-k}} &:= t_i(q_1^{(n-k)}, \cdots, q_n^{(n-k)}) \\ &= t_i(x_1, \cdots, x_{n-k} - 1, x_{n-k+1} + r_{n-(n-k+1)+2} + \sum_{i=1}^{n} c_i t_i, \cdots, x_n + r_{n-(n)+2} + \sum_{i=1}^{n} d_i t_i) \\ &= t_i(x_1, \cdots, x_{n-k} - 1, x_{n-k+1} + r_{k+1} + \sum_{i=1}^{n} c_i t_i, \cdots, x_n + r_2 + \sum_{i=1}^{n} d_i t_i) \end{aligned}$$

We would like to see which polynomials will be added to the basis in this step. We will look at four cases:

- $1 \leq i \leq n-k-1$:

  $t_i^{a_{n-k}} = x_i = t_i$. This is in the span of the basis, and so the algorithm will have $r = 0$ and the process terminates.

- $i = n-k$:

  $t_{n-k}^{a_{n-k}} = x_{n-k} - 1 = t_{n-k} + r_1$. This is in the span of the basis, and so the algorithm will have $r = 0$ and the process terminates.

- $i = n-k+1$:

  $t_{n-k+1}^{a_{n-k}} = x_{n-k+1} + r_{k+1} + \sum_{i=1}^{n} c_i t_i$. By assumption $r_{k+1}$ is not in the span of the basis. The algorithm will then add $r_{k+1}$ to the basis, resulting in $B = \{t_1, \cdots, t_n, r_1, r_2, \cdots, r_{k+1}\}$. Then

$t_{n-k} := t_{n-k}^{a_{n-k}}$ and we have

$$\begin{aligned} t_{n-k}^{a_{n-k}} =& (x_{n-k+1} + r_{k+1} + \sum_{i=1}^{n} c_i t_i) + r_{k+1} + \sum_{i=1}^{n} c_i t_i \\ =& x_{n-k+1} + 2r_{k+1} + 2\sum_{i=1}^{n} c_i t_i \end{aligned}$$

This is in the span of the basis, and so the algorithm will have $r = 0$ and the process terminates.

- $i = n - k + j$ for $2 \leq j \leq k$:

$$\begin{aligned} t_{n-k+j}^{a_{n-k}} =& x_{n-k+j} + r_{n-(n-k+j)+2} + \sum_{i=1}^{n} d_i t_i \\ =& x_{n-k+j} + r_{k-j+2} + \sum_{i=1}^{n} d_i t_i \end{aligned}$$

As $j$ runs from 2 to $k$, $k - j + 2$ runs from $k$ to 2 and so we have that $x_{n-k+j} + r_{k-j+2} + \sum_{i=1}^{n} d_i t_i$ is in the span of the basis for each $j$.

Hence, for any $k \leq n$ we know after $k$ iterations the resulting basis will be $B = \{t_1, \cdots, t_n, r_1, \cdots, r_k\}$. Note that in order to close the module under the action of $a_{n-k}$ the loop that adds images of each $t_i$ in Copy($B$) under powers of $a_{n-k}$ is repeated $n+1$ times. The loop in Figure 6.2 will be repeated $n$ times (one for each element in the nilpotent generating sequence), and the resulting basis will be $B = \{t_1, \cdots, t_n, r_1, \cdots, r_n\}$, as desired. □

**Corollary 6.2.2.** *[18] When the algorithm presented in [37] is implemented using the coordinate functions $t_i$ for $1 \leq i \leq n$ the worst case dimension of the matrix representation produced depends linearly on the Hirsch length of the group; that is, the dimension of the matrix representation is $O(n)$ where $n$ is the Hirsch length of the group.*

*Proof.* This follows immediately from Theorem 6.2.1. □

**Theorem 6.2.3.** *[18] The running time of the algorithm presented in Figure 6.1 using the coordinate functions $t_i$ for $1 \leq i \leq n$ is $O(n^2)$, where $n$ is the Hirsch length of the group.*

*Proof.* Before determining the worst case complexity of the algorithm presented in Figure 6.1, we must determine the number of steps needed by the method *Insert*. *Insert* takes a basis of polynomials in ascending order with respect to reverse lexicographic ordering and a polynomial $f$, and determines if $f$ is in the span of the basis by subtracting the appropriate multiple of each polynomial in the basis, starting with the polynomial that is largest with respect to the ordering and continuing in descending order. If $f \neq 0$ after this process is completed, then $f$ is not in the span of the basis and is added to the original basis. It is clear that the number of subtractions used on a given polynomial $f$ by *Insert* is

at most the number of terms in $f$. We will denote the number of terms in a given polynomial $f$ by $m_f$.

Recall the algorithm to form a $\mathbb{Q}$-basis of the $G$-module generated by the coordinate functions $t_i$ for $1 \leq i \leq n$ begins by building a basis from the coordinate functions $t_1, \cdots, t_n$ by implementing $Insert(B,\ t_i)$ for $1 \leq i \leq n$. It is clear from the above remarks that this process will take a total of $\sum_{i=1}^{n} m_{t_i}$ steps. The next step to determine the complexity of the algorithm for building a $\mathbb{Q}$-basis is to determine how many iterations of the inner loop in Figure 6.2 occur for each $a_j$. First note that the outer loop is repeated $n$ times since the nilpotent generating sequence has $n$ elements. In the proof of Theorem 6.2.1, one can see that closing the module under the action of each $a_j$ requires at most $n+1$ repetitions of *Insert*. Let $m$ denote the maximum number of steps that *Insert* requires at any step. Then to close the module under $a_j$ the maximum number of steps required is bounded above by $m(n+1)$. Hence, the loop in Figure 6.2 takes at most $nm(n+1)$ steps and the total number of steps required by the algorithm is at most $\sum_{i=1}^{n} m_{t_i} + m(n^2+n) \leq m(n^2+2n)$. Thus, the total running time of the algorithm is $O(n^2)$.

□

## 6.3 Improved Algorithm

From Theorem 6.2.1, we see that the $\mathbb{Q}$-basis of the $G$-module generated by the coordinate functions $t_1, \cdots, t_n$ is a subset of $B = \{t_1, \cdots, t_n, r_1, \cdots, r_n\}$, where $r_i$ are as above. Using this fact we may improve the algorithm presented by Nickel in [37].

In order to find a $\mathbb{Q}$-basis of the $G$ module generated by the coordinate functions $t_1, \cdots, t_n$ we must begin by building a basis up from the coordinate functions using *Insert* as is done in [37]. By the nature of the coordinate functions, it is clear that the action of each $a_j$ for $1 \leq j \leq n$ will only add new polynomials to the basis at certain steps in the algorithm. Recall that in the first time the loop in Figure 6.2 is repeated the only polynomial that could be added to the basis is when $Insert(B, t_n^{a_n})$ is applied. Then in the $k+1^{st}$ iteration of the loop the only polynomial that could be added to the basis arises from $Insert(B, t_{n-k+1}^{a_{n-k}})$. This is clear in the proof of Theorem 6.2.1. Hence, in order to determine the $\mathbb{Q}$-basis of the $G$ module generated by $t_1, \cdots, t_n$ it is enough to implement *Insert* for only these polynomials; that is, we need only perform

Figure 6.3: Building a Basis for a $G$-module using Coordinate Functions, [18]

**Input:** A finitely generated torsion-free nilpotent group $G$ with nilpotent generating sequence $a_1, \cdots, a_n$ and multiplication polynomials $q_1, \cdots, q_n$; coordinate functions $t_1, \cdots, t_n$.
**Output:** A $\mathbb{Q}$-basis $B$ for the $G$-module generated by $t_1, \cdots, t_n$.
**Algorithm:**
$B := []$;
  **for** $j$ **in** $[1, \cdots n]$ **do** $Insert(B, t_j)$; **od**;
  **do** $Insert(B, t_n^{a_n})$; **od**;
  **for** $j$ **in** $[n, n-1, \cdots, 2]$ **do** $Insert(B, t_j^{a_{j-1}})$; **od**;
**return** $B$;

$Insert(B, t_n^{a_n})$ and $Insert(B, t_j^{a_{j-1}})$ for $2 \leq j \leq n$. The pseudo-code for the algorithm is displayed in Figure 6.3.

**Theorem 6.3.1.** *[18] The algorithm presented in Figure 6.3 has running time $O(n)$, where $n$ is the Hirsch length of the group.*

*Proof.* We denote the maximum number of subtractions done by $Insert$ at any step in the algorithm by $m$. Then the number of steps required for $Insert(B, t_i)$ for $1 \leq i \leq n$ is bounded above by $mn$, the number of steps of $Insert(B, t_n^{a_n})$ is bounded above by $m$ and the number of steps of $Insert(B, t_j^{a_{j-1}})$ for $2 \leq j \leq n$ is bounded above by $m(n-1)$. Therefore the total running time of the algorithm is bounded above by
$mn + m + m(n-1) = 2mn$, and the result follows. □

## 6.4 Remarks

We would like to note that we have shown that the algorithm to build a $\mathbb{Q}$-basis for a finite dimensional faithful $G$-module, where $G$ is a finitely generated torsion-free nilpotent group given by a nilpotent presentation, in [37] has quadratic time complexity with respect to the Hirsch length if the coordinate functions are utilized, and that the dimension of the matrix representation produced depends linearly on the Hirsch length of the group. It is possible to improve the time complexity and the bound on the dimension by using a different set of functions rather than the coordinate functions. Unfortunately, there is not a known way to choose these functions.

The improved algorithm is only for generating a $\mathbb{Q}$-basis for the $G$-module generated by the coordinate functions unlike the algorithm presented in [37], which works for a general set of functions $f_1, \cdots, f_k$. Hence, while our algorithm is more efficient the algorithm presented in [37] is more general. Although it improved the running time of the algorithm presented in [37], it does not affect the dimension of the matrix representation produced.

# Bibliography


[1] Iris Anshel, Michael Anshel, and Dorian Goldfeld, *An algebraic method for public-key cryptography*, Math. Res. Lett. (1999), 6:287–291.

[2] Sanjeev Arora and Boaz Barak, *Computation complexity: A modern approach*, Cambridge University Press, 2009.

[3] Gregory V. Bard, *Algebraic cryptanalysis*, Springer, 2009.

[4] B.Fine, Alexei Myasnikov, and G.Rosenberger, *Generic subgroups of group amalgams*, Groups Complexity Cryptology **1** (2009), no. 1, 51–61.

[5] Simon R. Blackburn, Carlos Cid, and Ciaran Mullan, *Cryptanalysis of three matrix-based key establishment protocols*, arXiv:1102.2358v1 [math.GR] (2011), 1–9.

[6] D. Boneh and R. Venkatesan, *Breaking rsa may not be equivalent to factoring*, Advances in Cryptology - Eurocrypt '98 (1998), 59–71.

[7] Alexandre V. Borovik, Alexei G. Myasnikov, and Vladimir Shpilrain, *Measuring sets in infinite groups*, Computational and Statistical Group Theory (Las Vegas, NV/Hoboken, NJ, 2001), Contemp. Math., Amer. Math. Soc. **298** (2002), 21–42.

[8] J.A. Buchman, *Introduction to cryptography*, Springer, 2004.

[9] C. Carstensen, B.Fine, and G.Rosenberger, *On asymptotic densities and generic properties in finitely generated groups*, preprint.

[10] D.Kahrobaei and M.Anshel, *Decision and search in non-abelian cramer shoup public key cryptosystem*, Journal of Groups, Complexity, Cryptography (2009), 1–8.

[11] David S. Dummit and Richard M. Foote, *Abstract algebra*, Wiley, 2003.

[12] Bettina Eick and Delaram Kahrobaei, *Polycyclic groups: A new platform for cryptology?*, math.GR/0411077 (2004), 1–7.

[13] B. Fine, M. Habeeb, D. Kahrobaei, and G.Rosenberger, *Survey and open problems in non-commutative cryptography*, JP Journal of Algebra, Number Theory and Applications **21** (2011), 1–40.

[14] M. Garey and J. Johnson, *Computers and intractability, a guide to np-completeness*, W.H. Freeman, 1979.

[15] S. Goldwasser and M. Bellare, *Lecture notes on cryptography*, 2001.

[16] W. A. De Graaf and W. Nickel, *Constructing faithful representations of finitely-generated torsion-free nilpotent groups*, J. Symbolic Computation **33** (2002), 31–41.

[17] D. Grigoriev and V. Shpilrain, *Unconditionally secure multi-party computation and secret sharing*, preprint.

[18] M. Habeeb and D. Kahrobaei, *On the dimension of matrix representations of finitely generated torsion free nilpotent groups*, Submitted (2011), 1–13.

[19] M. Habeeb, D. Kahrobaei, and V. Shpilrain, *A new public key using semi-direct products*, Proceedings of the Second International Conference on Symbolic Computation and Cryptography, Royal Holloway University of London, 137-142 (refereed) (2010), 1–5.

[20] ———, *Public key exchange using semidirect product of groups and metabelian groups*, Submitted (2011), 1–10.

[21] ———, *A secret sharing scheme based on non-commutative groups*, Preprint (2011).

[22] Philip Hall, *Nilpotent groups*, Notes of lectures given at the Canadian Mathematical Congress, University of Alberta (1957), 1–42.

[23] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, *Proactive secret sharing or: How to cope with perpetual leakage*, Advances in Cryptology-Crypto'95, Lecture Notes in Computer Science **963** (1995), 339–352.

[24] D. Holt, B. Eick, and E. O'Brien, *Handbook of computational group theory*, Discrete Mathematics and its Applications (Boca Raton), Chapman and Hall/CRC, Boca Raton, FL, 2005.

[25] Thomas W. Hungerford, *Algebra*, Springer, 1974.

[26] S. Jarecki, *Proactive secret sharing and public key cryptosystems*, Massachusetts Institute of Technology Master of Science Thesis (1995), 1–82.

[27] D.L. Johnson, *Presentations of groups*, London Mathematical Society Student Texts; 15, 2004.

[28] Delaram Kahrobaei and Bilal Khan, *A non-commutative generalization of the elgamal key exchange using polycyclic groups*, Proceeding of IEEE (2006), 1–5.

[29] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jae Woo Han, Ju-sung Kang, and Choonsik Park, *New public-key cryptosystem using braid groups*, Advances in cryptology—CRYPTO 2000 (Santa Barbara, CA), Lecture Notes in Comput. Sci., vol. 1880, Springer, Berlin, 2000, pp. 166–183.

[30] A.I. Kostrikin and I.R. Shafarevich, *Encyclopedia of mathematical sciences, volume 37, algebra iv: Infinite groups, linear groups*, Springer-Verlag, 1991.

[31] Francois Ledrappierl, *Some asymptotic properties of random walks on free groups*, Topics in probability and Lie groups: boundary theory, CRM Proc. Lecture notes, Amer. math. Soc. **IT-31** (2001), no. 28, 117–152.

[32] C.R. Leedham-Green and L.H. Soicher, *Symbolic collection using deep thought*, LMS J. Comput. Math. **1** (1998), 9–24.

[33] R. C. Lyndon and P. E. Schupp, *Combinatorial group theory*, Ergebnisse der Mathematic, band 89, Springer, 1977.

[34] Ayan Mahalanobis, *The diffie-hellman key exchange protocol and non-abelian nilpotent groups*, Israel Journal of Mathematics **165** (2008), 161–187.

[35] A. Myasnikov, V. Shpilrain, and A. Ushakov, *Group-based cryptography*, Advanced Courses in Mathematics. CRM Barcelona. Birkhauser Verlag, Basel, 2008.

[36] A.D. Myasnikov and A. Ushakov, *Length based attack and braid groups: Cryptanalysis of anshel-anshel-goldfeld key exchange protocol*, Lecture Notes in Computer Science, Springer, Public Key Cryptography - PKC 2007 (2007).

[37] W. Nickel, *Matrix representations for torsion-free nilpotent groups by deep thought*, Journal of Algebra **300** (2006), 376–383.

[38] D. Panagopoulos, *A secret sharing scheme using groups*, http://arxiv.org/PScache/arxiv/pdf/1009/1009.0026v1.pdf (2010).

[39] V. A. Roman'kov, *Equations in free metabelian groups*, SiberianMath. J. **20** (1979), 469–471.

[40] V. Shpilrain, *Assessing security of some group based cryptosystems*, Group theory, statistics, and cryptography, Contemp. Math., Amer. Math. Soc. **360** (2004), 167–177.

[41] ———, *Search and witness problems in group theory*, Groups, Complexity, and Cryptology **2** (2010), 231–246.

[42] V. Shpilrain and G. Zapata, *Using the subgroup membership search problem in public key cryptography*, Contemp. Math., Amer. Math. Soc. **418** (2006), 169–179.

[43] ———, *Using decision problems in public key cryptography*, Groups, Complexity, Cryptology **1** (2009), 199–206.

[44] Douglas R. Stinson, *Cryptography: Theory and practice*, Chapman and Hall, 2006.