City University of New York (CUNY)
CUNY Academic Works

Dissertations, Theses, and Capstone Projects Graduate Center

6-3-2014

Exploring platform (semi)groups for non-commutative key-exchange protocols

Ha Lam
Graduate Center, City University of New York

How does access to this work benefit you? Let us know!

Follow this and additional works at: http://academicworks.cuny.edu/gc_etds

Part of the Mathematics Commons

Recommended Citation

Lam, Ha, "Exploring platform (semi)groups for non-commutative key-exchange protocols" (2014). *CUNY Academic Works.*
http://academicworks.cuny.edu/gc_etds/241

This Dissertation is brought to you by CUNY Academic Works. It has been accepted for inclusion in All Graduate Works by Year: Dissertations, Theses, and Capstone Projects by an authorized administrator of CUNY Academic Works. For more information, please contact deposit@gc.cuny.edu.

EXPLORING PLATFORM (SEMI)GROUPS FOR NON-COMMUTATIVE KEY-EXCHANGE PROTOCOLS

by

Ha T. Lam

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

2014

©2014

Ha T. Lam

All Rights Reserved

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

Linda Keen

Date

Executive Officer

Delaram Kahrobaei

Vladimir Shpilrain

Melvyn Nathanson

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

EXPLORING PLATFORM (SEMI)GROUPS FOR NON-COMMUTATIVE KEY-EXCHANGE PROTOCOLS

By: HA T. LAM

Advisor: Delaram Kahrobaei

In this work, my advisor Delaram Kahrobaei, our collaborator David Garber, and I explore polycyclic groups generated from number fields as platform for the AAG key-exchange protocol. This is done by implementing four different variations of the length-based attack, one of the major attacks for AAG, and submitting polycyclic groups to all four variations with a variety of tests. We note that this is the first time all four variations of the length-based attack are compared side by side. We conclude that high Hirsch length polycyclic groups generated from number fields are suitable for the AAG key-exchange protocol. This work is also in the paper [11].

Delaram Kahrobaei and I also carry out a similar strategy with the Heisenberg groups, testing them as platform for AAG with the length-based attack. We conclude that the Heisenberg groups, with the right parameters are resistant against the length-based attack. This work can also be found in [29].

Another work in collaboration with Delaram Kahrobaei and Vladimir Shpilrain is to propose a new platform semigroup for the HKKS key-exchange protocol, that of matrices over a Galois field. We discuss the security of HKKS under this platform and advantages in computation cost. Our implementation of the HKKS key-exchange protocol with matrices over a Galois field yields fast run time. Our work is presented in [31].

Acknowledgements

I would like to express my deepest gratitude toward my thesis advisor, Delaram Kahrobaei, her continuous support and encouragement has helped me to be where I am today. Delaram has provided me with every opportunity a student could wish for, and has kept me focused and motivated throughout the many projects we did together. I am also very grateful to Vladimir Shpilrain whose many interesting ideas, commentaries and suggestions provide a solid foundation that I could build my work on. I would like to thank David Garber for the opportunity to collaborate with him. Enric Ventura provided a very hospitable environment during my trip to Barcelona and I am thankful for the opportunity to collaborate with him. Last but not least, I would like to thank Melvyn Nathanson for serving on my defense committee.

I acknowledge the support of the Office of Naval Research through a grant by Delaram Kahrobaei and Vladimir Shpilrain, PSC CUNY research grant through Delaram Kahrobaei and the American Association for the Advancement of Science research grant through Delaram Kahrobaei.

New York HTL
2014

To Gvidas Dambrauskas,

who anchors me to the world...

Preface

Almost eighty years ago, G. H. Hardy, in his famous "A Mathematician's Apology", wrote about the inapplicability of number theory to "ordinary human activities". He would be surprised by the rise of the field of cryptography, based on number theoretic problems, in the 1970s. A similar phenomenon is happening with group theory. From the introduction of the Anshel-Anshel-Goldfeld key-exchange protocol [1] to the work of Ko-Lee et. al. [32], non-commutative groups like the braid groups are increasingly being used for various cryptographic primitives.

Although relatively new, non-commutative cryptography has become an active field of research, attracting both mathematicians and computer scientists. In the last twenty years, numerous non-commutative cryptographic primitives have been proposed, and so have attacks against them. Of equal importance is the study of different platforms for these protocols, as the security of a cryptosystem relies not only on the security of the protocol but also on the underlying platform (semi)group. In this work, we explore different platform groups for the AAG key-exchange protocol using one of its major attacks, the length-based attack. We also propose a different platform semigroup for the HKKS key-exchange protocol which yields fast computation.

In **Chapter 1**, we give some background on public-key cryptography and a list of problems commonly used in non-commutative cryptography. We then describe the AAG key-exchange protocol and analyze its security. Finally, we describe some criteria for platform groups for AAG in particular, and for other non-commutative cryptographic primitives in general.

Chapter 2 is about polycyclic groups. We define polycyclic groups and talk about their presentation. We describe a method of generating polycyclic groups with number field. At the end, we give reasons why polycyclic groups generated this way is suitable as platform group.

In **Chapter 3**, we describe the idea behind the length-based attack, its history and related work. We then talk about the four variations of the length-based attack that we implemented in GAP and compare their advantages and disadvantages.

The previous three chapters build up to **Chapter 4**, which explores polycyclic groups generated from number fields as platform for the AAG key-exchange protocol by testing them under the length-based attack. The implementations details and results of our experiments are detailed here. We conclude that high Hirsch length polycyclic groups generated this way are suitable for AAG.

Chapter 5 attempts a similar process with the Heisenberg groups. We introduce the Heisenberg groups, put them through the same vigorous testing that we talked about in the previous chapter and concludes that Heisenberg groups, with the correct parameters, are resistant against the length-based attack.

Chapter 6 describes the HKKS key-exchange protocol and its inspiration, the classic Diffie-Hellman key-exchange protocol. Two platforms for HKKS, the multiplicative group of integers modulus a prime and matrices over group rings, are described.

The last chapter, **Chapter 7** introduces a new platform semigroup for the HKKS key-exchange protocol, that of matrices over a Galois field. The security

of the protocol under this platform is discussed. Finally, we talk about our implementation of the HKKS key-exchange protocol with matrices over a Galois field and give measurements showing that the protocol under this platform has fast run time.

Table of Contents

Chapter 1

Non-Commutative Cryptography and the AAG Key-Exchange Protocol

1.1 Background on Non-Commutative Cryptography

In anticipation of the discussion about the AAG protocol, here we give an overview of several concepts relating to public-key cryptography and non-commutative cryptography. For an introduction to public-key commutative cryptography, we refer to the book by Koblitz [33]; for more detailed discussion on non-commutative cryptography, see the book by Myasnikov, Shpilrain and Ushakov [38].

1.1.1 Public-key and symmetric-key cryptography

There are currently two main classes of cryptographic primitives, *public-key* (or *asymmetric*) and *symmetric-key.* In public-key algorithms, there are two separate keys, a public key that is published and a private key which is kept secret. In encryption, for example, the public key is used to encrypt the plaintext and the private key is used to decrypt the ciphertext. Knowledge of the public key does not imply knowledge of the private key in any efficient computation. In fact, the public key is usually generated from the private key using a *one-way trapdoor function*, a function that is easy to compute given any input but hard to invert given the image of a random input, unless some special information, called the trapdoor, is given. A typical example of public-key encryption that is used widely in electronic commerce is the RSA cryptosystem whose one-way trapdoor function is the product of two large primes p, q. Computing their product is easy, but factoring a large number into its prime factors, at this time, remains a hard problem.

The other type of cryptographic primitive, symmetric-key has been in used far longer than public-key, tracing back to Julius Caesar and possibly further. The main difference between public-key and symmetric-key primitives is that in symmetric-key ciphers, knowledge of encryption key is usually equivalent to, or exactly equal to, knowledge of decryption key, hence the name “symmetric”. This property requires the two parties to agree on a shared secret before communicating through an open channel. In the past, this exchange of secret keys usually relied on human factor and thus prone to information leakage. After the advent of public-key cryptography in the 1970s, it became much less troublesome to exchange secret key over an insecure channel. The Diffie-Hellman key-exchange protocol is a good example of how to exchange a shared secret, we will visit it more in depth in chapter 7.

As of current date, public-key algorithms are still more computationally costly than symmetric algorithshms given the same security parameter [42]. Therefore, some modern cryptosystem, OpenPGP for example, employs a hybrid system in which a session key is distributed using an asymmetric cipher prior to symmetric encryption.

1.1.2 Commonly used problems in non-commutative cryptography

All of our discussion of cryptographic primitives thus far has involved only finite abelian (or commutative) groups. There are several problems used in commutative cryptography, but the two main ones are factoring and discrete logarithm. With current technology, these two problems remain hard; however, there are efficient quantum algorithms to solve both of them. Thus, there is motivation to expand the search for new cryptographic primitives based on other branches of mathematics. A currently active branch of research is cryptography based on non-commutative groups.

The problems used in non-commutative cryptography are based on combinatorial group theory; however, our goal here is not to provide a detailed exposition, we refer instead to the book by Lyndon and Schupp [34].

In general, there are three kinds of problems in group theory: decision, witness, and search problems.

1. *Decision problems:* given an object $\mathcal{O}$ and a property $\mathcal{P}$, decide whether or not $\mathcal{O}$ has the property $\mathcal{P}$.

2. *Witness problems:* given an object $\mathcal{O}$ with property $\mathcal{P}$, find a proof (or a "witness") of the fact that $\mathcal{O}$ has property $\mathcal{P}$.

3. *Search problems:* given a property $\mathcal{P}$ and information that there are objects with property $\mathcal{P}$, find a proof of a particular instance of an object having property $\mathcal{P}$. Hence, search problems are a special case of witness problems.

Every decision problem has an accompanied witness problem, and most of them has a natural search version. This is illustrated in the following list of problems most commonly used in non-commutative cryptography. In all of these problems, G is a group with finite presentation $\langle X \mid R\rangle$ where X is the set of generators and R is the set of relators. The identity element of G is 1. We use the following notation: $y^x = x^{-1}yx$ where $x, y \in G$.

- **Word Problem:** the *word (decision) problem* (WP) is: given a group G and an element $g \in G$, find whether or not $g =_G 1$. The *word search problem* (WSP) is: given a $g =_G 1$, find a presentation of g as a product of conjugates of defining relators and their inverses. The word problem was first used in a public-key protocol by Wagner and Magyarik in 1985 [49], arguably the first non-commutative cryptosystem. It was neither secure nor practical, but nonetheless, pioneering.

- **Membership Problem:** the *subgroup membership (decision) problem* is: given a subgroup $H \leq G$ and an element $g \in G$, find whether or not $g \in H$. The *membership search problem* is: given a subgroup $H \leq G$ generated by $h_1, \ldots, h_k$ and an element $h \in H$, find an expression of h in terms of

$h_1, \ldots, h_k$. A cryptosystem invented in 2006 by Shpilrain and Zapata [47] makes use of the subgroup membership search problem directly. Aside from that, the security of the AAG key-exchange protocol also implicitly relies on it as detailed in section 1.2.

- **Conjugacy Problem:** the *conjugacy (decision) problem* (CP) is: given $g, h \in G$, find whether or not there is an $x \in G$ such that $g^x = h$. The *conjugacy search problem* (CSP) is: given $g, h \in G$ such that they are conjugates of each other, find an $x \in G$ such that $g^x = h$. There are other variants of the conjugacy search problem such as the *simultaneous conjugacy search problem* (otherwise known as the generalized conjugacy search problem in [13]): given $u_i, v_i \in G, 1 \leq i \leq n$ such that $u_i^x = v_i$ for some $x \in G$, find $x' \in G$ such that $u_i^{x'} = v_i$ for all $i = 1, \ldots, n$. The prime example of the conjugacy search problem being used in a cryptosystem is the AAG key-exchange protocol, we discuss AAG in details in section 1.2.

1.2 The Anshel-Anshel-Goldfeld Key Exchange Protocol

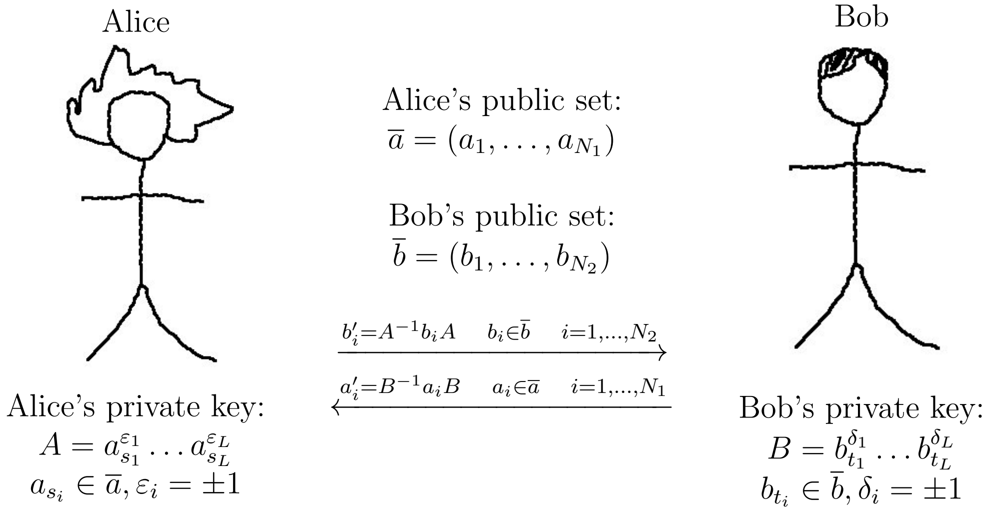
In 1999, Anshel, Anshel, and Goldfeld introduced a key-exchange protocol [1] called the Anshel-Anshel-Goldfeld key-exchanged protocol (or Arithmetica, or commutator key-exchange) that stands out from other protocols because it does not employ any commuting subgroups of the platform group nor require them to be commutative. In fact, the Anshel-Anshel-Goldfeld key-exchanged protocol

(AAG for short) can use any non-abelian group in which the word problem is efficiently solvable.

1.2.1 Description of AAG

We use as usual two entities, called Alice and Bob, to present the two parties which plan to communicate over an insecure channel.

The AAG Key-Exchange Protocol



Alice

Bob

Alice's public set:
$\overline{a} = (a_1, \ldots, a_{N_1})$

Bob's public set:
$\overline{b} = (b_1, \ldots, b_{N_2})$

$$\xrightarrow{b'_i = A^{-1} b_i A \quad b_i \in \overline{b} \quad i=1,\ldots,N_2}$$

$$\xleftarrow{a'_i = B^{-1} a_i B \quad a_i \in \overline{a} \quad i=1,\ldots,N_1}$$

Alice's private key:
$A = a_{s_1}^{\varepsilon_1} \ldots a_{s_L}^{\varepsilon_L}$
$a_{s_i} \in \overline{a}, \varepsilon_i = \pm 1$

Bob's private key:
$B = b_{t_1}^{\delta_1} \ldots b_{t_L}^{\delta_L}$
$b_{t_i} \in \overline{b}, \delta_i = \pm 1$

Let G be a general group with generators $g_1, \ldots, g_n$. First, Alice chooses, as her public set, $\overline{a} = (a_1, \ldots, a_{N_1})$ where $a_i \in G$ and Bob chooses, as his public set, $\overline{b} = (b_1, \ldots, b_{N_2})$ where $b_i \in G$. They both publish their sets. Alice then chooses her private key $A = a_{s_1}^{\varepsilon_1} \ldots a_{s_L}^{\varepsilon_L}$ where $a_{s_i} \in \overline{a}$ and $\varepsilon_i = \pm 1$. Bob also chooses his private key $B = b_{t_1}^{\delta_1} \ldots b_{t_L}^{\delta_L}$ where $b_{t_i} \in \overline{b}$ and $\delta_i = \pm 1$. Alice computes $b'_i = b_i^A$ for all $b_i \in \overline{b}$ and sends $\overline{b'} = (b'_1, \ldots, b'_{N_2})$ to Bob. Bob also computes $a'_i = a_i^B$ for all $a_i \in \overline{a}$ and sends $\overline{a'} = (a'_1, \ldots, a'_{N_1})$ to Alice. Now the shared secret key is $K = [A, B] = A^{-1}B^{-1}AB$. Alice can computes this key by

$$\begin{aligned} K_A &= A^{-1}a'^{\varepsilon_1}_{s_1} \cdots a'^{\varepsilon_L}_{s_L} = A^{-1}(B^{-1}a_{s_1}B)^{\varepsilon_1} \cdots (B^{-1}a_{s_L}B)^{\varepsilon_L} \\ &= A^{-1}B^{-1}a^{\varepsilon_1}_{s_1} \cdots a^{\varepsilon_L}_{s_L}B = A^{-1}B^{-1}AB = K \end{aligned}$$

Bob can likewise computes $K_B = B^{-1}b'^{\delta_1}_{t_1} \cdots b'^{\delta_L}_{t_L} = B^{-1}A^{-1}BA$, then the shared key is $K = K_B^{-1}$.

1.2.2 Security analysis of AAG

It may appear at first that solving the simultaneous conjugacy search problem in G for A in the equations $b'_i = b_i^A$ (or symmetrically, for B in the equations $a'_i = a_i^B$), is enough to obtain the secret key K. However, this is not the case, as Shpilrain and Ushakov noted in [46]; the adversary Eve would need to know A (or B) not just as a word in the generators of G but also as a word in $(a_1, \ldots, a_{N_1})$ (respectively $(b_1, \ldots, b_{N_2})$), otherwise, for example, she cannot compute A^B from $(a'_1, \ldots, a'_{N_1})$. Hence, after solving the simultaneous conjugacy search problem for G, Eve would need to solve the membership search problem: given A and $(a_1, \ldots, a_{N_1})$, find an expression of A as a word in $(a_1, \ldots, a_{N_1})$.

Another consideration is that even if Eve finds an $A' \in G$ such that $b_i^A = b_i^{A'}$ for all b_i, it does not imply that $A' = A$ in G. Indeed, there could be a $c_b \in G$ such that $A' = c_bA$ and $c_bb_i = b_ic_b$, then $b_i^{A'} = (c_bA)^{-1}b_i(c_bA) = A^{-1}c_b^{-1}b_ic_bA = A^{-1}c_b^{-1}c_bb_iA = b_i^A$. Since this is true for all b_i in Bob's public set $\overline{b} = (b_1, \ldots, b_{N_2})$, it's true for any element in the subgroup generated by $(b_1, \ldots, b_{N_2})$, in particular, it's true for B, Bob's private key, i.e., $B^{A'} = B^A$.

This can lead to Eve finding the wrong key if she does not perform a check, equivalent to solving the membership decision problem. Suppose Eve finds A' such that $A' = c_b A$ where $c_b b_i = b_i c_b$, and B' such that $B' = c_a B$ where $c_a a_i = a_i c_a$. Then c_a commutes with A and c_b commutes with B. From the information she has, Eve computes the key

$$\begin{aligned} K' &= (A')^{-1}(B')^{-1}A'B' = (c_bA)^{-1}(c_aB)^{-1}(c_bA)(c_aB) \\ &= A^{-1}c_b^{-1}B^{-1}c_a^{-1}c_bAc_aB = A^{-1}B^{-1}c_b^{-1}c_a^{-1}c_bc_aAB \qquad (1.1) \end{aligned}$$

This is the same as the key K that Alice and Bob compute if and only if c_a commutes with c_b. Eve does not know what K is, but there is a test that she could do, namely to check if $A' \in \langle a_1, \ldots, a_{N_1} \rangle$, or equivalently, if $B' \in \langle b_1, \ldots, b_{N_2} \rangle$. If one of these conditions is true, say $A' \in \langle a_i \rangle$, then c_b is also in $\langle a_i \rangle$ since $A' = c_b A$. We have c_a commutes with all the a_i, so it follows that c_a commutes with c_b. Hence, by equation 1.1, $K' = A^{-1}B^{-1}AB = K$.

These considerations show that if the adversary chooses to break AAG by directly solving the conjugacy search problem then she will have to either solve the membership search problem or the membership decision problem next. In certain group, the membership decision problem is not even solvable, for example, in $F_2 \times F_2$, Mihailova shows that this problem has no solution [36].

In summary, to break AAG, the eavesdropper needs to solve the *subgroup-restricted simultaneous conjugacy search problem*: given a subgroup $H < G$ and $u_i, v_i \in G, 1 \leq i \leq n$ such that $u_i^x = v_i$ for some $x \in H$, find $x' \in H$ such that $u_i^{x'} = v_i$ for all $i = 1, \ldots, n$.

Fortunately, there are heuristic attacks on the AAG protocol that avoid this problem because they are specifically designed to find solution of a system of equations from a set of given elements. One of these heuristic attacks, the length-based attack, is discussed in details in chapter 3.

Furthermore, the conjugacy search problem has been used for several other cryptographic protocols, such as the non-commutative Diffie-Hellman key exchange [32], the non-commutative El-Gamal key exchange [27], the non-abelian Cramer-Shoup key exchange [2] and the non-commutative digital signatures [28]. The length-based attack can be applied to all of these protocols, hence testing different groups against it and collecting data about parameters that make them resistant to LBA is important, not just for the implementation of AAG but also of other protocols.

1.3 Platform Groups for Non-Commutative Cryptographic Primitives

There are some requirements for a group G to be used as platform group for the AAG protocol; however, these requirements are not restricted to AAG but are useful ground-rules for other non-commutative cryptographic primitives as well. The paper [45] by Shpilrain first attempted to establish these requirements, they are later formalized in the book [38].

(P 0) The group G should be finitely presented. This allows encoding of the group in a computer system.

(P 1) The word problem in G can be solved effectively. This requirement is needed for fast computation of the common key. Since the presence of a normal form for G implies that the word problem is solvable, and normal form is also useful for another purpose (see next item), a group G with normal form is usually preferable.

(P 2) Elements in G after conjugation should be sufficiently disguised, i.e. it should be impossible to recover x from $x^{-1}ux$ just by inspection. This is usually achieved by requiring that G has normal form. The normal form of the element $x^{-1}ux$ typically has different presentation from the element itself, making recovering x hard. If G lacks a normal form, it should have at least one short relator to allow for cancellation in $x^{-1}ux$.

(P 3) The search conjugacy problem has no efficient solution in G. This is a hard requirement to prove, so in practice, we only expect the search conjugacy problem in G to be well-studied.

(P 4) The group G should have exponential or intermediate (subexponential but not polynomial) growth. Informally, the growth function in a group determines the number of elements of length n in the group. Requiring G to have fast growth function makes the key space large, which discourages brute force attack.

There are other considerations besides these core requirements, such as the ease of implementation of G, how well-studied G is to avoid attacks from different areas of study and how well-known G is for marketability. The paper [45], in particular, has an interesting discussion about marketability versus security, with the prime example of braid groups.

Chapter 2

Polycyclic Groups

2.1 Polycyclic Groups

The study of polycyclic groups started with Hirsch's series of papers [20, 19, 21, 22, 23], and has since been continued by R. Baer, A.I. Mal'cev, P. Hall and most recently, in the seminal book by Segal [43]. Polycyclic groups, despite their simple concept, have deep algebraic structure which make them attractive for applications. However, polycyclic groups have not been considered in conjunction with cryptography until the paper by Eick and Kahrobaei [7]. In this chapter, we review the basic properties of polycyclic groups, we investigate a method of generating them from number fields and finally, we consider their suitability as platform group for the AAG key-exchange protocol.

2.1.1 Polycyclic groups: definitions and examples

The concept of polycyclic groups arises from finite extensions of cyclic groups, one of the most fundamental of all group structures, but the resulting structure is much more complex. Here, we give a brief review of some of the definitions and results involving polycyclic groups. We refer to [25, 9, 43] for more details.

Definition 2.1.1. G is a *polycyclic group* if it has a subnormal series with non-trivial cyclic factors, i.e.,

$$G = G_1 \trianglerighteq G_2 \trianglerighteq \ldots \trianglerighteq G_{n+1} = 1$$

where G_i/G_{i+1} is cyclic for $1 \leq i \leq n$.

The subnormal series for G is also called a *polycyclic series*, but it is not unique for the group G, as we will see in example 2.1.1.

For each polycyclic series, since G_i/G_{i+1} is cyclic, there exists a $g_i \in G$ for each factor group such that $G_i = \langle g_i, G_{i+1} \rangle$. On the other hand, consider a sequence $X = (g_1, \ldots, g_n), x_i \in G$. We can define subgroups $G_i = \langle g_i, \ldots, g_n$ for $1 \leq 1 \leq n$, so X defines the subgroup series $G_1 \geq \ldots \geq G_n \geq G_{n+1} = 1$ of G. We come to the following definition:

Definition 2.1.2. A sequence $X = (g_1, \ldots, g_n), x_i \in G$ is a *polycyclic (generating) sequence* for G if the subgroup series determined by X forms a polycyclic series for G.

It is clear that a group is polycyclic if and only if it has a polycyclic sequence. We note, however, that each polycyclic series can have several different polycyclic sequences as we will see in example 2.1.1.

Fix a polycyclic sequence X for G, if we let the orders of the the factors groups be $r_i = [G_i : G_{i+1}], 1 \leq i \leq n$, we have the following definitions:

Definition 2.1.3. $R(X) = (r_1, \ldots, r_n)$ is the *sequence of relative orders*, and $I(X) = \{i \in \{1, \ldots, n\} \mid r_i \text{ is finite}\}$ is the *finite index set* for the polycyclic sequence in consideration.

Example 2.1.1. *[25] Consider the dihedral group of order 8, $D_8 \cong \langle (1,2,3,4),(1,3)\rangle$. It has polycyclic series:*

1. *$D_8 \cong G_1 \trianglerighteq G_2 \trianglerighteq G_3 = 1$ where $G_2 \cong C_4$, the cyclic group of order 4. This series has several different polycyclic sequences, two of which are:*
   (a) *Polycyclic sequence $X = ((1\,3),(1\,2\,3\,4))$ with relative orders $R(X) = (2,4)$ and finite index set $I(X) = \{1,2\}$, and*
   (b) *Polycyclic sequence $Y = ((2\,4),(1\,4\,3\,2))$ with relative orders $R(Y) = R(X) = (2,4)$, and finite index set $I(Y) = I(X) = \{1,2\}$.*
2. *Another polycyclic series is $D_8 = G_1 \trianglerighteq G_2 \trianglerighteq G_3 \trianglerighteq G_4 = 1$ where $G_2 \cong V_4$, the Klein four-group, and $G_3 \cong C_2$. This series also has, for example, two different polycyclic sequences:*
   (a) *Polycyclic sequence $X = ((2\,4),(1\,2)(3\,4),(1\,3)(2\,4))$ with relative orders $R(X) = (2,2,2)$ and finite index set $I(X) = \{1,2,3\}$, and*
   (b) *Polycyclic sequence $Y = ((1\,2\,3\,4),(1\,2)(3\,4),(1\,3)(2\,4))$ with relative orders $R(Y) = R(X) = (2,2,2)$, and finite index set $I(Y) = I(X) = \{1,2,3\}$.*

For the finite polycyclic groups, the order of the group is the product of relative orders. In the previous example, $|D_8| = 2 \cdot 4 = 2 \cdot 2 \cdot 2 = 8$ is the product of relative orders for all cases. Otherwise, G is an infinite polycyclic groups if and only if at least one of the relative orders is infinite; in that case, $|I(X)| < n$.

As we can see, both the relative orders $R(X)$ and the finite index set $I(X)$ is invariant of the polycyclic sequence, but dependent on the polycyclic series. On the other hand, a very useful notion relating to polycyclic groups is the Hirsch length:

Definition 2.1.4. The *Hirsch length* of a polycyclic group G, $h(G)$, is the number of infinite entries in the relative orders.

We can easily see that $h(G)$ is invariant regarding the polycyclic series chosen because, given any two polycyclic series, we can use the Schreier refinement theorem to make sure that they have equivalent refinements.

Example 2.1.2. *[9] An example of an infinite polycyclic group is the infinite dihedral group G generated by*

$$a = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad \textit{and} \quad b = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

G has polycyclic series representing by the following sequences:

1. *$X = (a, b)$ with relative orders $(2, \infty)$ and finite index set $I = \{1\}$, and*
2. *Other polycyclic sequences are $X = (a, b, b^m)$ for $m \in \mathbb{N}$ with relative orders $(2, m, \infty)$ and finite index set $I = \{1, 2\}$.*

Here, the Hirsch length, $h(G)$, is 1, regardless of which polycyclic series is used, whereas the finite entries in the relative orders are almost arbitrary.

Another useful feature of polycyclic sequences is that we can write any element in G *uniquely* in terms of the elements of a polycyclic sequence, i.e., given

$g \in G$ and a polycyclic sequence $X = (g_1, \ldots, g_n)$ of G with relative orders $(r_1, \ldots, r_n)$ and finite index set I, we can uniquely write g as $g = g_1^{e_1} \cdots g_n^{e_n}$ where $e_1, \ldots, e_n \in \mathbb{Z}$ and $0 \leq e_i < r_i$ for $i \in I$.

Definition 2.1.5. The *normal form* of an element $g \in G$ is the unique expression $g = g_1^{e_1} \cdots g_n^{e_n}$ above and the corresponding vector $(e_1, \ldots, e_n)$ is the *exponent vector* of g.

We say a group G has normal form if every element of G has normal form. Hence, polycyclic groups has normal form. Furthermore, there are algorithms, such as the collection algorithm, that allow normal form of an element of a polycyclic group to be determined effectively for most common representations [9]. This makes the word problem solvable efficiently in polycyclic groups.

2.1.2 Polycyclic presentation

There are some well-known presentations for polycyclic groups: polycyclic presentation and matrix presentation. Out of the two, polycyclic presentation allows us to easily see a polycyclic sequence and, if the presentation is consistent, let us derive its sequence of relative orders. We refer to the book by D. Holt [25] and B. Eick's habilitation [9] for more details.

Definition 2.1.6. A *polycyclic presentation* is a finite presentation with generators $g_1, \ldots, g_n$ and a sequence $S = (s_1, \ldots, s_n)$ that satisfies:

$$\begin{aligned} g_i^{s_i} &= w_i(g_{i+1}, \ldots, g_n) \text{ where } 1 \leq i \leq n \text{ and } s_i < \infty, \\ g_i^{g_j} &= u_{ij}(g_{i+1}, \ldots, g_n) \text{ where } 1 \leq j < i \leq n \text{ and,} \\ g_i^{g_j^{-1}} &= v_{ij}(g_{i+1}, \ldots, g_n) \text{ where } 1 \leq j < i \leq n. \end{aligned}$$

The first type of relations are called *power relations*, the second and third types are called *conjugate relations.* The sequence $S = (s_1, \dots, s_n)$ is called the sequence of *power exponents.* Note that other relations of the form $g_i^{g_j} = g_i$ are called *trivial polycyclic relations* and are usually omitted from a polycyclic presentation.

By investigating normal forms of powers and conjugates of elements in a polycyclic sequence, we can see that they follow the format of polycyclic relations. The following result is easy to see:

Theorem 2.1.3. *[25] Every polycyclic sequence determines a unique polycyclic presentation. Thus, every polycyclic group can be defined by a polycyclic presentation.*

We note that the sequence of power exponents is not exactly the same as the relative orders, however, they are related to each other in the following theorem:

Theorem 2.1.4. *[25] Let $\langle g_1, \dots, g_n \mid R\rangle$ be a polycyclic presentation of a group G with power exponents $S = (s_1, \dots, s_n)$. Then G is polycyclic and $X = (g_1, \dots, g_n)$ is a polycyclic sequence for G with relative orders $R(X) = (r_1, \dots, r_n)$ satisfying $r_i \leq s_i$ for $1 \leq i \leq n$.*

Among other things, this allows an “almost normal form” for elements of the group G, giving rise to the following concept:

Definition 2.1.7. Let G be a polycyclic group defined by a polycyclic presentation with generators $g_1, \dots, g_n$ and power exponents $S = (s_1, \dots, s_n)$, then every element $g \in G$ can be represented by a *collected word* of the form $g = g_1^{e_1} \cdots g_n^{e_n}$ where $e_1, \dots, e_n \in \mathbb{Z}$ and $0 \leq e_i < s_i$ if $s_i < \infty$.

Because the exponents of a collected word is bounded by the power exponents, not the relative orders, we do not have the (unique) normal form of the element. However, the normal form of g is a collected word.

The *collection algorithm* can be used to find a collected word of an element. Suppose G is a group with a polycyclic presentation and element $g \in G$ is given as a word w in the generators. The collection algorithm essentially goes through subwords of w and modify them using the relations in the polycyclic presentation of G until the end result is a collected word. It can be shown that the algorithm terminates. For more information on the collection algorithm, see [25, 9].

So far, we have looked at the general case where the power exponents and the relative orders are different. It is useful to consider the case where they are the same because this is the presentation that is most used in algorithms for polycyclic groups. We arrive at the following definition:

Definition 2.1.8. A polycyclic presentation where the power exponents and the relative orders are the same is called *consistent* (or *confluent*).

With a similar proof to the one of theorem 2.1.3, we have the following theorem:

Theorem 2.1.5. *[25] Every polycyclic sequence determines a consistent polycyclic presentation. Thus, every polycyclic group can be defined by a consistent polycyclic presentation.*

As an example of the usefulness of a consistent polycyclic presentation, we note that with a consistent polycyclic presentation, the collection algorithm produces normal forms of elements instead of just collected words. Thus, the word problem can be solved.

2.1.3 Generating polycyclic groups with number field

There are many types of polycyclic groups and many ways of generating them. In this section, we talk about a particular method, generating polycyclic groups with number field. The main idea here is to construct polycyclic groups as semidirect products of the maximal order and the unit group of a number field. We use this method because it gives us polycyclic group of high Hirsch length that has exponential growth. This construction follows [25].

First, we give a brief reminder of number fields and some concepts relating to it. For more details, see [48].

Definition 2.1.9. $\mathcal{F}$ is an (algebraic) number field if it is a finite field extension of $\mathbb{Q}$.

There are several ways of generating number field, for our purpose, we consider an irreducible polynomial $f(x) \in \mathbb{Z}[x]$, then f defines a field extension $\mathcal{F}$ over $\mathbb{Q}$.

Definition 2.1.10. The *maximal order* or the *ring of integers* $\mathcal{O}_{\mathcal{F}}$ of the number field $\mathcal{F}$ is the set of algebraic integers in $\mathcal{F}$, i.e., $\mathcal{O}_{\mathcal{F}} = \{a \in \mathcal{F} \mid \text{ there exists a monic polynomial } f_a(x) \in \mathbb{Z}[x] \text{ such that } f_a(a) = 0\}$. The *unit group* of $\mathcal{F}$ is $\mathcal{U}_{\mathcal{F}} = \{a \in \mathcal{O}_{\mathcal{F}} \mid a \neq 0 \text{ and } a^{-1} \in \mathcal{O}_{\mathcal{F}}\}$.

To generate polycyclic group from the maximal order and unit group of a number field $\mathcal{F}$ where $[\mathcal{F} : \mathbb{Q}] = n$, we recall two results.

Theorem 2.1.6. *[48] The maximal order $\mathcal{O}_{\mathcal{F}}$ forms a ring whose additive group is isomorphic to $\mathbb{Z}^n$.*

The proof of this theorem involves finding an integral basis for $\mathcal{F}$ as a vector space over $\mathbb{Q}$, which is also a basis for $\mathcal{O}_{\mathcal{F}}$. We do this by choosing a $\mathbb{Q}$-basis of $\mathcal{F}$ of minimal discriminant. For more details of the proof, see [48].

The second theorem is due to Dirichlet:

Theorem 2.1.7. *[48] (Dirichlet) Let $n = s+2t$ where s and $2t$ are the numbers of real and complex field monomorphisms $\mathcal{F} \to \mathbb{C}$, then the unit group $\mathcal{U}_{\mathcal{F}}$ is a finitely generated abelian group of the form $\mathcal{U}_{\mathcal{F}} \cong C_\infty^{s+t-1} \times C_m$ where $m \in 2\mathbb{N}$ and $C_\infty \cong \mathbb{Z}$ and C_m is the cyclic group of order m.*

Here, we mainly make use of the fact that the unit group is a finitely generated abelian group, and hence also polycyclic.

Let G be a group and $N \trianglelefteq G$, it is easy to see that if N and G/N are both polycyclic then the group G is also polycyclic. We show this by putting together the polycyclic series of N and the series induced from the polycyclic series of G/N. Since the above results guarantee that the maximal order is a polycyclic group and the unit group, which is isomorphic to $G/\mathcal{O}_{\mathcal{F}}$, is also polycyclic, the group $G = \mathcal{O}_{\mathcal{F}} \rtimes \mathcal{U}_{\mathcal{F}}$ is polycyclic. The action here is multiplication from the right of $\mathcal{U}_{\mathcal{F}}$ on $\mathcal{O}_{\mathcal{F}}$.

Since the Hirsch length of a polycyclic group G is $h(G) = h(N) + h(G/N)$ if $N \trianglelefteq G$; in our case, $h(G) = h(\mathcal{O}_{\mathcal{F}}) + h(\mathcal{U}_{\mathcal{F}})$ where $h(\mathcal{O}_{\mathcal{F}})$ is n, the degree of the generating polynomial f. Hence, to achieve a polycyclic group of high Hirsch length, all we need to do is find an irreducible polynomial of high enough order and the polycyclic group generated by the above method will have Hirsch length bigger than the degree of the polynomial.

2.2 Polycyclic groups as platform group

Even though polycyclic groups have been studied since the 1930s and is an active area of research, it was not until 2004 that Eick and Kahrobaei suggested to use them in cryptography [7], in particular, as a platform for the AAG key-exchange protocol. In chapter 1, we listed the requirements for a group to be used as platform group for non-commutative cryptographic primitives, in particular, the AAG key-exchange protocol. In this section, we consider polycyclic groups in relation to these requirements.

(PC 0) Polycyclic groups are finitely presented. The presentation can be easily coded into a computer system, for example, the polycyclic package by Eick and Nickel [8] for the GAP system [15], or the MAGMA system [5].

(PC 1) The word problem can be solved effectively in polycyclic groups with consistent polycyclic presentations via the collection algorithm [9, 7].

(PC 2) A result of Eick [9] shows that polycyclic groups with consistent polycyclic presentations have normal form; in fact, the collection algorithm can be used to produce normal form of any element, making the word problem trivial.

(PC 3) The search conjugacy problem in certain polycyclic groups have been studied by Eick and Kahrobaei and found to be inefficient [7]. We talk more about their result at the end of this list.

(PC 4) Polycyclic groups which are not virtually nilpotent have exponential growth rate (Wolf [50] and Milnor [37]). As mentioned above, polycyclic groups generated with number fields have this property.

As mentioned above, to study the search conjugacy problem in polycyclic groups, Eick and Kahrobaei [7] conducted the following experiment: let $K = \mathbb{Q}[x]/(f_w)$ be an algebraic number field for a cyclotomic polynomial f_w, where w is a primitive r-th root of unity, and let $G(w) = O \rtimes U$ where O is the maximal order and U the unit group of K, r the order of w and $h(G(w))$ the Hirsch length. The average time used for 100 applications of the collection algorithm on random words and the average time used for 100 applications of the conjugacy algorithm on random conjugates are:

r	h(G(w))	coll	conj
3	2	0.00 sec	9.96 sec
4	2	0.00 sec	9.37 sec
7	6	0.01 sec	10.16 sec
11	14	0.05 sec	> 100 hrs

We can see that the collection algorithm works very fast even for polycyclic groups of high Hirsch length, which enables the word problem to be solved efficiently; but the solution to conjugacy problem is not efficient for groups of Hirsch length 14 or more. This study gives hint that the conjugacy search problem for polycyclic groups cannot be solved efficiently.

Together with the other considerations (CP 0-4), polycyclic groups look like a good candidate for a platform group of the AAG key-exchange protocol in particular, and a platform group for other non-commutative cryptographic primitive in general. Taking this as an inspiration, we investigate the usefulness of polycyclic groups as platform for AAG, particularly under the length-based attack, one of the major attacks against AAG.

Chapter 3

The Length-Based Attack

3.1 History and Related Work

The Anshel-Anshel-Goldfeld key-exchange protocol has been studied under various attacks, of which the length-based attack is one of the major methods. The length-based attack originated in 2002 with the paper by Hughes and Tannenbaum [26]. In this paper, they gave an example of the method applied to the braid group. They also noted that the method is based on the assumption that elements in the group used in the AAG protocol has a canonical presentation whose length can be computed rapidly. However, there were doubts regarding the method [14] because no realization of the idea was given; in particular, no definition of an effective length function was given, making the success probability of the approach hard to calculate.

A few years after that, Garber, Kaplan, Teicher, Tsaban and Vishne set out to settle this problem. In an earlier paper [12], they developed a technique which extended the standard length-based attack by using memory with the intention of giving probabilistic solutions to equations in braid group. In particular, given a system of equations in a finitely generated subgroup of the braid group, they were able to find a solution to the system with high probability. This system of equations is exactly the situation that we have with AAG, so it's not surprising that a year after that, they expanded this idea into another paper [13], in which they introduced several effective realizations of the approach. They defined a new length function on the braid group, and compared it with the Garside normal form. Finally, they gave experimental results suggesting that it is infeasible to use this method to solve the Generalized Conjugacy Search Problem with parameters as in existing protocols.

More recently, Myasnikov and Ushakov developed another variation of the length-based attack for braid group [39]. They suggested a heuristic algorithm for the approximation of geodesic length of braids which they used as length function. They also analyzed the reasons behind failure in some of the implementations of this variation, such as the commutator-type peak, which are elements whose length after conjugations are shorter than length before conjugations. With this analysis, they designed an implementation which is capable of breaking AAG with high rate of success.

Expanding on this work, Myasnikov, Shpilrain and Ushakov proved that the same results of the feasibility of length-based attack could be observed in much larger classes of groups [38]. In particular, it would work for free groups, pure braid groups, locally commutative non-abelian groups, etc.

Of a different flavor, Shpilrain and Ushakov suggested Thompson's groups for a cryptosystem based on the decomposition problem [44]. The length-based attack has also been deployed against this system in a paper by Ruinskiy, Shamir and Tsaban [41].

3.2 Description of the Length-Based Attack

The length-based attack is a probabilistic attack against the AAG, with the goal of finding Alice's (or Bob's) private key. It is based on the idea that a conjugation of the right element will decrease the length of the captured package.

Recall that Eve, the eaves-dropper, observing the communications between Alice and Bob, will see $\overline{a} = (a_1, \ldots, a_{N_1})$ which is Alice's public set, $\overline{b} = (b_1, \ldots, b_{N_2})$ which is Bob's public set, $\overline{b'} = (b'_1, \ldots, b'_{N_2})$ such that $b'_i = b_i^A$ for $i = 1, \ldots, N_2$ which is Alice's private key in disguise, and $\overline{a'} = (a'_1, \ldots, a'_{N_1})$ such that $a'_i = a_i^B$ for $i = 1, \ldots, N_1$ which is Bob's private key in disguise. For the purpose of the length-based attack, suppose that Eve wants to guess $A' \in \langle a_1, \ldots, a_{N_1} \rangle$ such that $b'_i = b_i^{A'}$ or symmetrically, $B' \in \langle b_1, \ldots, b_{N_2} \rangle$ such that $a'_i = a_i^{B'}$, then she can compute the shared key $B^{-1}A^{-1}BA$.

For a bit of notation, if $\overline{c} = (c_1, \ldots, c_k)$, then define its *total length* $|\overline{c}|$ to be $\sum_{i=1}^{k} |c_i|$.

For the purpose of breaking AAG, suppose we want to find the conjugate A'. We capture the package $\overline{b'} = (b'_1, \ldots, b'_{N_2})$ where $b'_i = A^{-1}b_iA$. If we conjugate $\overline{b'}$ with elements from the group $\langle a_1, \ldots, a_{N_1} \rangle$ and the resulting tuple has decreased total length, then we know we have found a conjugating factor, i.e., a factor of the conjugate A'. The process of conjugation is then repeated with the decreased length tuple until another conjugating factor is found. The process ends when the conjugated captured package is the same as $\overline{b} = (b_1, \dots, b_{N_2})$, which is public knowledge. Then the conjugate A' can be recovered by taking a product of the sequence of conjugating factors in reverse. The entire process can be visualized as going up the following tower from the bottom

$$\begin{tikzcd}
b_i \arrow[d] \\
a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1} \arrow[d] \\
a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \arrow[d] \\
\vdots \arrow[d] \\
a_{s_L}^{-\varepsilon_L} \dots a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} b_i a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \dots a_{s_L}^{\varepsilon_L}
\end{tikzcd}$$

3.3 Variations of the Length-Based Attack

In [13, 12, 39, 41], several variations of LBA are given. Here we give four variants of LBA that we implemented for AAG based on polycyclic group. In all these algorithms, the following input and output are expected:

- Input: $\overline{a} = (a_1, \dots, a_{N_1})$, $\overline{b} = (b_1, \dots, b_{N_2})$ and $\overline{b'} = (b'_1, \dots, b'_{N_2})$ such that $b'_i = b_i^A$ for $i = 1, \dots, N_2$.
- Output: An element $A' \in \langle a_1, \dots, a_{N_1} \rangle$ such that $b'_i = b_i^{A'}$ for $i = 1, \dots, N_2$ or FAIL if the algorithm cannot find such A'.

3.3.1 LBA with backtracking

The most straight-forward variation of the length-based attack, Algorithm 1 conjugates $\overline{b'}$ directly with $a_i^{\pm 1} \in \{a_1, \ldots, a_{N_1}\}$. This is termed "LBA with backtracking" by Myasnikov and Ushakov [39]. First, we initialize the set of current conjugated tuples, S, with $(\overline{b'}, id_G)$. The first coordinate of the pair acts as the current tuple to be conjugated, the second coordinate keeps track of what element has been used to in the conjugation. In each round, we take out an element of S whose first coordinate has the smallest total length, and conjugate it with all the candidates $a_i^{\pm 1} \in \{a_1, \ldots, a_{N_1}\}$. For each of the newly conjugated tuple, we check if the total length of after conjugation is smaller than the length of the tuple before conjugation. If it is, then that candidate $a_i^{\varepsilon_i}$ is a conjugating factor, i.e., a factor of A'. We update the second coordinate of that tuple to reflect the new factor by multiplying the current factors so far with the new candidate. We then add the this conjugated tuple back into the set of current conjugated tuples, S. We continue this until the conjugated tuple that we get back is equal to $\overline{b}$. Then we stop and reverse the sequence of conjugation factors for the conjugate A'. If we take out every tuple of S, i.e., $S = \emptyset$, and conjugate each of them with all potential candidates but still do not reach the original $\overline{b}$, then we stop and return FAIL.

We note that this algorithm fails in cases where the total length of a conjugated tuple is actually shorter than its length before conjugation, i.e., there exists a peak. In that case, the tuple is never added back into S, hence branches of possibilities are never investigated.

Algorithm 1 LBA with backtracking

```
Initialize $S = \{(\overline{b'}, id_G)\}$.
while $S \neq \emptyset$ do
    Choose $(\overline{c}, x) \in S$ such that $|\overline{c}|$ is minimal. Remove $(\overline{c}, x)$
    for $i = 1, \ldots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\delta_{i,\varepsilon} = |\overline{c}| - |\overline{c}^{a_i^\varepsilon}|$
        if $\overline{c}^{a_i^\varepsilon} = \overline{b}$ then output inverse of $xa_i^\varepsilon$ and stop
        if $\delta_{i,\varepsilon} > 0$ {length has been decreased} then
            Add $(\overline{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ to $S$
        end if
    end for
end while
Otherwise, output FAIL {no more element to conjugate}
```

3.3.2 LBA with a dynamic set

Through analysis, Myasnikov and Ushakov concluded that the different types of peaks make LBA unsuccessful [39]. To overcome this, they suggested a new version of the algorithm, which they termed “LBA with dynamic set”. Here, depending on whether a candidate causes length reduction, either only the conjugates and products involving the candidate are added to the dynamic set, or, in the unlucky case of no length reduction, all conjugates and two generators products are added. Their experimental results suggest that this algorithm works especially well in the case of keys constructed from long generators, but no less successful than the naive algorithm in other cases. Algorithm 2 is a modified version of their algorithm, which we implemented to attack AAG based on polycyclic group.

This is similar to Algorithm 1, but we use $a_i^{\pm 1} \in \{a_1, \ldots, a_{N_1}\}$ as candidates only in the first round. After the first round, the length of the conjugated tuple might decrease or not. If it does, say by candidate $a_m^{\varepsilon_m}$, we focus on this element by adding all the conjugates and two-generator products involving it to the candidates set, i.e., $\overline{a}_{ext} = \overline{a} \cup \{a_j a_m^{\varepsilon_m} a_j^{-1}, a_m^{\varepsilon_m} a_j, a_j a_m^{\varepsilon_m}, (a_m^{\varepsilon_m})^2 \mid a_j \in \overline{a}^{\pm 1}, m \neq j\}$ where a_m is such that the total length of the tuple conjugated by it decreased the most after the first round, $\delta_m = \max\{\delta_{i,\varepsilon} \mid i = 1, \ldots, N_1\}$.

If, after the first round, none of the conjugated tuple has decreased length, we extend the candidates set by adding all conjugates and all two-generators products, $\overline{a}_{ext} = \overline{a} \cup \{a_i a_j a_i^{-1}, a_i a_j, a_i^2 \mid a_i, a_j \in \overline{a}^{\pm 1}, i \neq j\}$. It is this last measure that helps against cases where all conjugated tuples have total length shorter than their length before conjugations, and it is the reason why Algorithm 2 is more successful than Algorithm 1. Even then, this is not an exhaustive search because we are only considering conjugates and two-generator products, so the algorithm may still fail. However, there is a trade off between the depth-first idea of adding candidates of more complex structure (say, four-generator products) to test against, which increase computation time, and the breadth-first idea of moving on to the next tuple and hope for a lucky hit.

3.3.3 LBA with memory 1

Based on [12], we came up with another version of LBA that is based on a fixed-size memory M allocated to the algorithm. Here, S holds M tuples every round and is sorted by the first coordinate (length of conjugated element) of each tuple. In every round, the smallest element of S is removed and conjugated by all the generators and their inverses. Then the conjugated tuples are added back into S depending on whether there is still a free place in S, i.e., whether $|S| \leq M$, as opposed to whether their length has been reduced like in Algorithm 1 and 2.

Algorithm 2 LBA with dynamic set

```
Initialize $S = \{(\overline{b'}, id_G)\}$.
while $S \neq \emptyset$ do
    Choose $(\overline{c}, x) \in S$ such that $|\overline{c}|$ is minimal. Remove $(\overline{c}, x)$
    for $i = 1, \ldots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\delta_{i,\varepsilon} = |\overline{c}| - |\overline{c}^{a_i^\varepsilon}|$
    end for
    if $\delta_{i,\varepsilon} \leq 0$ for all $i$ then
        Define $\overline{a}_{ext} = \overline{a} \cup \{x_i x_j x_i^{-1}, x_i x_j, x_i^2 \mid x_i, x_j \in \overline{a}^{\pm 1}, i \neq j\}$
    else
        Define $\overline{a}_{ext} = \overline{a} \cup \{x_j x_m x_j^{-1}, x_m x_j, x_j x_m, x_m^2 \mid x_j \in \overline{a}^{\pm 1}, m \neq j\}$ where
        $x_m$ s.t. $\delta_m = \max\{\delta_{i,\varepsilon} \mid i = 1, \ldots, N_1\}$
    end if
    for all $w \in \overline{a}_{ext}$ do
        Compute $\delta_w = |\overline{c}| - |\overline{c}^w|$
    end for
    if $\overline{c}^w = \overline{b}$ then output inverse of $xw$ and stop
    if $\delta_w > 0$ {length has been decreased} then
        Add $(\overline{c}^w, xw)$ to $S$
    end if
end while
Otherwise, output FAIL {no more element to conjugate}
```

If there is no more places in S, and if the conjugated tuple has total length smaller than the length of the largest element in S, then swap them instead, and then re-sort S. Note that since S is kept sorted always, any operation to find "smallest element" is in constant time. Because S is never empty as in Algorithm 1 and 2, we use a time-out that can be changed as the stopping condition.

The problem with this approach is that the same conjugated tuple might be removed and conjugated repeatedly if conjugation does not change its total length. Because its length stay the same, it gets added back to the beginning of the S queue, from then it is taken out and re-added ad infinitum.

Algorithm 3 LBA with Memory 1

```
Initialize $S = \{(|\overline{b'}|, \overline{b'}, id_G)\}$.
while not time out do
    Choose $(|\overline{c}|, \overline{c}, x) \in S$ such that $|\overline{c}|$ is minimal. Remove $(|\overline{c}|, \overline{c}, x)$
    for $i = 1, \ldots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\overline{c}^{a_i^\varepsilon}$
        if $\overline{c}^{a_i^\varepsilon} = \overline{b}$ then output inverse of $xa_i^\varepsilon$ and stop
        if Size$(S) < M$ then
            Add $(|\overline{c}^{a_i^\varepsilon}|, \overline{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ to $S$ and sort $S$ by first element of every tuple
        else
            if $|\overline{c}^{a_i^\varepsilon}|$ smaller than first element of all tuples in S then swap them
        end if
    end for
end while
Otherwise, output FAIL {no more element to conjugate}
```

3.3.4 LBA with memory 2

A different algorithm, truer to the spirit of [12], is implemented here. In this algorithm, S holds M tuples every round similar to Algorithm 3. The difference is that in every round, all elements of S are conjugated, but only the M smallest conjugated tuples (by total length) are added back into S. Because we are adding back M smallest tuples and not just one single element, we avoid the pitfall of Algorithm 3. Moreover, this method of saving several tuples, not just the ones whose length decreased after conjugation, keeps us away from problems generated by peaks as in Algorithm 1. Here, for the stopping condition, we use a time-out that is defined by the user as with the previous variant.

Algorithm 4 LBA with Memory 2

```
Initialize $S = \{(|\overline{b'}|, \overline{b'}, id_G)\}$.
while not time out do
    for $(|\overline{c}|, \overline{c}, x) \in S$ do
        Remove $(|\overline{c}|, \overline{c}, x)$ from $S$
        Compute $\overline{c}^{a_i^\varepsilon}$ for all $i \in \{1 \dots N_1\}$ and $\varepsilon = \pm 1$
        if $\overline{c}^{a_i^\varepsilon} = \overline{b}$ then output inverse of $xa_i^\varepsilon$ and stop
        Save $(|\overline{c}^{a_i^\varepsilon}|, \overline{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ in $S'$
    end for
    After finished all conjugations, sort $S'$ by the first element of every tuple
    Copy the smallest $M$ elements into $S$ and delete the rest of $S'$
end while
Otherwise, output FAIL
```

Chapter 4

Exploring Polycyclic Groups as Platform for the AAG Key-Exchange Protocol

In chapter 1, we introduced the Anshel-Anshel-Goldfeld key-exchange protocol and talked about platform groups for non-commutative cryptographic primitives. In chapter 2, we talked about polycyclic groups and why they are a good choice for platform group, in particularly for the AAG protocol. We recall that one of the criteria for being a platform group is that the search conjugacy problem has no efficient solution in the group and we pointed out that this is a hard requirement to prove.

It is natural then, that in order to study the suitability of polycyclic groups as platform group, particularly for the AAG protocol, that we deploy one of the best known attacks against AAG, the length-based attack (LBA), which we talked about in-depth in chapter 3. To that end, we implemented the four variants of LBA presented in chapter 3, section 3.3 and ran experiments on polycyclic groups generated with number fields with different Hirsch lengths.

4.1 Length Function

Choosing a good length function is an important step in the execution of the length-based attack. The original Hughes and Tannenbaum paper on length-based attack [26] was criticized [14] precisely because there was no definition of an effective length function for the braid group, which was the platform group suggested. Other researchers have also written about the importance of the choice of the length function [12, 24, 39].

Since the length-based attack is primarily based on the idea that a conjugation of the right element will decrease the length of the captured package, it is important that in general, the length of an element after conjugation is larger than its length before conjugation; in other words, for $a, b \in G$,

$$\ell(a^{-1}ba) \gg \ell(b) \tag{4.1}$$

In our case, the length of a word is chosen to be the sum of the absolute values of the exponents in its normal form, i.e., if $g \in G$ has normal form $g = g_1^{e_1} \cdots g_n^{e_n}$, then $|g| = \sum_{i=1}^{n} |e_i|$. We choose this function because experimental results below show that it satisfies the requirement 4.1.

The experiments are done by first constructing a polycyclic group G of Hirsch length $h(G)$ following the construction in chapter 2, section 2.1.3. Then an element b of length between 10 and 13 is randomly chosen; we choose elements of this length for consistency in the length-based attack parameters. Another random element a of the same length interval is chosen and b^a is computed, and finally, we compute $|b^a| - |b|$. We ran 100 tests for each group and the average difference is recorded.

polynomial	h(G)	average difference
$x^2 - x - 1$	3	79.92
$x^5 - x^3 - 1$	7	80.17
$x^{11} - x^3 - 1$	16	44.93

As we can see, the average difference is large, in particular, $|b^a| - |b|$ is significantly larger than $|a|$, indicating that the condition $\ell(a^{-1}ba) \gg \ell(b)$ is satisfied.

4.2 Implementation Details

As mentioned above, we want to implement the AAG key-exchange protocol, and simulate attacks on it using the length-based attack. We decide to choose polycyclic groups generated by number fields for this purpose. Below are the details of the implementation.

Each polycyclic group is generated by choosing a polynomial f which is irreducible over $\mathbb{Z}$, then f defines a number field $\mathcal{F}$ over $\mathbb{Q}$. Let $\mathcal{O}_{\mathcal{F}}$ be its maximal order and $\mathcal{U}_{\mathcal{F}}$ its unit group, then $\mathcal{O}_{\mathcal{F}} \rtimes \mathcal{U}_{\mathcal{F}}$ is the desired polycyclic group. This construction is a part of the Polycyclic Package of GAP [8].

A random element a_i, for Alice's public set, or b_i, for Bob's public set, is generated by taking either some random generators of the group or their inverses and multiplying them together, while maintaining that the length of the element is between a predefined minimum and maximum. The advantage of this method is having more control over the length of the element.

Alice's private key A is generated by taking a fixed number of random elements in $\overline{a} = (a_1, \ldots, a_{N_1})$ and multiplying them together, similarly with Bob's private key. Here we forgo control over absolute length of private keys to preserve interesting cases of conjugations actually decreasing the length of b_i, i.e. a commutator-type peak. This way of choosing private keys also better reflects the nature of the underlying group because there might be cancellations during multiplications of the elements making up the key, and so the key length might turn out to be small. This method is similar to what has been used in [39].

4.3 Results

We conducted several sets of tests, all of which were performed on an Intel Core i7 quad-core 2.0GHz computer with 12GB of RAM, running Ubuntu version 12.04 with GAP version 4.5 under 10GB of memory allowance. In all these tests, the polycyclic group G with Hirsch length $h(G)$ is constructed by the above method with polynomial f. The size of Alice's and Bob's public sets are both $N_1 = N_2 = 20$.

4.3.1 Effect of Hirsch length

In the first set of tests, every random element a_i or b_i has length in $[L_1, L_2] = [10, 13]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The time for each batch of 100 tests are recorded together with its success rate. In each case, a time-out of 60 minutes is enforced for each test. The following results are obtained by algorithm 2.

polynomial	h(G)	time	success
$x^2 - x - 1$	3	0.20 hours	100%
$x^5 - x^3 - 1$	7	76.87 hours	35%
$x^7 - x^3 - 1$	10	94.43 hours	8%
$x^9 - 7x^3 - 1$	14	95.18 hours	5%
$x^{11} - x^3 - 1$	16	95.05 hours	5%

From this set of results, we can see that with low Hirsch length, the length-based attack breaks AAG easily with high success rate. However, as Hirsch length is increased to 7, success rate decreases. The tipping point is Hirsch length 10, when success rate is only 8% and the time taken is significant. At higher Hirsch length, we can see the effect of the time-out more prominently as the total time did not increase much more, but the success rate dropped to only 5%. Although a succuss rate of 5% is not negligible, note that for this set of test, we purposely chose a very small value for the key length L. Below, we will see more experiments where the key length is increased.

The important point here is that in order for LBA to have low success rate, we only need to choose group of modest Hirsch length, say 14. This helps in minimizing the amount of memory taken to encode group elements, which makes transmission of elements between the two parties less expensive.

4.3.2 Effect of key length

In the second set of tests, we vary the number of elements L that compose Alice's private key. Myasnikov and Ushakov [39] suggested that LBA with dynamic set (algorithm 2) has high success rate with long generators, i.e. random elements have longer length $[L_1, L_2]$. Therefore, we also vary the length of random elements according to parameters in [39]. The following results are obtained by

Algorithm 2, also with a time-out of 30 minutes.

polynomial	h(G)	[10,13]	[20,23]		[40,43]
		$L = 10$	$L = 10$	$L = 20$	$L = 50$
$x^7 - x^3 - 1$	10	2%	0%	0%	0%
$x^9 - 7x^3 - 1$	14	0%	0%	0%	0%
$x^{11} - 3x^3 - 1$	17	0%	0%	0%	0%

The result of this set of tests indicates that just by increasing the number of generators of Alice's private key from 5 (as in the previous set of tests) to 10, LBA already fails with polycyclic group of Hirsch length as low as 10. Hence, for polycyclic groups to be useful as platform group for AAG, key length needs not be large.

4.3.3 Comparing the four variants of LBA

A novelty of our approach is the comparison of the four variants of the LBA for the first time on any platform group. We implement all the four variants to verify that none of them can break AAG with polycyclic groups as platform.

For comparing the success rate of the four variants of LBA presented in chapter 3, section 3.3, we purposely choose the test parameters very small in this set of of tests. They are as follows: $N_1 = N_2 = 20$, $[L_1, L_2] = [5, 8]$, $L = 5$, there is a time-out of 30 minutes and memory of $M = 500$. The polynomial used is $f = x^3 - x - 1$, generating polycyclic group of Hirsch length only 4.

Algorithm	Time	Success
Algorithm 1 (LBA with backtracking)	0.57 hours	58%
Algorithm 2 (LBA with dynamic set)	37.35 hours	95%
Algorithm 3 (Memory 1)	32.00 hours	36%
Algorithm 4 (Memory 2)	4.01 hours	92%

Algorithm 2 gives the best success rate but took much longer than algorithm 4 which gives a comparable success rate with much shorter time. We conclude that with a sufficient memory size, algorithm 4 is the best variant of LBA.

4.3.4 Exploring the four variants of LBA on commonly used test parameters

In the fourth set of tests, we want to see the effect of the four different variants of LBA applied to test parameters commonly used in other work [13, 39]. Therefore, we keep the same following parameters for all the algorithms: each random element has length in $[L_1, L_2] = [10, 13]$, Alice's private key is the product of 10 elements and the length of both public sets are $N_1 = N_2 = 20$. There is a time-out of 30 minutes per test and in the case of the two memory algorithms, algorithm 3 and algorithm 4, a memory $M = 1000$ is used. The same polycyclic group G of Hirsch length 14 constructed from the polynomial $x^9 - 7x^3 - 1$ is used for all algorithms.

Algorithm	Time	Success
Algorithm 1 (LBA with backtracking)	48.68 hours	0%
Algorithm 2 (LBA with dynamic set)	50.04 hours	0%
Algorithm 3 (Memory 1)	50.00 hours	0%
Algorithm 4 (Memory 2)	49.35 hours	3%

As we can see, algorithm 4 does best under this set of parameters, but even then, it only has a 3% success rate. Contrast this to work done on the braid group of similar parameters, which succeed in breaking AAG with high success rate [39].

To further test algorithm 4, we ran another set of tests where we increase the length of random elements to $[L_1, L_2] = [20, 23]$ and increase the number of factors of the private key to $L = 20$. To give it a chance of success, we increase the memory M to 40,000. The result is still 0% success rate, with a total run time of 189.42 hours.

4.3.5 Effect of time-out increase

Since it is possible that the time-out of 30 minutes for each test is not enough, we ran another set of tests where the time-out is 4 hours for each test. Algorithm 4 showed the most promise, so we chose it with the following parameters: the length of random elements is in the interval $[L_1, L_2] = [20, 23]$, the number of factors of the private key is $L = 20$ and memory M is 1000. The polynomial used is $x^9 - 7x^3 - 1$ with Hirsch length 14. Due to the long time-out, we performed only 50 tests. The result is still 0% success rate, with a total run time of 200.37 hours.

4.3.6 Conclusion

Based on the above experimental results, we conclude that polycyclic groups of high Hirsch lengths are resistant to the length-based attack. Thus, polycyclic groups generated with number fields are suitable to be used as platform group for the AAG key-exchange protocol in particular, and for other non-commutative cryptographic primitives in general.

4.3.7 Additional results with Algorithm 2

These are additional test results conducted with a time-out of 1 hour per test. The polynomials used are f and $h(G)$ is the Hirsch length of the generated polycyclic group. The size of Alice's and Bob's public sets are N_1, N_2 respectively. Every random element a_i or b_i has length in $[L_1, L_2]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The success rate of a batch of 100 tests is recorded in percentage.

polynomial	h(G)	$N_1 = N_2 = 5$		$N_1 = N_2 = 20$
		[5,8]	[15,18]	[10,13]
$x - 1$	1	98%		98%
$x^2 - x - 1$	3	98%	96%	100%
$x^3 - x - 1$	4	95%		100%
$x^5 - x^3 - 1$	7			35%
$x^7 - x^3 - 1$	10			8%
$x^9 - 7x^3 - 1$	14			5%
$x^{11} - x^3 - 1$	16	59%	53%	5%

Chapter 5

Exploring Heisenberg Groups as Platform for the AAG Key-Exchange Protocol

In chapter 4, we explored polycyclic groups generated with number fields as platform group for the AAG key-exchange protocol. Having success with that, we now turn our attention to a different kind of polycyclic groups, Heisenberg groups.

5.1 Heisenberg groups

Before talking about Heisenberg groups, we make a short detour through nilpotent groups. We will see that nilpotent groups is a type of polycyclic group and since Heisenberg groups are nilpotent, they are also polycyclic.

5.1.1 Nilpotent groups

Here, we give a short review of nilpotent groups based on [25, 43, 17].

Let $Z(G)$ be the center of the group G, i.e.,

$$Z(G) = \{g \in G \mid gx = xg \text{ for all } x \in G\}.$$

Definition 5.1.1. The *central series* of a group G is a normal series

$$G = G_1 \trianglerighteq G_2 \trianglerighteq \ldots \trianglerighteq G_{n+1} = 1,$$

where $G_i/G_{i+1} \leq Z(G/G_{i+1})$ for $1 \leq i \leq n$.

Definition 5.1.2. A group G is *nilpotent* if it has such a series, and the *nilpotency class* of G is the length of the shortest central series for G.

Another way to think about nilpotent groups is through the upper and lower central series.

Definition 5.1.3. The *upper central series* of a group G is the ascending series

$$1 = Z_0(G) \trianglelefteq Z_1(G) \trianglelefteq \ldots \trianglelefteq Z_i(G) \trianglelefteq Z_{i+1}(G) \trianglelefteq \ldots,$$

where $Z_0(G) = 1$ and recursively, $Z_{i+1}(G)/Z_i(G) = Z(G/Z_i(G))$ for $i \geq 1$.

Definition 5.1.4. The *lower central series* of a group G is the descending series

$$G = \gamma_1(G) \trianglerighteq \gamma_2(G) \trianglerighteq \ldots \trianglerighteq \gamma_i(G) \trianglerighteq \gamma_{i+1}(G) \trianglerighteq \ldots,$$

where $\gamma_1(G) = G$ and recursively, $\gamma_{i+1}(G) = [G, \gamma_i(G)]$ for $i \geq 1$.

It can be shown that the both the lower and upper central series are central series if and only if they terminate [17]. Hence, a group G is nilpotent if and only if either its lower or upper central series is finite. In this case, the upper and lower central series both have the same length and this length is the nilpotency class of G. An easy example is a nilpotent group of class 1, it is an abelian group.

It can also be shown that if G is a finitely generated nilpotent group then each group in the lower central series of G is finitely generated [17]. Hence, each quotient in the lower central series is a finitely generated abelian group, which means G is polycyclic.

5.1.2 Heisenberg groups

Heisenberg groups have been studied widely from different points of views: analysis, geometry, physics, ... [4]. From the group theory point of view, they are often used as examples of nilpotent groups. For more discussion of Heisenberg groups and their group-theoretic properties, see [10].

The three dimensional Heisenberg group, H, often known as the discrete Heisenberg group, is the group of 3×3 upper triangular matrices of the form

$$\begin{pmatrix} 1 & x & y \\ 0 & 1 & z \\ 0 & 0 & 1 \end{pmatrix}$$

where $x, y, z \in \mathbb{R}$. If we let

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

we can compute directly that $[A, B] = C, [A, C] = [B, C] = I_3$. Hence, another presentation for the three dimensional Heisenberg group is

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

We can also compute directly the center of H and its commutator subgroup. Since they are equal, H is nilpotent of nilpotency class 2.

Generalizing the Heisenberg group, we have the higher dimension Heisenberg groups, $H^{2n+1}, n \geq 1, n \in \mathbb{Z}$. As matrix groups, they are groups of dimension $n+2$ matrices of the form

$$\begin{pmatrix} 1 & x_1 & \dots & x_n & c \\ 0 & 1 & 0 & \dots & y_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & y_1 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

where $x_i, y_i, z \in \mathbb{R}$. The calculation for the commutator subgroup and the center of H^{2n+1} is straight-forward, showing that it is also a nilpotent group, thus, also polycyclic. The Hirsch length of H^{2n+1} is $2n+1$.

The Heisenberg groups of higher dimension H^{2n+1} also has presentation

$$\begin{aligned} H^{2n+1} =& \langle a_1, \dots, a_n, b_1, \dots, b_n, c \mid \\ & [a_i, b_i] = c, [a_i, c] = [b_i, c] = 1, [a_i, a_j] = [b_i, b_j] = 1, i \neq j\rangle \end{aligned}$$

This presentation makes it easy to encode H^{2n+1} in a computer system. In fact, the GAP (Groups, Algorithms, Programming) system [15] can compute H^{2n+1} as part of the polycyclic package by Eick and Nickel [8]. Using this implementation of H^{2n+1}, we want to study the Heisenberg groups as platform group for the AAG key-exchange protocol, in particular, under the length-based attack, one of the major attacks for AAG.

5.2 Results

As a test of the resilience of polycyclic groups against the length-based attack, Garber, Kahrobaei, and Lam [11] implemented four variants of the length-based attack and performed experiments against all four variants. The conclusion was that Algorithm 4, LBA with Memory 2, was the most effective. Hence, in this study, we use Algorithm 4 for all of our experiments.

We performed several sets of tests, all of which were run on an Intel Core I7 quad-core 2.0GHz computer with 12GB of RAM, running Ubuntu version 12.04 with GAP version 4.5 and 10GB of memory allowance. In all these tests, the Heisenberg group G with $2n$ generators having Hirsch length $h(G) = 2n + 1$ is generated using GAP with the Polycyclic group package [8]. The size of Alice's and Bob's public sets are both $N_1 = N_2 = 20$, the memory used is $M = 1000$ and the time-out is 30 minutes. The implementation details of how a random element is generated and how the private keys are generated are similar to that of chapter 4, section 4.2.

To see the effect of element length, we fix a small key length, $L = 10$, but changes the range of random element between $[L_1, L_2] = [10, 13]$ and [20,23]. The result is as follows:

n	h(G)	$[L_1, L_2] = [10, 13]$	$[L_1, L_2] = [20, 23]$
5	11	29%	53%
6	13	69%	39%
7	15	51%	58%
8	17	62%	67%

As we can see, changing the length of random element does not have a dramatic effect on the success rate. Hence, to ensure a lower rate of success, one should look for other factors like the Hirsch length or the key length.

To see the effect of Hirsch length, we use a small key length $L = 10$ to increase the possibility of success. The length of each random element is in the range $[L_1, L_2] = [10, 13]$. The result is as follows:

n	h(G)	Time	Success rate
3	7	45.42 hours	11%
5	11	37.82 hours	29%
6	13	21.12 hours	69%
7	15	28.95 hours	51%
8	17	24.33 hours	62%

Curiously, the higher the Hirsch length in this case, the higher and quicker the length-based attack succeeded. This interesting result warrant further investigations.

To illustrate the effect of key length, in the following experiment, we let the length of each random element to be in $[L_1, L_2] = [20, 23]$ and vary the key length L.

n	h(G)	L=10	L=20	L=50
5	11	53%	11%	1%
6	13	39%	7%	0%
7	15	58%	5%	1%
8	17	67%	9%	7%

Clearly, increasing the key length reduced the success rate dramatically.

What is interesting is that the same algorithm, together with the same parameters, when applied on polycyclic group of Hirsch length 10 generated with number field as in [11] gave 0% success rate. However, it is clear that we need different parameters to ensure low success rates.

We turn the key length to $L = 50$, and let length of random element be $[L_1, L_2] = [40, 43]$. Note that this is the parameters used in [39], which succeeded in breaking AAG with braid groups.

n	h(G)	L=50
6	13	0%
7	15	0%
8	17	1%

As expected, this produces almost zero success rate. We recommend this as parameters for Heisenberg group as platform for AAG.

Finally, as a stress test, we increase the time-out to 4 hours instead of the usual 30 minutes per test. Because of the long time-out, we only do 50 tests each, instead of 100 test batch as in the previous experiments. The element length is kept at $[L_1, L_2] = [20, 23]$, key length is $L = 20$.

n	h(G)	L=20
6	13	3%
7	15	4%
8	17	7%

Even at such long time-out, success rate is quite modest, given that the key length we use here is quite small.

With these results, we conclude that the Heisenberg groups work well as platform for the AAG protocol given the correct parameters. This strengthens the idea to use polycyclic group as platform for AAG in particular, and non-commutative cryptographic primitives in general.

Chapter 6

The HKKS key-exchange protocol

In this chapter, we talk about the HKKS key-exchange protocol, invented by Habeeb, Kahrobaei, Koupparis, and Shpilrain in [16]. The protocol uses semidirect product and extensions by automorphisms. Since it has close relation to the Diffie-Hellman key-exchange protocol, we are going to make a short visit to the Diffie-Hellman protocol first.

6.1 The Diffie-Hellman key-exchange protocol

In chapter 1, we introduced the notion of public-key cryptography, a revolutionary concept that makes possible a large part of the current e-commerce system. What we did not mention was that the field of public-key cryptography was started single-handedly from a paper by Whitfield Diffie and Martin Hellman [6] (with contributions from Ralph Merkle), which detailed the first public key-exchange protocol.

In its original form, the Diffie-Hellman key-exchange protocol uses the multiplicative group of integers modulus p, $G = (\mathbb{Z}/p\mathbb{Z})^{\times}$, where p is a prime, and g is primitive mod p, i.e., g is a generator of the cyclic group G. The following steps outline the protocol:

Step 0 Alice and Bob agree on a prime p and g, a primitive root mod p. Both p and g are public.

Step 1 Alice secretly chooses a random natural number a and sends g^a to Bob.

Step 2 Bob secretly chooses a random natural number b and sends g^b to Alice.

Step 3 Alice computes $K_A = (g^b)^a = g^{ba}$.

Step 4 Bob computes $K_B = (g^a)^b = g^{ab}$.

Since $ab = ba$ because $a, b \in \mathbb{N}$, we have $K_A = K_B$. Hence, Alice and Bob now possess the same secret key.

The security of the protocol is based on the *Diffie-Hellman problem.*

Definition 6.1.1. *(Diffie-Hellman Problem)* Given a finite cyclic group G, a generating element g and g^a, g^b, recover g^{ab} where $a, b \in \mathbb{N}$.

This is similar to the *discrete logarithm problem.*

Definition 6.1.2. *(Discrete Logarithm Problem or DLP)* Given a finite cyclic group G, a generating element g and g^a where $a \in \mathbb{N}$, recover a.

Clearly, an efficient algorithm that can solve DLP will be able to solve the Diffie-Hellman problem: given g^a, g^b, a and b can be computed by the DLP-solver, then the secret key g^{ab} can be easily computed. However, the two problems are not known to be equivalent. Currently, the Diffie-Hellman problem is considered to be hard for a “good” choice of parameters [35].

A short note about computation efficiency of the protocol: it may seems that to compute g^a or g^b, either party have to perform $O(|g|)$ multiplications; however, g^a or g^b can be computed in $O(log\ a)$ multiplications using the “square-and-multiply” method. See [35] for details of the method.

For more details on the Diffie-Hellman protocol as well as a collection of known attacks, see the book by Menezes et. al. [35].

6.2 The HKKS Key-Exchange Protocol

In this section, we introduce the key-exchange protocol invented by Habeeb, Kahrobaei, Koupparis, and Shpilrain based on semidirect product [16]. We will then describes some platforms that have been suggested for this protocol. For now, we make a small detour to talk about semidirect product.

6.2.1 Semidirect product and extensions by automorphisms

Definition 6.2.1. Let G, H be two groups, let $Aut(G)$ be the group of automorphisms of G, and let $\phi : H \to Aut(G)$ be a homomorphism. The semidirect product of G and H is the set

$$\Gamma = G \rtimes_{\phi} H = \{(g,h) \mid g \in G, h \in H\},$$

with the group operation given by

$$(g,h) \cdot (g',h') = (\phi(h)(g) \cdot g', h \cdot h').$$

In the case that $H = Aut(G)$ then the group Γ is called the *holomorph*, $Hol(G)$, of the group G.

Definition 6.2.2. Let G be a group, the holomorph of G is the set

$$Hol(G) = G \rtimes_{Id} Aut(G) = \{(g,\phi) \mid g \in G, \phi \in Aut(G)\},$$

with the group operation given by

$$(g,\phi) \cdot (g',\phi') = (\phi'(g) \cdot g', \phi \cdot \phi').$$

Note that when we write $\phi \cdot \phi'$ we mean ϕ is applied first.

A special case of the semidirect product construction is *extension by automorphisms* where we do not use the whole group $Aut(G)$ but only a cyclic subgroup of it, which is generated by a fixed $\phi \in Aut(G)$. The resulting object Γ is also a group. Since every automorphism that we are concerned with now is an element of $\langle\phi\rangle$, the group operation becomes

$$(g,\phi^r)(h,\phi^s) = (\phi^s(g) \cdot h, \phi^{r+s}).$$

In particular, to calculate exponents of an element of Γ, we have:

$$\begin{aligned}(g,\phi)^m &= (g,\phi)\cdot(g,\phi)\cdot(g,\phi)^{m-2}\\ &= (\phi(g)\cdot g,\phi^2)\cdot(g,\phi)\cdot(g,\phi)^{m-3}\\ &= (\phi^2(g)\cdot\phi(g)\cdot g,\phi^3)\cdot(g,\phi)\cdot(g,\phi)^{m-4}\\ &= \dots\\ &= (\phi^{m-1}(g)\cdot\phi^{m-2}(g)\cdots\phi(g)\cdot g,\phi^m)\end{aligned}$$

This construction works just as well with a semigroup G and and endomorphism ϕ instead of a group and an automorphism. In that case, the resulting object Γ is a semigroup instead.

6.2.2 The HKKS key-exchange protocol

The HKKS key-exchange protocol is based on extension of a (semi)group by automorphisms [16]. It has some resemblance to the classical Diffie-Hellman protocol, but there are distinctive features that give it advantages. The following steps describes the protocol:

Step 0 Alice and Bob agree on a (semi)group G, an element $g \in G$ and an automorphism $\phi \in Aut(G)$ (or an endomorphism $\phi \in End(G)$). All G, g and ϕ are public. Alice chooses a private $m \in \mathbb{N}$ and Bob chooses a private $n \in \mathbb{N}$.

Step 1 Alice computes $(g,\ \phi)^m = (\phi^{m-1}(g)\cdot\phi^{m-2}(g)\cdots\phi(g)\cdot g,\phi^m)$ and sends **only the first component** of this pair to Bob. Thus, she sends to Bob **only** the element $a = \phi^{m-1}(g)\cdot\phi^{m-2}(g)\cdots\phi(g)\cdot g$.

Step 2 Bob computes $(g,\ \phi)^n = (\phi^{n-1}(g) \cdot \phi^{n-2}(g) \cdots \phi(g) \cdot g, \phi^n)$ and sends **only the first component** of this pair to Alice. Thus, he sends to Alice **only** the element $b = \phi^{n-1}(g) \cdot \phi^{n-2}(g) \cdots \phi(g) \cdot g$.

Step 3 Alice computes $(b,\ x) \cdot (a,\ \phi^m) = (\phi^m(b) \cdot a, x \cdot \phi^m)$. Her key is now $K_A = \phi^m(b) \cdot a$. Note that she does not actually "compute" $x \cdot \phi^m$ because she does not know the automorphism $x = \phi^n$; recall that it was not transmitted to her. But she does not need it to compute K_A.

Step 4 Bob computes $(a,\ y) \cdot (b,\ \phi^n) = (\phi^n(a) \cdot b, y \cdot \phi^n)$. His key is now $K_B = \phi^n(a) \cdot b$. Again, Bob does not actually "compute" $y \cdot \phi^n$ because he does not know the automorphism $y = \phi^m$.

Note that

$$\begin{aligned} K_A &= \phi^m(b) \cdot a \\ &= \phi^m\big(\phi^{n-1}(g) \cdot \phi^{n-2}(g) \cdots \phi(g) \cdot g\big) \cdot \phi^{m-1}(g) \cdot \phi^{m-2}(g) \cdots \phi(g) \cdot g \\ &= \phi^{m+n-1}(g) \cdot \phi^{m+n-2}(g) \cdots \phi^{m+1}(g) \cdot \phi^m(g) \cdot \phi^{m-1}(g) \cdots \phi(g) \cdot g \end{aligned}$$

which is the first coordinate of $(g, \phi)^{m+n}$ and also equal to $\phi^n(a) \cdot b = K_B$. Hence, $K_A = K_B = K$, the shared secret key.

Note that, in contrast with the "standard" Diffie-Hellman key-exchange, correctness here is based on the equality $h^m \cdot h^n = h^n \cdot h^m = h^{m+n}$ rather than on the equality $(h^m)^n = (h^n)^m = h^{mn}$. In the "standard" Diffie-Hellman set up, this trick would not work because, if the shared key K was just the product of two openly transmitted elements g^a, g^b, then anybody, including the eavesdropper, could compute K.

For the computational cost of the protocol, it may seem at first that both parties have to compute a product of m (or n) elements of the (semi)group G. However, $(g, \phi)^m$ (or $(g, \phi)^n$) can be computed using the "square-and-multiply" method as in the standard Diffie-Hellman protocol. The exact cost, of course, depends on the platform being used.

6.3 Platforms for the HKKS key-exchange protocol

6.3.1 Multiplicative group of integers modulus p

The simplest instantiation of the protocol uses the multiplicative group of integers modulus p where p is a prime [16]. The public endomorphism ϕ is selected by choosing $k > 1$ so that for $h \in (\mathbb{Z}/p\mathbb{Z})^\times$, we have $\phi(h) = h^k$. If k is relatively prime to $p-1$ then ϕ is actually an automorphism. Alice and Bob choose their private m, n according to the guideline for the Diffie-Hellman protocol.

Note that if $g \in (\mathbb{Z}/p\mathbb{Z})^\times$, then exponentiation of an element in the semidirect product is:

$$\begin{aligned}
(g, \phi)^m &= (\phi^{m-1}(g) \cdot \phi^{m-2}(g) \cdots \phi(g) \cdot g, \phi^m) \\
&= (g^{k^{m-1}} \cdot g^{k^{m-2}} \cdots g^k \cdot g, \phi^m) \\
&= (g^{k^{m-1}+\ldots+k+1}, \phi^m) = (g^{\frac{k^m-1}{k-1}}, \phi^m)
\end{aligned}$$

Hence, Alice sends to Bob $g^{\frac{k^m-1}{k-1}}$ and Bob sends to Alice $g^{\frac{k^n-1}{k-1}}$. The shared secret key is then the first component of $(g, \phi)^{m+n}$ which is $g^{\frac{k^{m+n}-1}{k-1}}$.

Eve can choose a "direct" attack of the protocol by trying to recover Alice's and Bob's private key m, n. To guess either of them, Eve will have to first recover $\frac{k^m-1}{k-1}$ from $g^{\frac{k^m-1}{k-1}}$ and then recover m from $\frac{k^m-1}{k-1}$, solving the discrete log problem twice.

Eve can also look at this as an analog of the Diffie-Hellman problem: recover the shared key $K = g^{\frac{k^{m+n}-1}{k-1}}$ from the triple $(g, g^{\frac{k^m-1}{k-1}}, g^{\frac{k^n-1}{k-1}})$. Since g, k are public, this is equivalent to recovering $g^{k^{m+n}}$ from the triple (g, g^{k^m}, g^{k^n}), which is exactly the standard Diffie-Hellman problem.

Since this instantiation of the protocol is the equivalent of the classic Diffie-Hellman protocol, breaking the HKKS protocol for any cyclic group would imply breaking the Diffie-Hellman protocol.

6.3.2 Matrices over group rings

In [16], another platform of the HKKS protocol is also introduced, this time using a semigroup and inner automorphism instead.

A (semi)group ring $R[G]$ of a (semi)group G over a commutative ring R is the set of all formal sums

$$\sum_{g_i \in G} r_i g_i,$$

where $r_i \in R$, and almost all r_i are zero.

The sum of two elements in $R[G]$ is defined by

$$\left(\sum_{g_i \in G} a_i g_i\right) + \left(\sum_{g_i \in G} b_i g_i\right) = \sum_{g_i \in G} (a_i + b_i) g_i.$$

The product of two elements in $R[G]$ is defined using distributivity.

The platform suggested by Habeeb, Kahrobaei, Koupparis and Shpilrain uses the semigroup G of 3×3 matrices over the group ring $\mathbb{Z}_7[A_5]$ where A_5 is the alternating group on 5 elements. The semigroup G is extended by an inner automorphism ϕ_H, which is conjugation by a matrix $H \in GL_3(\mathbb{Z}_7[A_5])$, i.e., $\phi_H(M) = H^{-1}MH$. Both $M \in G$ and H are public knowledge. Conveniently, for any matrix $M \in G$, and any integer $k \geq 1$, we have $\phi_H^k(M) = H^{-k}MH^k$.

As usual, we look at exponentiation of an element of the semidirect product:

$$\begin{aligned}(M, \phi_H)^m &= (\phi_H^{m-1}(M) \cdot \phi_H^{m-2}(M) \cdots \phi_H(M) \cdot M, \phi_H^m)\\ &= (H^{-m+1}MH^{m-1} \cdots H^{-2}MH^2 \cdot H^{-1}MH, \phi_H^m)\\ &= H^{-m}(HM)^m\end{aligned}$$

Hence, Alice sends to Bob the matrix $H^{-m}(HM)^m$ and Bob sends to Alice the matrix $H^{-n}(HM)^n$, and the shared secret key is $H^{-(m+n)}(HM)^{(m+n)}$.

The eavesdropper can try to recover the private exponent m from $H^{-m}(HM)^m$. This problem appears to be hard, in the special case that $H = Id$, we have the analog of the DLP for matrices over $\mathbb{Z}_7[A_5]$: recover m given M and M^m. This problem was addressed in [30], which shows using statistical experiments that for a random matrix M, matrices M^m are indistinguishable from random.

Chapter 7

Matrices over Galois Field: A New Platform for HKKS

As we have already seen in chapter 6, the HKKS key-echange protocol can be used with *any* non-commutative group G if ϕ is selected to be a non-trivial inner automorphism, i.e., conjugation by an element which is not in the center of G. Furthermore, it can be used with any non-commutative *semigroup* G as well, as long as G has some invertible elements; these can be used to produce inner automorphisms. In this section, we introduce a new platform for the HKKS key-exchange protocol, that of matrices over a Galois field.

7.1 Galois Field

First, a short introduction to Galois field, otherwise known as finite field. In our application, we are only concerned with *binary fields*, $\mathbb{GF}(2^m)$, that is, finite fields of order 2^m. Binary fields are widely used in cryptography and there are several well-studied algorithms in binary field arithmetic, making computation with them very fast.

One way to represent $\mathbb{GF}(2^m)$ is to use *binary polynomials*, whose coefficients are in the field $\mathbb{GF}(2) = \{0, 1\}$. Then the elements of $\mathbb{GF}(2^m)$ are binary polynomials of degree at most $m - 1$:

$$\mathbb{GF}(2^m) = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_2z^2 + a_1z + a_0 \mid a_i \in \{0, 1\}\}.$$

For each $\mathbb{GF}(2^m)$, there is an irreducible binary polynomial $f(z)$ of degree m such that $\mathbb{GF}(2^m) \cong \mathbb{Z}_2[x]/(f(x))$. Multiplication of field elements of $\mathbb{GF}(2^m)$ is performed modulo $f(z)$. Addition of field elements is the usual polynomial addition in $\mathbb{Z}_2[x]$.

To represent field element, we associate a field element $a(z) = a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \ldots + a_2z^2 + a_1z + a_0$ with the binary number $a = (a_{n-1} \ldots a_2a_1a_0)$. Addition and subtraction in the field is then simply bitwise XOR (exclusive-or). Multiplication, without reduction, can be done using a number of algorithms, mostly taking advantage of vector shifts, and thus can be realized in polynomial time (see [18]).

Polynomial squaring of a field element $a(z)$ can be achieved by inserting a 0 bit between consecutive bits of the binary representation of $a(z)$, hence it is a linear operation. Reduction of a binary polynomials are relatively fast, but particularly fast if certain irreducible polynomials are used for certain binary field. The National Institute of Standards and Technology (NIST) has recommendations for such polynomials [40].

For more details of the various algorithms for binary field arithmetic, see the book by Menezes et. al. [35] and the book by Hankerson et. al. [18]. The software package RELIC implements, among other things, binary fields and several algorithms for binary field arithmetic [3].

7.2 HKKS with Matrices over a Galois Field

As a new platform for the HKKS key-exchange protocol, we suggest G to be the semigroup of 2×2 matrices over the Galois field $\mathbb{GF}(2^t)$. Here we use an extension of the semigroup G by an endomorphism φ, which is a composition of a conjugation by a matrix $H \in GL_2(\mathbb{GF}(2^t))$ with the endomorphism ψ that raises each entry of a given matrix to the power of 4. The composition is such that ψ is applied first, followed by conjugation, i.e., with public element M and H, we have $\varphi(M) = H^{-1}\psi(M)H$. Thus, for any matrix $M \in G$ and for any integer $k \geq 1$, we have:

$$\begin{aligned}\varphi^k(M) &= \varphi^{k-1}(H^{-1}\psi(M)H)\\ &= \varphi^{k-2}\big(\varphi(H^{-1}\psi(M)H)\big) = \varphi^{k-2}\big(H^{-1}\psi(H^{-1})\psi^2(M)\cdots\psi(H)H\big)\\ &\cdots\\ &= H^{-1}\psi(H^{-1})\cdots\psi^{k-1}(H^{-1})\psi^k(M)\psi^{k-1}(H)\cdots\psi(H)H\end{aligned}$$

The HKKS protocol is specialized to this platform as follows:

Step 0 Alice and Bob agree on a degree k for the Galois field and public matrices $M \in G$ and $H \in GL_2(\mathbb{GF}(2^t))$. Alice selects a private positive integer m, and Bob selects a private positive integer n.

Step 1 Alice computes $(M, \varphi)^m$ and sends **only the first component** of this pair to Bob. Thus, she sends to Bob **only** the matrix

$$A = H^{-1}\psi(H^{-1})\cdots\psi^{m-1}(H^{-1})\psi^{m}(M)\psi^{m-1}(H)\cdots\psi(H)H.$$

Step 2 Bob computes $(M, \varphi_{_H})^n$ and sends **only the first component** of this pair to Alice. Thus, he sends to Alice **only** the matrix

$$B = H^{-1}\psi(H^{-1})\cdots\psi^{n-1}(H^{-1})\psi^{n}(M)\psi^{n-1}(H)\cdots\psi(H)H.$$

Step 3 Alice computes $(B, x)\cdot(A,\ \varphi^m) = (\varphi^m(B)\cdot A,\ x\cdot\varphi^m)$. Her key is now $K_A = \varphi^m(B)\cdot A$, which is the first component of $(M,\varphi)^{m+n}$. Note that she does not actually "compute" $x\cdot\varphi^m$ because she does not know the automorphism $x = \varphi^n$; recall that it was not transmitted to her. But she does not need it to compute K_A.

Step 4 Bob computes $(A, y)\cdot(B,\ \varphi^n) = (\varphi^n(A)\cdot B,\ y\cdot\varphi^n)$. His key is now $K_B = \varphi^n(A)\cdot B$. Again, Bob does not actually "compute" $y\cdot\varphi^n$ because he does not know the automorphism $y = \varphi^m$.

Since $(B, x)\cdot(A,\ \varphi^m) = (A,\ y)\cdot(B,\ \varphi^n) = (M,\ \varphi)^{m+n}$, we have $K_A = K_B = K$, the shared secret key.

7.3 Security Analysis

7.3.1 Security assumption

Since the shared secret key K is the first component of $(M,\ \varphi)^{m+n}$, it is a product

$$K = \varphi^{m+n-1}(M) \cdots \varphi(M)M, \text{ where}$$
$$\varphi^i(M) = H^{-1}\psi(H^{-1}) \cdots \psi^{i-1}(H^{-1})\psi^i(M)\psi^{i-1}(H) \cdots \psi(H)H$$

The security assumption of the protocol is that it is hard to recover K from the matrices H, M, A, and B, where A and B are the first components of $(M,\ \varphi)^m$ and $(M,\ \varphi)^n$, respectively.

To analyze the security assumption of the protocol, we first drop ψ and only consider $\varphi = H^{-1}MH$, we note that now $\varphi^k = H^{-k}MH^k$. A further simplification, letting $H = Id$ yields the analog of the discrete log problem for matrices over $\mathbb{GF}(2^t)$: recover m given M and M^m. Standard attacks of the discrete log problem like “Pollard’s rho” or “baby-step giant-step” are based on the difficulty of detecting cycles. In our system, looking for a “cycle” would mean looking for i, j such that $x^{-i}y^i = x^{-j}y^j$, where $x = H,\ y = HM$. Assuming that both x and y are invertible, the last equality is equivalent to $x^k = y^k$, where $k = i - j$. Thus, the problem becomes not detecting a cycle, but rather to find an intersection of two independent cycles. This is a more general problem than cycle detection and apparently more difficult.

Another attack that Eve can employ is by looking at the determinants of the public matrices H, M, A, and B. She can reduce the problem of recovering the private exponent m to the discrete log problem for the pair $(\det M, (\det M)^n)$. To foil this kind of attack, it makes sense to select a matrix M with the determinant equal to 0 or 1.

7.3.2 Questions about randomness

To verify the robustness and security of our platform, we have experimentally addressed two questions. The first question is whether or not any information about the private exponent n is leaked from transmission. That is, for a random exponent n, how different is the first component of $(M,\ \varphi)^n$ from N, where N is a random matrix?

To address this question, we chose the semigroup G to be 2×2 matrices over the Galois field $\mathbb{GF}(2^{127})$, hence each entry of a matrix is a bit string of length 127. We generated random matrices $P, N \in G$, repeated this process 500 times and generated a frequency distribution table for the two distributions. We then plotted the resulting $Q - Q$ plots of the frequency distributions generated by the corresponding entries of the first component of $(P,\ \varphi)^n$. The result is indistinguishable from a random matrix.

The second question is to determine how different the final shared key is from a random matrix. More specifically, if Alice and Bob choose secret integers m and n respectively, how different is the first component of $(P, \varphi)^{n+m}$ from $(P, \varphi)^c$, where c is of the same bit size as $n + m$?

We used a similar experiment to determine this question, namely, we generated a random matrix P, and random m, n and c such that $m + n \approx c$. We repeated this process 500 times and generated a frequency distribution table for the two distributions. From the table, we produced $Q - Q$ (quantile) plots of the entries of the two matrices: the corresponding entries of the first component of $(P, \varphi)^{m+n}$ and $(P, \varphi)^c$. The experiment result confirmed that there is no difference between them.

7.4 Implementation and Results

7.4.1 Parameters and key generation

The private exponents m and n of the HKKS protocol should be of the magnitude 2^t, where t is the security parameter, to make brute force search infeasible. Thus, m and n are roughly t bits long.

In our implementation of the protocol, we chose $t = 127$ and $t = 571$, aiming for 127-bit and 571-bit security. That means we are working with Galois field $\mathbb{GF}(2^{127})$ and $\mathbb{GF}(2^{571})$, hence the private keys m, n have size 127 bits and 571 bits respectively.

Our realization of the Galois field $\mathbb{GF}(2^{127})$ is the factor algebra $\mathbb{Z}_2[x]/\langle p(x)\rangle$, where $\langle p(x)\rangle$ is the ideal of the polynomial algebra $\mathbb{Z}_2[x]$ generated by the irreducible polynomial $p(x) = x^{127} + x^{63} + 1$. Elements of $\mathbb{GF}(2^{127})$ are therefore polynomials of degree at most 126 over $\mathbb{Z}_2$.

Similarly, our realization of the Galois field $\mathbb{GF}(2^{571})$ is the factor algebra $\mathbb{Z}_2[x]/\langle q(x)\rangle$, where $\langle q(x)\rangle$ is the ideal of $\mathbb{Z}_2[x]$ generated by the irreducible polynomial $q(x) = x^{571} + x^{10} + x^5 + x^2 + 1$. Elements of $\mathbb{GF}(2^{571})$ are therefore polynomials of degree at most 570 over $\mathbb{Z}_2$. Both the polynomials $p(x)$ and $q(x)$ are chosen because there are algorithms concerning trinomials and quintic polynomials that make binary field arithmetics in them faster. See the RELIC package [3] for more details.

The public matrix M is selected as a random 2×2 matrix over the Galois field $\mathbb{GF}(2^{127})$ (respectively, $\mathbb{GF}(2^{571})$), which means that each entry of M is a random bit string of length 127 (respectively, length 571) corresponding to coefficients of a polynomial of degree at most 126 (resp. 570) over $\mathbb{Z}_2$. For security reasons M should be either non-invertible or have determinant 0 or 1. To select such a 2×2 matrix, we first select 3 entries randomly, and then select the remaining entry so that the determinant is what we want. This is an efficient procedure.

In the case of $t = 271$, the bit complexity of the matrix M is $127 \cdot 4 = 508$ bits, the whole public key consists of the matrix M and an invertible matrix H, so the total size of the public key is 1016 bits. In the case of $t = 271$, the bit complexity of a 2×2 matrix over $\mathbb{GF}(2^{571})$ is 2284 bits, so the total size of the public key is 4568 bits.

Then, we need to sample an *invertible* 2×2 matrix H over a Galois field. To do that, we sample a random 2×2 matrix as above, compute its determinant and check that it is not equal to 0 in the Galois field. If it is equal to 0 (this can happen with small probability), then we start over. Also, having computed the determinant of H, we then easily compute H^{-1}. Finally, we have to check that H does not commute with M, i.e., $HM \neq MH$. If it does, we select a different H.

7.4.2 How to make sure the base element has a large order

We note that there is always a concern (also in the standard Diffie-Hellman protocol) about the order of a public element: if the order is too small, then a brute force attack may be feasible. In our situation, this concern is significantly alleviated by the fact that our transmissions are products of powers of different matrices, $\varphi^k(M)$ for $k = 1, \ldots, m-1$, rather than powers of a single matrix. Therefore, even if the order of one of the matrices happens to be small by accident, this does not mean that the whole product will go into loop of a small size.

However, if one wants a guarantee that the base element $(M,\ \varphi)$ has a large order, this requires some extra effort. First we observe that the order of an element (g, φ) of a semidirect product tends to have the magnitude of the g.c.d. of the orders of the individual elements g and φ in their "native" (semi)groups.

Particularly, we can try to bound the order of the base element in our suggested platform of matrices over Galois field. Suppose the base element $(M,\ \varphi)$ has order k, the second component of $(M,\ \varphi)^k$ is $varphi^k$. This endomorphism acts on an arbitrary 2×2 matrix S over a Galois field by:

$$\varphi^k(S) = H^{-1}\psi(H^{-1}) \cdots \psi^{k-1}(H^{-1})\psi^k(S)\psi^{k-1}(H) \cdots \psi(H)H.$$

To bound the order k from below, we can look at the determinant of $\varphi^k(S)$. Since the determinant map is multiplicative, this is equal to $\det(\psi^k(S))$. Since we are working in characteristic 2, we have $\det(\psi^k(S)) = \det(\psi(S)^k) = (\det(\psi(S)))^k$ (recall that the endomorphism ψ acts by raising each entry of S to the power of 4). Now notice that if S is invertible, then $\det(S)$ is an element of the multiplicative group of the relevant Galois field.

If, for example, the Galois field is $\mathbb{GF}(2^{127})$, then the order of this group is $2^{127} - 1$, which happens to be a prime number. Therefore, if $\det(S) \neq 1$, then $\det(S)$, as well as $\det(\psi(S))$, has order $2^{127} - 1$ in the multiplicative group of $\mathbb{GF}(2^{127})$. This gives a lower bound for the order of the second component of a base element (M, ϕ). The actual order should be much higher for "generic" matrices S and H, but if one wants a guaranteed lower bound, then $2^{127} - 1$ should be satisfactory.

7.4.3 Computational cost and run time

Recall the transmitted elements are

$$A = H^{-1}\psi(H^{-1}) \cdots \psi^{m-1}(H^{-1})\psi^{m}(M)\psi^{m-1}(H) \cdots \psi(H)H, \text{ and}$$
$$B = H^{-1}\psi(H^{-1}) \cdots \psi^{n-1}(H^{-1})\psi^{n}(M)\psi^{n-1}(H) \cdots \psi(H)H$$

It may seem that the parties have to compute a product of m (respectively, n) elements of G, the semigroup of 2×2 matrices over a Galois field. However, since the parties obtain A (or B) as the first component of $(M, \varphi)^m$ (resp. $(M, \varphi)^n$), they actually compute powers of an element of the semigroup Γ, which is an extension of G by an endomorphism. Hence, they can use the "square-and-multiply" method, as in the standard Diffie-Hellman protocol.

To apply φ to an element of G, first there's the cost of conjugating by a matrix H, which amounts to just two multiplications of matrices in G (which boils down to 8 multiplications in $\mathbb{GF}(2^{127})$ or $\mathbb{GF}(2^{571})$). Second is the cost of applying the endomorphism ψ which in fact, does not require any multiplications, but just inserting "0" bits in a bit string representing an element the Galois field, see [3].

To compute powers of φ applied to an element of G, recall

$$\varphi^k(M) = H^{-1}\psi(H^{-1})\cdots\psi^{k-1}(H^{-1})\psi^k(M)\psi^{k-1}(H)\cdots\psi(H)H.$$

This can be broken into products of three components: $H^{-1}\psi(H^{-1})\cdots\psi^{k-1}(H^{-1})$, $\psi^k(M)$ and $\psi^{k-1}(H)\cdots\psi(H)H$. The first and third components are inverses of each other so we can compute the third component and take its inverse, which amounts to calculating determinant of a matrix and finding its inverses in the Galois field, all of which can be done efficiently.

To calculate $\psi^{k-1}(H)\cdots\psi(H)H$, in the case of $\mathbb{GF}(2^{127})$, we pre-compute powers $\psi^{2^i}(H)$ for $i = 0, \ldots, 128$, each of which is a matrix of $127 \cdot 4 = 508$ bits, for a total of 65532 bits (about 8 Kilobyte) to be put in storage. Then we again use "square-and-multiply" to find the long product. Similarly, in the case of $\mathbb{GF}(2^{571})$, we also pre-compute powers $\psi^{2^i}(H)$ for $i = 0, \ldots, 572$, each of which is a matrix of $571 \cdot 4 = 2284$ bits. Hence, we need to save 1308732 bits, about 160 Kilobyte. As we can see, the storage needed is minuscule.

To calculate $\psi^k(M)$, we raise each entry of the matrix M to the power 4^k mod $2^{127} - 1$ (entries of M are elements of $\mathbb{GF}(2^{127})$). This relies on RELIC's Montgomery powering ladder method.

We used the RELIC package [3] for big integers and binary field arithmetic in $\mathbb{GF}(2^{127})$ and $\mathbb{GF}(2^{571})$. The computations are done on a desktop computer with the following specifications: Intel Core 2 Duo CPU at 3.00GHz x 2, 8 GiB

RAM, running Ubuntu 12.04.

With the parameters specified in section 7.4.1, we ran two types of experiments, one where the endomorphism φ is $\varphi = H^{-1}\psi(M)H$ and the other where $\varphi = H^{-1}MH$ (i.e. dropping the extra endomorphism ψ which raises each entry of a matrix to the power of 4.) We measure the total time taken vs. time for private key generation and shared key computation, noting that in practice, once the key-exchange program is set up, it can be re-used to generate many different keys by randomizing m, n.

	with psi		without psi	
	total time	key generation & computation	total time	key generation & computation
$\mathbb{GF}(2^{127})$	0.323s	0.149s	0.153s	0.035s
$\mathbb{GF}(2^{571})$			6.907s	1.247s

7.4.4 Conclusion

We have presented a new platform for the HKKS key-exchange protocol based on extension by endomorphisms of a semigroup of matrices over $\mathbb{GF}(2^{127})$ and $\mathbb{GF}(2^{571})$. It has some resemblance to the classical Diffie-Hellman protocol, but there are several distinctive features that, we believe, give our protocol important advantages:

- Even though the parties do compute a large power of a public element (as in the classical Diffie-Hellman protocol), they do not transmit the whole result, but rather just part of it.

- By varying automorphisms (or endomorphisms) used for extension, we get new security assumptions. We illustrate this point by using a particular endomorphism which is a composition of an inner automorphism (i.e., conjugation by an invertible matrix) with the endomorphism that raises each entry of a given matrix to the power of 4.

- New security assumptions alluded to in the previous bullet point allow us to dodge standard attacks based on cycle detection methods.
- By working in a Galois field of characteristic 2, we make computation very efficient.

Bibliography

[1] I. Anshel, M. Anshel, and D. Goldfeld, *An algebraic method for public-key cryptography*, Mathematical Research Letters **6** (1999), 287–291.

[2] M. Anshel and D. Kahrobaei, *Decision and search in non-abelian Cramer-Shoup public key cryptosystem*, Groups, Complexity, Cryptology **1** (2009), 217–225.

[3] D. F. Aranha and C. P. L. Gouvêa, *RELIC is an Efficient LIbrary for Cryptography*, `http://code.google.com/p/relic-toolkit/`.

[4] Ernst Binz and Sonja Pods, *The geometry of Heisenberg groups: with applications in signal theory, optics, quantization, and field quantization*, vol. 151, American Mathematical Soc., 2008.

[5] Wieb Bosma, John Cannon, and Catherine Playoust, *The MAGMA algebra system I: The user language*, Journal of Symbolic Computation **24** (1997), no. 3, 235–265.

[6] Whitfield Diffie and Martin E Hellman, *New directions in cryptography*, Information Theory, IEEE Transactions on **22** (1976), no. 6, 644–654.

[7] B. Eick and D. Kahrobaei, *Polycyclic groups: a new platform for cryptography?*, (2004), `http://arxiv.org/abs/math/0411077`.

[8] B. Eick and W. Nickel, *Polycyclic: Computation with polycyclic groups, a GAP 4 package.*

[9] Bettina Eick, *Algorithms for polycyclic groups*, Habilitationsschrift, Universitt Kassel, 2000, 113 p.

[10] David Epstein, MS Paterson, JW Cannon, DF Holt, SV Levy, and William P Thurston, *Word processing in groups*, AK Peters, Ltd., 1992.

[11] D. Garber, D. Kahrobaei, and H. T. Lam, *Analyzing the Length-Based Attack on Polycyclic Groups*, (2013), `http://arxiv.org/abs/1305.0548`.

[12] D. Garber, S. Kaplan, M. Teicher, B. Tsaban, and U. Vishne, *Probabilistic solutions of equations in the braid group*, Advances in Applied Mathematics 35 (2005), 323–334.

[13] ———, *Length-based conjugacy search in the braid group*, Contemporary Mathematics 418 (2006), 75–87.

[14] Paul B Garrett, *Making, breaking codes: Introduction to cryptology*, Prentice Hall PTR, 2000.

[15] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.7.2*, `http://www.gap-system.org`.

[16] Maggie Habeeb, Delaram Kahrobaei, Charalambos Koupparis, and Vladimir Shpilrain, *Public key exchange using semidirect product of (semi) groups*, Applied Cryptography and Network Security, Springer, 2013, pp. 475–486.

[17] Marshall Hall, *The theory of groups*, vol. 288, American Mathematical Soc., 1976.

[18] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes, *Guide to elliptic curve cryptography*, Springer, 2004.

[19] KA Hirsch, *On infinite soluble groups (ii)*, Proceedings of the London Mathematical Society **2** (1938), no. 1, 336–344.

[20] ———, *On infinite soluble groupsi*, Proceedings of the London Mathematical Society **2** (1938), no. 1, 53–60.

[21] ———, *On infinite soluble groups (iii)*, Proceedings of the London Mathematical Society **2** (1946), no. 1, 184–194.

[22] ———, *On infinite soluble groups (iv)*, Journal of the London Mathematical Society **1** (1952), no. 1, 81–85.

[23] ———, *On infinite soluble groups (v)*, Journal of the London Mathematical Society **1** (1954), no. 2, 250–251.

[24] M. Hock and B. Tsaban, *Solving random equations in Garside groups using length functions*, Combinatorial and Geometric Group Theory (2010), 149–169.

[25] D. F. Holt, B. Eick, and E. A. O'Brien, *Handbook of computational group theory*, Chapman & Hall CRC, 2005.

[26] J. Hughes and A. Tannenbaum, *Length-based attacks for certain group based encryption rewriting systems*, Workshop SECI02 Securite de la Communication sur Internet (2002).

[27] D. Kahrobaei and B. Khan, *A non-commutative generalization of El-Gamal key exchange using polycyclic groups*, Proceedings of the Global Telecommunications Conference **4** (2006), no. 2.

[28] D. Kahrobaei and C. Koupparis, *Non-commutative digital signatures using non-commutative groups*, Groups, Complexity, Cryptology **4** (2012).

[29] D. Kahrobaei and H. T. Lam, *Heisenberg Groups as Platform for the AAG key-exchange protocol*, (2014), `http://arxiv.org/abs/1403.4165`.

[30] Delaram Kahrobaei, Charalambos Koupparis, and Vladimir Shpilrain, *Public key exchange using matrices over group rings*, Groups-Complexity-Cryptology **5** (2013), no. 1, 97–115.

[31] Delaram Kahrobaei, Ha T. Lam, and Vladimir Shpilrain, *Public key exchange using extensions by endomorphisms and matrices over a galois field*, (2014), Submitted.

[32] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. Kang, and C. Park, *New public-key cryptosystem using braid groups*, Advances in cryptology CRYPTO 2000 (Santa Barbara, CA), LNCS **1880** (2000), 166–183.

[33] Neal Koblitz, *A course in number theory and cryptography*, vol. 114, Springer, 1994.

[34] Roger C Lyndon, Paul E Schupp, RC Lyndon, and PE Schupp, *Combinatorial group theory*, vol. 19977, Springer-Verlag Berlin, 1977.

[35] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone, *Handbook of applied cryptography*, CRC press, 1996.

[36] KA Mihailova, *The occurrence problem for direct products of groups*, Dokl. Akad. Nauk SSSR **119** (1958), 1103–1105.

[37] J. Milnor, *Growth of nitely generated solvable groups*, Journal of Differential Geometry (1968), 447–449.

[38] A. Myasnikov, V. Shpilrain, and A. Ushakov, *Non-commutative cryptography and complexity of group-theoretic problems*, American Mathematical Society, 2011.

[39] A. D. Myasnikov and A. Ushakov, *Length based attack and braid groups: Cryptanalysis of Anshel-Anshel-Goldfeld key exchange protocol*, PKC 2007, LNCS 4450 (2007), 76–88.

[40] National Institute of Standards and Technology, *Digital Signature Standard, FIPS 186-4*, `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`, 2013.

[41] D. Ruinskiy, A. Shamir, and B. Tsaban, *Length-based cryptanalysis: the case of Thompsons group*, Journal of Mathematical Cryptology 1 (2007), 359–372.

[42] Bruce Schneier, *Applied cryptography: protocols, algorithms, and source code in c*, john wiley & sons, 2007.

[43] Dan Segal, *Polycyclic groups*, Cambridge Tracts in Mathematics, vol. 82, Cambridge University Press, 1983.

[44] V. Shpilrain and A. Ushakov, *Thompson's group and public key cryptography*, ACNS'05 Proceedings of the Third international conference on Applied Cryptography and Network Security (2005).

[45] Vladimir Shpilrain, *Assessing security of some group based cryptosystems*, Contemp. Math., Amer. Math. Soc **360** (2004), 167–177.

[46] Vladimir Shpilrain and Alexander Ushakov, *The conjugacy search problem in public key cryptography: unnecessary and insufficient*, Applicable Algebra in Engineering, Communication and Computing **17** (2006), no. 3-4, 285–289.

[47] Vladimir Shpilrain and Gabriel Zapata, *Using the subgroup membership search problem in public key cryptography*, Contemporary Mathematics **418** (2006), 169.

[48] I. Stewart and D. O. Tall, *Algebraic number theory and Fermat's last theorem*, AK Peters, 2002.

[49] Neal R Wagner and Marianne R Magyarik, *A public-key cryptosystem based on the word problem*, Advances in Cryptology, Springer, 1985, pp. 19–36.

[50] J. A. Wolf, *Growth of finitely generated solvable groups and curvature of Riemannian manifolds*, Journal of Differential Geometry (1968), 421–446.