4-2018

Implementation and Analysis of the Nonlinear Decomposition Attack on Polycyclic Groups

Yoongbok Lee

Implementation and Analysis of the Nonlinear Decomposition Attack on Polycyclic Groups

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science in Mathematics from
The College of William and Mary

by

Yoongbok Lee

Accepted for HONORS
(Honors, High Honors, Highest Honors)

Eric Swartz, Adviser

Ryan Vinroot

Xiaojing Liao

Williamsburg, VA
April 26, 2018

Implementation and Analysis of the Nonlinear Decomposition Attack on Polycyclic Groups

Yoongbok Lee

Abstract

Around two years ago, Roman'kov introduced a new type of attack called the non-linear decomposition attack on groups with solvable membership search problem. To analyze the precise efficiency of the algorithm, we implemented the algorithm on two protocols: semidirect product protocol and Ko-Lee protocol. Because polycyclic groups were suggested as possible platform groups in the semidirect product protocol and polycyclic groups have a solvable membership search problem, we used polycyclic groups as the platform group to test the attack. While the complexity could vary regarding many different factors within the group, there was always at least one exponential factor in the complexity analysis of the algorithm.

Acknowledgment

I acknowledge Professor Swartz for guiding me through the research and the committee members for their time and attention. Also, I would like to thank the Charles Center for supporting this research, and the donors who contributed to this research greatly.

Contents

Chapter 1

Public-key Cryptography

1.1 Key Exchange Protocol

Key exchange protocol is a way to share a secret key between two entities, namely Alice and Bob. Since the key is only known to Alice and Bob, they can now encrypt and decrypt messages with the key to make it hard for any adversary trying to obtain the secret message. For a simple example, given that the shared key K is encoded as a binary string of length n, Alice can compute $m_i \oplus K = e_i$ for each substring of m of length n ($|m_i| = n$). Then Bob, after receiving the encrypted ciphertext e, can compute $e_i \oplus K = (m_i \oplus K) \oplus K = m_i \oplus (K \oplus K) = m_i$ for each i to obtain the secret plaintext m.

Note that obtaining the value of the secret key K would imply that the adversary can perform the same operation to obtain the plaintext m. Thus, obtaining the key is equivalent to having an efficient way to break the cryptosystem. Even though there could be many different methods on how to make use of the shared key, we will focus on establishing the secret key assuming there is an effective use of the key to encrypt or decrypt messages.

1.2 Diffie-Hellman Key Establishment Protocol

The following description of Diffie-Hellman protocol is from the book *Group Based Cryptography* by Myasnikov, Shpilrain, and Ushakov [12], where more details can be found. From now on, we name the two sides of the key exchange protocol Alice and Bob, respectively.

1. Alice and Bob agree on a finite cyclic group G and a generating element $g \in G$.
2. Alice picks a random number $a \in \mathbb{N}$ and sends $g^a = \underbrace{g \cdot g \cdots g}_{a \text{ times}}$ to Bob.
3. Bob picks a random number $b \in \mathbb{N}$ and sends $g^b = \underbrace{g \cdot g \cdots g}_{b \text{ times}}$ to Alice.
4. Alice computes $K_{ab} = (g^b)^a = g^{ba} = g^{ab}$
5. Bob computes $K_{ba} = (g^a)^b = g^{ab}$

Thus we have the shared key $K = K_{ab} = K_{ba} = g^{ab}$. This protocol depends on the hardness of the *Diffie-Hellman Problem*, which is finding g^{ab} from g^a and g^b. Note that G and g have to be chosen carefully to avoid the adversary Eve from obtaining the shared key efficiently. The kind of problem that recovers a from g^a and g is called the *discrete logarithm problem*, and using brute force to search for the shared key requires time complexity of $O(|g|)$, where $|g|$ is the order of g.

1.3 Ko-Lee Protocol

The Ko-Lee protocol was first proposed by Ko, Lee, Cheon, Han, Kang, and Park [9] on braid groups to apply the Diffie-Hellman key exchange on non-commutative groups. Moreover, K.J. Gryak and D. Kahrobaei in [6] proposed that it could be a plausible scheme with a polycyclic group as its platform group (the group where the key exchange is performed), and the general procedure of the key exchange protocol they proposed is below.

1. A group G, an element $g \in G$, and subgroups $A, B \subseteq G$ such that $ab = ba$ for all $a \in A$ and $b \in B$ are made public.

2. Alice chooses an element $a \in A$ and then publishes g^a.

3. Bob chooses an element $b \in B$ and then publishes g^b.

4. Since A and B commute element-wise, Alice and Bob have the shared key $g^{ab} = g^{ba}$.

The *conjugacy search problem* is the following:
given a subgroup $A = \langle a_1, \ldots, a_q \rangle$ in G, an element $g \in G$, and an element g^a where $a \in A$, find an expression of a in terms of $a_1, \ldots, a_q$.
If an adversary can solve the conjugacy search problem, then he can break the system. Thus the secrecy of the above scheme depends on the hardness of the conjugacy search problem of the platform group.

1.4 ElGamal Cryptosystem

As in [**?**], the commonly used ElGamal encryption scheme (or the ElGamal Cryptosystem) is an encryption protocol based on the Diffie-Hellman key establishment protocol. The method goes as following:

1. Alice and Bob agree on a finite cyclic group G, and a generating element $g \in G$.

2. Alice, the receiver, picks a random natural number a as a private key.

3. Alice then publishes $g^a == \underbrace{g \cdot g \cdots g}_{a \text{ times}}$ as her public key.

4. Bob, who wants to send Alice a shared secret m, picks a random natural number b and computes $m \cdot (g^a)^b$.

5. Bob sends Alice two elements g^b and $m \cdot (g^a)^b$.

6. Alice recovers m by calculating $(m \cdot (g^a)^b)) \cdot ((g^b)^a)^{-1} = m$.

Using the common ElGamal cryptosystem that mainly depends on the discrete logarithm problem for its security, we can modify the exponents (the secret keys) of Alice and Bob to be the elements of the group. Kahrobaei and Khan [8] proposed a way to adapt the ElGamal key encryption algorithm for cryptosystems in non-commutative groups. The description of the scheme below is from the article by K.J. Gryak and D. Kahrobaei [6]. For this cryptosystem, we assume there is a group G and finitely generated subgroups $A, B \subseteq G$ such that A and B commute element-wise, meaning $ab = ba$ for all $a \in A$ and $b \in B$.

1. Bob chooses a random element $g \in G$. Then he picks his private key $b \in B$, and publishes g and $c = g^b = b^{-1}gb$.

2. To establish a shared secret $x \in G$, Alice chooses x and her secret key $a \in A$ then publishes $g^a = a^{-1}ga$ and $y = x^{c^a}$.

3. To recover the shared secret x, Bob computes $(g^a)^b$. Since the elements in A and B commute, we get

$$(g^a)^b = b^{-1}a^{-1}gab = a^{-1}b^{-1}gba = (g^b)^a = c^a \tag{1.1}$$

Then Bob can retrieve the shared secret x by computing $x = y^{(c^a)^{-1}}$. Note that the security of this protocol depends on the hardness of the conjugacy search problem of the platform group G.

1.5 Semidirect Product Protocol

The key exchange using semidirect product was proposed by Kahrobaei, Koupparis, and Shpilrain in [7]. Even though the general protocol seemed vulnerable to the linear decomposition attack [10], it was suggested that the dimension might grow too large for the attack to be efficient. Therefore, we discuss how the nonlinear decomposition attack could work against this protocol. The outline of the protocol is described below. Alice and Bob are going to work with following elements $\phi \in Aut(G)$, (ϕ^l, g) where $l \in \mathbb{N}$ and $g \in G$. The former element is multiplied as: $(\phi^p, g)(\phi^q, g) = (\phi^{p+q}, \phi^q(g) \cdot g)$.

1. Alice picks a private integer m then computes $(\phi, g)^m$
   $= (\phi^m, \phi^{m-1}(g) \dots \phi(g) \cdot g)$. Then she sends $g_m = \phi^{m-1}(g) \dots \phi(g) \cdot g$ to Bob.

2. Bob picks a private integer n then computes $(\phi, g)^n = (\phi^n, \phi^{n-1}(g) \dots \phi(g) \cdot g)$. Then she sends $g_n = \phi^{n-1}(g) \dots \phi(g) \cdot g$ to Alice.

3. Alice computes $(x, g_n) \cdot (\phi^m, g_m) = (x \cdot \phi^m, \phi^m(g_n) \cdot g_m)$. Then Alice's key is now $K_A = \phi^n(g_n) \cdot g_m$. Note that since Alice does not know n, so she does not know $\phi^n (:= x)$. Hence she does not calculate $x \cdot \phi^m$ where $x = \phi^n$, as she does not need it to calculate K_A.

4. Bob computes $(x, g_m) \cdot (\phi^n, g_n) = (y \cdot \phi^n, \phi^n(g_m) \cdot g_n)$. Then Bob's key is now $K_B = \phi^m(g_m) \cdot g_n$. Note that since Bob does not know m, so he does not know $\phi^m (:= y)$. Hence he does not calculate $y \cdot \phi^n$, as he does not need it to calculate K_B.

5. Alice and Bob now have the shared secret key

$$(\phi^n, g_n) \cdot (\phi^m, g_m) = (\phi^m, g_m) \cdot (\phi^n, g_n) = (\phi, g)^{m+n} = (\phi^{m+n}, g_{m+n}).$$

Chapter 2

Polycyclic Groups

The security of the above protocols (except the semidirect product based protocol) depend on the hardness of the conjugacy problem of the group. As polycyclic groups have no known efficient solutions to the conjugacy decision and search problems, it was suggested to fit as the platform group for those protocols. However, Roman'kov claimed in [14] that the groups in which the membership search problem is efficiently solvable can be vulnerable to the nonlinear decomposition attack. Since polycyclic groups have efficient solutions to the membership search problem, we used them as the platform groups to test the efficiency of the nonlinear decomposition attack. Theorems and definitions in this chapter are from the book "Computation with Finitely Presented Goups" by C.C. Sims [15]. More detailed proofs and background information can be found in the book as well.

2.1 Polycyclic Groups

All kinds of groups can be a possible candidate to be a platform group. However, finitely generated groups are most generally adopted, and finite groups are in practical use. In the case of polycyclic groups, it was shown in [15] that all polycyclic groups are finitely generated. Polycyclic groups can be infinite, but we mainly focused on

finite polycyclic groups to limit the computation time in the implementation and analysis stages.

2.1.1 Solvable and Nilpotent Groups

Before we discuss the properties of polycyclic groups, we need some background on solvable and nilpotent groups as the basis. The background information on solvable and nilpotent groups is in the appendix.

Definition 2.1.1. Define $[G, H] = \langle g^{-1}h^{-1}gh | g \in G, h \in H \rangle$. Then the ***derived subgroup of*** G is $G' = G^{(1)} = [G, G]$, and we recursively define $G^{(i)} = [G^{(i-1)}, G^{(i-1)}]$. The ***derived series of*** G is the sequence $G^{(0)} \supseteq G^{(1)} \supseteq \cdots$

Definition 2.1.2. Let $\gamma_1(G) = G$, and we also recursively define $\gamma_{i+1}(G) = [\gamma_i(G), G]$. Then the ***lower central series of*** G is the sequence $\gamma_1 \supseteq \gamma_2 \supseteq \cdots$

Definition 2.1.3. A ***solvable group*** is a group where $G^{(n)}$ is trivial for some n, and a ***nilpotent group*** is a group where γ_m is trivial for some m.

The smallest i (if it exists) such that $G^{(i)} = 1$ is called the ***derived length*** of G. The smallest j (if it exists) such that $\gamma_j(G) = 1$ is called the ***nilpotency class*** (or just simply ***class***) of G.

Proposition 2.1.4. Subgroups and quotient groups of solvable/nilpotent groups are solvable/nilpotent.

Proposition 2.1.5. If G is a group, then the group $G/G^{(i)}$ is solvable with derived length less than or equal to i. The group $G/\gamma_j(G)$ is nilpotent of class at most $j - 1$.

Proposition 2.1.6. If N is normal in G and both N and G/N are solvable, then G is solvable.

Proposition 2.1.7. Nilpotent groups are solvable.

Proof. Suppose $\gamma_i(G) = 1$. By Corollary A.1.11, we have j such that $G^{(j)} = 1$ where $2^j \geq i$. □

Proposition 2.1.8. If G/G' is generated by $x_1, \ldots, x_n$, then $\gamma_2(G)/\gamma_3(G)$ is generated by the images of $[x_j, x_i]$ with $1 \leq i < j \leq n$, under the natural homomorphism $f : G \to G/\gamma_3(G)$.

Proposition 2.1.9. Suppose G/G' is generated by a set X and $Y \subseteq \gamma_i(G)$ where $i \geq 2$ such that the image of Y under the natural homomorphism to $\gamma_i(G)/\gamma_{i+1}(G)$ generates that group. Then $\gamma_{i+1}(G)/\gamma_{i+2}(G)$ is generated by the image of

$$Z = \{[y, x] | y \in Y, x \in X\}.$$

Corollary 2.1.10. If G is generated by n elements and $i \geq 2$, then $\gamma_i(G)/\gamma_{i+1}(G)$ is generated by $(n-1)^{i-1}/2$ elements.

Proposition 2.1.11. If $N \leq Z(G)$ and G/N is nilpotent, then G is nilpotent.

2.1.2 Polycyclic Groups

Definition 2.1.12. A ***polycyclic series of length n*** of a group G is a sequence

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

where each group $G_i/G_{i+1}, 1 \leq i \leq n$, is cyclic. Any group with a polycyclic series is called a polycyclic group, and the group G above is said to have a polycyclic length of n.

Note that a group's polycyclic series can have different lengths since one can take different quotients for each i. However, the number of infinite quotients (where G_i/G_{i+1} is infinite) is the same for all polycyclic series. We call the number of infinite quotients the ***Hirsch number*** of the group G.

Proposition 2.1.13. Polycyclic groups are solvable.

Proposition 2.1.14. Finitely generated abelian groups are polycyclic.

Proposition 2.1.15. If N is normal in G and both N and G/N are polycyclic, then G is polycyclic.

Proposition 2.1.16. If G has a polycyclic series of length n, then G can be generated by n elements.

Proof. Suppose we have a polycyclic series $G = G_1 \supseteq G_2 \supseteq \cdots \supseteq G_{n+1} = 1$. Then let a_i $(1 \leq i \leq n)$ be an element in G_i such that $a_i G_{i+1}$ generates G_i/G_{i+1}. Then every coset of G_{i+1} in G_i contains a power of a_i. Then if $g \in G$, $g = a_1^{\alpha_1} g_2$, where $g_2 \in G_2$, and similarly $g_2 = a_2^{\alpha_2} g_3$, where $g_3 \in G_3$. So $g = a_1^{\alpha_1} a_2^{\alpha_2} g_3$. Proceeding inductively, we get $g = a_1^{\alpha_1} a_2^{\alpha_2} \dots a_n^{\alpha_n} g_{n+1}$ where $g_{n+1} = 1$. Therefore $g = a_1^{\alpha_1} a_2^{\alpha_2} \dots a_n^{\alpha_n}$, and thus G is generated by $a_1, a_2, \dots, a_n$. □

Remark 2.1.17. Note that by the above theorem, any element in G can be represented as $a_1^{\alpha_1} a_2^{\alpha_2} \dots a_n^{\alpha_n}$ $(\alpha_i \in \mathbb{Z})$.

Proposition 2.1.18. Quotient groups of polycyclic groups are polycyclic.

Proposition 2.1.19. Subgroups of polycyclic groups are polycyclic.

Corollary 2.1.20. If G has a polycyclic series of length n, then every subgroup of G can be generated by n or fewer elements.

2.2 Polycyclic Presentations

Definition 2.2.1. Suppose we have a group G and a polycyclic series $G = G_1 \supseteq G_2 \supseteq G_3 \supseteq \cdots \supseteq G_{n-1} \supseteq G_n \supseteq G_{n+1} = 1$. Then let a_i be an element of G_i whose image in G_i/G_{i+1} generates the group. Then the sequence $a_1, \dots, a_n$ is called the ***polycyclic generating sequence*** of the group G.

Definition 2.2.2. If $G = \langle a_i, \ldots, a_n \rangle$ as in Proposition 2.1.16, any element can be written in a form $a_1^{\alpha_1} \ldots a_n^{\alpha_n}$ where the α_j are integers. Let $I = I(a_1, \ldots, a_n)$ denote the set of subscripts i such that G_i/G_{i+1} is finite, and $m_i = |G_i : G_{i+1}|$, the order of a_i relative to G_{i+1}, if i is in I. Assuming no generating element is redundant (meaning no a_i is in G_{i+1}), we have $m_i > 1$ for each $i \in I$. Then an element $g \in G$ is in a ***collected word*** if $g = a_1^{\alpha_1} a_2^{\alpha_2} \ldots a_n^{\alpha_n}$ and each $0 \leq \alpha_j < m_j$ for all j, $1 \leq j \leq n$.

Then by the definition above, we can get relations

$$
\begin{aligned}
a_j a_i &= a_i a_{i+1}^{\alpha_{ij,i+1}} \cdots a_n^{\alpha_{ij,n}} \ ; \ j > i, \\
a_j^{-1} a_i &= a_i a_{i+1}^{\beta_{ij,i+1}} \cdots a_n^{\beta_{ij,n}}, \ j > i \ ; \ j \notin I, \\
a_j a_i^{-1} &= a_i^{-1} a_{i+1}^{\gamma_{ij,i+1}} \cdots a_n^{\gamma_{ij,n}}, \ j > i \ ; \ i \notin I, \\
a_j^{-1} a_i^{-1} &= a_i^{-1} a_{i+1}^{\delta_{ij,i+1}} \cdots a_n^{\delta_{ij,n}}, \ j > i \ ; \ i, j \notin I, \\
a_i^{m_i} &= a_{i+1}^{\mu_{i,i+1}} \cdots a_n^{\mu_{i,n}}, \ i \in I, \\
a_i^{-1} &= a_i^{m_i - 1} a_{i+1}^{\nu_{i,i+1}} \cdots a_n^{\nu_{i,n}}, \ i \in I,
\end{aligned}
$$

where the right sides are collected words in G. The above relations are called the ***standard polycyclic presentation*** relative to $a_1, \ldots, a_n$. If $i \in I$, since a_i^{-1} only appears once in the set of relations above, we can remove the relation by adding $a_i a_i^{-1} = 1$ on the relation when $i \notin I$ and by cancelling the left side by multiplying a_i on both sides when $i \in I$. Then the presentation after the operations is called the ***standard monoid polycyclic presentation***. Rewriting an element with respect to a standard polycyclic rewriting system is called ***collection***. Moreover, since the collected forms are unique, the rewriting system is said to be ***confluent***.

Proposition 2.2.3. Suppose G is defined by a polycyclic presentation with generators $a_1, \ldots, a_n$. Then G is polycyclic and $a_1, \ldots, a_n$ is a polycyclic generating sequence for G.

2.3 Membership Search Problem

The *membership search problem* is the following: Given a subgroup $H = \langle h_1, \ldots, h_t \rangle$ in G and an element $h \in H$, find an expression for h in terms of $h_1, \ldots, h_t$.

Definition 2.3.1. Suppose $G = \langle a_1, \ldots, a_n \rangle$. Then, in a generating sequence $U = (g_1, \ldots, g_s)$ of a subgroup H of G, let the *collected form* of g_i be $a_1^{\alpha_{i_1}} \ldots a_n^{\alpha_{i_n}}$

The s-by-n matrix A of the elements α_{ij} in i^{th} row and j^{th} column is called the *associated exponent matrix* of U. Also, we have the following elementary operations on exponent matrices:

1. Interchange g_i and g_j, where $i \neq j$.
2. Replace g_i with g_i^{-1}.
3. Replace g_i with $g_i g_j^{\alpha}$ where α is an integer and $i \neq j$.
4. Add a new row g_{s+1} any element of $\langle g_1, \ldots, g_s \rangle$
5. Remove g_k if $g_k = 1$

Note that none of the operations changes the subgroup $\langle g_1, \ldots, g_s \rangle$, and we say the sequences U and V are *equivalent under elementary operations* if one can be transformed into another with a sequence of above operations.

Example 2.3.2. Given a sequence $A = (g_1 g_2^4 g_3, g_1 g_3, g_2^6)$, we can construct the corresponding exponent matrix U associated with A

$$U = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 0 & 1 \\ 0 & 6 & 0 \end{bmatrix}$$

Definition 2.3.3. A sequence of elements $U = (g_1, \ldots, g_n)$ of G is said to be in ***standard form*** if the associated exponent matrix A satisfies the following properties.

- All rows of A are nonzero.
- A is row reduced over $\mathbb{Z}$.
- If A_{ij} is a corner entry and $j \in I$, then $A_{ij} | m_j$.

Example 2.3.4. Let G be the group generated by group elements

$$a_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$a_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given a generating sequence $U = (g_1, g_2, g_3)$, where

$$g_1 = a_1^2 a_2^{-1} a_4,$$
$$g_2 = a_3^3 a_4 a_6,$$
$$g_3 = a_4^2 a_5 a_6.$$

Then, U is in standard form with the associated exponent matrix

$$\begin{bmatrix} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 & 1 \end{bmatrix}.$$

An *admissible sequence of exponents* for U is a sequcnce $(\alpha_1, \ldots, \alpha_s)$ of integers such that if A_{ij} is a corner entry and $j \in I$, then $0 < \alpha_j \leq m_j / A_{ij}$. Let $E(U)$ denote the admissible sequence of exponents for U, and let $S(U)$ denote the set of products of $g_1^{\alpha_1} \ldots g_n^{\alpha_n}$ where $\alpha_1, \ldots, \alpha_n$ ranges over $E(U)$. $S(U)$ is generally not a subgroup of G, unless some conditions are satisfied which we will discuss later on in this paper.

Proposition 2.3.5. Suppose $U = (g_1, \ldots, g_s)$ is a sequence of elements in G in standard form, and $(\beta_1, \ldots, \beta_s), (\gamma_1, \ldots, \gamma_s) \in E(U)$. If $g_1^{\beta_1} \ldots g_s^{\beta_s} = g_1^{\gamma_1} \ldots g_s^{\gamma_s}$, then $\beta_i = \gamma_i$ where $1 \leq i \leq s$.

Proof. Let A be the exponent matrix associated with U, and let $g = g_1^{\beta_1} \ldots g_s^{\beta_s} = g_1^{\gamma_1} \ldots g_s^{\gamma_s}$. If A_{1j} is the corner entry of the first row and $a_1^{\delta_1} \ldots a_n^{\delta_n}$ is the collected word of g, then $\delta_k = 0$, where $1 \leq k < j$. Then we have two cases:

1. If $j \notin I$, then G_j/G_{j+1} is isomorphic to $\mathbb{Z}$, and hence $\delta_j = A_{1j}\beta_1 = A_{1j}\gamma_1$. Since $A_{1j} \neq 0$, $\beta_1 = \gamma_1$.

2. If $j \in I$, then G_j/G_{j+1} is isomorphic to $\mathbb{Z}_{m_j}$ and $\delta_j \equiv A_{1j}\beta_1 \equiv A_{1j}\gamma_1 \pmod{m_j}$. But both β_1 and γ_1 are nonnegative and less than m_j/A_{1j}, so $A_{1j}\beta_1$ and $A_{1j}\gamma_1$ are nonnegative and less than m_j. Therefore $A_{1j}\beta_1 = A_{1j}\gamma_1$ and $\beta_1 = \gamma_1$.

Thus, after the first step we can multiply $g_1^{-\beta_1}$ on both sides to get $g_2^{\beta_2} \ldots g_s^{\beta_s} = g_2^{\gamma_2} \ldots g_s^{\gamma_s}$. Continuing inductively, we get $\beta_i = \gamma_i$, where $1 \leq i \leq s$. □

The above procedure in the proof of Proposition 2.3.3 gives us an algorithm to solve the membership decision problem in $S(U)$ of G. Throughout the process, consider $h = a_1^{\beta_1} \ldots a_n^{\beta_n}$.

```
function POLY_MEMBER (U,g)
    #U = sequence (g_1,...,g_s) of elements of G in standard form
    #g = an element in G
    (the function returns true if g is in S(U) and false otherwise.
    A := exponent matrix of U;
    h := g;
    done := false;

    for i in [1..s] do
        if done break;
        A_{ij} := corner entry of A in i-th row;
        if some \beta_k != 0 for some (beta < j) then return false;
        if A_{ij} does not divide \beta_j return false;

        q := \beta_j / A_{ij};
```

```
        h := g_i^{-q}h;
    od;
    return (h = 1);
end;
```

Definition 2.3.6. Given a sequence $U = (g_1, \ldots, g_n)$ of elements of G in standard form, U is *full* if it satisfies the following conditions:

1. For $1 \leq i < j \leq s$ the set $S(U)$ contains $g_i^{-1} g_j g_i$.

2. Let A be the associated exponent matrix of U. If A_{ij} is a corner entry and $j \in I$, then $S(U)$ contains g_i^q where $q = m_j / A_{ij}$

Example 2.3.7. Consider the sequence U from Example 2.3.4. To make the sequence full, we need to add the elements in the conditions above and iterate through the row reduction process. The resulting associated exponent matrix of the full sequence is

$$A = \begin{bmatrix} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Now we can decide whether a sequence U and its acceptable set of exponents $S(U)$ is a subgroup of G.

Proposition 2.3.8. If $U = (g_1, \ldots, g_s)$ is a sequence of elements of G in standard form, then $S(U)$ is a subgroup of G if and only if U is full. If U is full, then the sequence $g_1, \ldots, g_s$ is the polycyclic generating sequence of $S(U)$.

Proposition 2.3.9. If H is a subgroup of G, then there is a unique sequence $U = (g_1, \ldots, g_n)$ such that $H = S(U)$.

Now, suppose we have a subgroup H of G, generated by the sequence $U = (h_1, \ldots, h_s)$. If we could transform U into standard form with a deterministic algorithm, then we can solve the membership search problem on H using the above algorithm POLY_MEMBER.

We can first apply elementary row operations on the associated matrix A of U to reduce it to row echelon form. Suppose h_i and h_j have the leading term a_k^β and a_k^γ, with $|\beta| > |\gamma|$. Replacing h_i with $h_i h_j^{-q}$ where $q = \lfloor \beta/\gamma \rfloor$, we can change the exponent of the leading term a_k of h_i to $\beta \mod \gamma$. Repeating this step until it is impossible, one can obtain an exponent matrix A to have no two elements having the leading terms involving the same generator. Then we can switch rows and delete zero rows (identity element) from A and assume that A is in row echelon form. (Also, one can replace h_i by h_i^{-1} to make all the leading terms positive.)

Now we transform the row echelon matrix A into a full exponent matrix. Suppose there is a corner entry A_{ij} such that $j \in I$ and A_{ij} does not divide m_j. Let $\alpha = gcd(A_{ij}, m_k) = pA_{ij} + qm_k$ and add h_i^p to U as a new element. Then, repeat the above process to get A back into row echelon form. Repeating this procedure for all corner entries of A, it now satisfies the second requirement of being full.

In order to satisfy the first requirement, we test, for each $i, j \in \{1, \ldots, s\}$, whether $h_i h_j h_i^{-1} \in S(U)$. If the POLY_MEMBER returns false, let u be the last value assigned to h before the function terminated. Add it to U, then repeat the above procedure from reducing to the row echelon form. The resulting matrix A is the associated exponent matrix of V, the polycyclic generating sequence of H. Then, we can apply the POLY_MEMBER function to solve the membership search problem on the subgroup H.

Chapter 3

The Nonlinear Decomposition Attack

Roman'kov [14] suggested the Nonlinear Decomposition Attack as the supplement of the Linear Decomposition Attack [10], when the platform group does not admit a faithful linear representation. The Nonlinear Decomposition Attack depends heavily on efficiently solving the membership search problem on the platform group to decompose the secret private key into known elements. In our case, the membership search problem in polycyclic groups is efficiently solvable [15].

There are three major components for general implementation for the nonlinear decomposition attack. First we need to find the generating set in which either one of the two public keys are stored. Then, we solve the membership search problem for the chosen public key in terms of those generators using properties of polycyclic groups. Finally, we obtain the shared key using the public key, which is a word of the generating set. Below, the claims and proofs from the first section "Finding the Shared Key" are from [14].

3.1 Finding the Shared Key

Roman'kov proposed a deterministic method of finding a generating set for a public key. We will analyze the general time complexity on the presented calculations in later sections. For example, in the protocol proposed by Ko, Lee et al., [9] Alice's public key is g^a, where g is public. Then, we can generate $g^A := \langle g^a \mid g^a = a^{-1}ga,\ a \in A\rangle$.

3.1.1 Semidirect Product Protocol

While discussing the semidirect-product protocol, denote

$$g_i = \phi^{i-1}(g) \cdot \phi^{i-2}(g) \dots \phi(g) \cdot g.$$

Lemma 3.1.1. Let G be a group, g an element in G, and ϕ an endomorphism on G. Then define $H = \langle g_i \mid i \in \mathbb{N}\rangle$ and $g_i = \phi^{i-1}(g) \cdot \phi^{i-2}(g) \dots \phi(g) \cdot g$. Assuming that H is finitely generated and the membership decision problem is solvable in G, there is an algorithm to find a finite generating set of H in terms of the g_i's.

Proof. Note that $\langle g, g_1, \dots, g_n\rangle = \langle g, \phi(g), \dots, \phi^{n-1}(g)\rangle$, since one can apply $g_i \cdot (g_{i-1})^{-1} = (\phi^{i-1}(g) \dots \phi(g) \cdot g) \cdot (g^{-1} \cdot (\phi(g))^{-1} \dots (\phi^{i-2}(g))^{-1}) = \phi^{i-1}(g)$ for any $i \in \mathbb{N}$.

For any $l \in \mathbb{N}$, denote $H_l = \langle g_0, \dots, g_{l-1}\rangle = \langle g, \phi(g), \dots, \phi^{l-1}(g)\rangle$. By the assumptions above, we can efficiently calculate the minimum number k such that $g_{k+1} \in H_k$, meaning $H_{k+1} = H_k$. Thus, there is a group word w of $H_k = \langle g, \phi(g), \dots, \phi^{k-1}(g)\rangle$ such that $w = \phi^k(g)$. Then it follows that

$$\begin{aligned} \phi^{k+1}(g) &= u(\phi(g), \dots, \phi^{k-1}(g), \phi^k(g)) \\ &= u(\phi(g), \dots, \phi^{k-1}(g), w(g, \phi(g), \dots, \phi^{k-1}(g))) \\ &= v(g, \phi(g), \dots, \phi^{k-1}(g)) \end{aligned} \tag{3.1}$$

for some other group word $v(g, \phi(g), \ldots, \phi^{k-1}(g)) \in H_k$. Therefore $H_{k+n} \in H_k$ for all $n \in \mathbb{N}$, and the subgroup stabilizes after a finite number of steps. Then, we have the generating set $\{g, \phi(g), \ldots, \phi^{k-1}(g)\}$ of H_k. □

Now we have either Alice's or Bob's public key (suppose from now we have Alice's) as a word in $(g, g_1, \ldots, g_k)$. Letting $h_i = g_{i-1}$ where $g_0 = g$, the subgroup H_k above is equal to $\langle h_1, \ldots, h_k \rangle$. We have Alice's (or Bob's) public key as a word in $(g_1, \ldots, g_k)$:

$$g_m = u(h_1, \ldots, h_k) = \textstyle\prod_{j=1}^{k} g_{i_j}^{\epsilon_j}, \text{ where}$$
$$i_j \in \{i, \ldots, k\}, \epsilon_j \in \{1, -1\}, j \in \{1, \ldots, k\}, k \in \mathbb{N}.$$

Then since $\phi^{i_j}(g_n) \cdot g_{i_j} = \phi^n(g_{i_j}) \cdot g_n$,

$$K = g_{m+n} = \phi^n(g_m) \cdot g_n = \phi^n(\textstyle\prod_{j=1}^{k} g_{i_j}^{\epsilon_j}) \cdot g_n = \prod_{j=1}^{k} \phi^n(g_{i_j})^{\epsilon_j} \cdot g_n =$$
$$\textstyle\prod_{j=1}^{k} (\phi^n(g_{i_j} \cdot g_n \cdot g_n^{-1}))^{\epsilon_j} \cdot g_n = \prod_{j=1}^{k} (\phi^{i_j}(g_n) \cdot g_{i_j} \cdot g_n^{-1})^{\epsilon_j} \cdot g_n,$$

where all elements g_n, g_{i_j}, ϵ_j, and ϕ^{i_j} for $i_j \in \{1, \ldots, k\}$ are known. Thus, we have the shared key without solving the underlying problem of finding $n \in \mathbb{N}$ such that $g_n = g_m$ of the platform group.

3.1.2 Ko-Lee Protocol

Lemma 3.1.2. Suppose G is a group and $g \in G$. Assuming the word and membership search problems are solvable in G and all subgroups are finitely generated in G, there is an algorithm that finds the generating set of the subgroup $g^A = \langle g^a \mid a \in A \rangle$.

Proof. One can construct an algorithm that eventually terminates to obtain the generating set, similar to the semidirect product protocol.
Let $L_0 = g$, the known public element. Then let $M_i = \{a \mid a \in A\}$, where a is a word in $A = \langle a_1, \ldots, a_k \rangle$ of length i. Then, $M_1 = \{a_1, a_2, \ldots, a_k, a_1^{-1}, a_2^{-1}, \ldots, a_k^{-1}\} = \{m_1, \ldots, m_{2k}\}$. The algorithm proceeds as follows:

1. Set some order among the elements in M_1.

2. Iterating in order, test whether each g^{m_j} is in the group $\langle g, g^{m_1}, \ldots, g^{m_{j-1}} \rangle$.

3. After iterating through all elements in M_1, one can obtain the subset $L_1 = \{g, g^{m_{j_1}}, \ldots, g^{m_{j_l}}\}$ such that $g^{m_{j_i}} \notin \langle g, g^{m_{j_1}}, \ldots, g^{m_{j_{i-1}}} \rangle$

4. If $\langle L_0 \rangle \neq \langle L_1 \rangle$, proceed to $i = 2$

Note that $g^A = \langle g^a \mid a \in A \rangle = \langle g \cup \bigcup_{i=1}^{\infty} M_i \rangle$. Moreover, since we assume that all subgroups of G are finitely generated, the process terminates for some $n \in \mathbb{N}$, and we have the generating set L_n such that $\langle L_n \rangle = g^A$. □

Then, suppose we have $g^A = \langle g^{c_1}, \ldots, g^{c_s} \rangle$. Since we assumed that the membership search problem is solvable, we can obtain $g^a = w(g^{c_1}, \ldots, g^{c_s})$, where w is a group word.

Note that since A and B commute element-wise, $bc_i = c_i b$ for all $1 \leq i \leq s$ and $b \in B$, so we have

$$w((g^b)^{c_1}, \ldots, (g^b)^{c_s}) = w(g^{c_1}, \ldots, g^{c_s})^b = g^{ab} = K.$$

Thus we obtain the shared key without solving the conjugacy search problem of the platform group.

3.2 Implementation

As mentioned above, the nonlinear decomposition attack is divided up into three major parts:

- Find the generating set of a public key.
- Solve the membership search problem of the public key within the subgroup.
- Obtain the key from the public key as a word of the generating set of the subgroup.

As the membership search problem is described above, we describe finding the generating set and obtaining the solution in pseudocode to visualize the logistics.

3.2.1 Ko-Lee Protocol

```
find_generating_set := function(G,A,g);

    input
    G : public platform group G
    A : public subgroup of G
    g : public element g

    gen_list := [g];                 # L_0 in the lemma 3.1.2
    exponent_list := [Identity(G)]; #exponent list to keep track of c_i's
    i := 1;                         # counter for number of iterations.
    gen_set_A := Gen(A);             # known polycyclic generating set of A

    extended := true;
    while extended do
        prev_subgroup := Subgroup(G,gen_list);

        # since we only deal with finite groups, we have a_i^{-1} = a^m for
        # some positive m.therefore, we only have a^i elements to test for
        # i-th iteration, where a is the number of generating elements of A.

        for j in [1..a^i] do
            result_word := Identity(G);
            result := Identity(G);

            # generate words of length i
            for k in [1..i] do
                new_index := ((j mod Length(gen_set_A)^(k+1)) -
                    (j mod Length(gen_set_A)^k))
                    / Length(gen_set_A)^k;
                new_letter := gen_set_A[new_index+1];
                result := new_letter*result;
            od;

            # test whether it extends the subgroup
            if not (g^result in Subgroup(G, gen_list)) then
                Add(gen_list,g^result);
                Add(exponent_list,result);
            fi;

        od;
```

```
        # if the last iteration did not extend the group, break.
        if prev_subgroup = Subgroup(G, gen_list) then
            extended := false;
        fi;
        i := i + 1;
    od;

    return [gen_list,exponent_list];
end;
```

Note that we keep track of the exponents c_i's of the generating sequence, since we need to calculate a group word in terms of $(g^b)^{c_i}$'s later when we arrive at the shared key.

```
find_key := function(G,gen_set,gm_in_terms_of_gen_set,gn);

    input
    G          : public platform group G
    gen_set    : generating set (including the exponents) of A
    gm         : Alice's key in terms of A's original generating set
    gn         : public key of Bob

    result := Identity(G);

    # K = w((g_b)^c_1,...,(g_b)^c_s)
    # gen_set and gm are lists with specified formatting.
    for i in [1..Length(gm)] do
            result := result * gn^(gen_set[2][gm[i]]);
    od;
    return result;
end;
```

3.2.2 Semidirect Product Protocol

The method of finding the generating set of the subgroup containing the public keys is not much different than the one used in the Ko-Lee Protocol.

```
find_generating_set := function(G,endomorphism,g);

    input
    G                  : the public group G
    endomorphism       : known endomorphism \phi : G -> G
```

```
    g                 : known public element in G

    result_gen_list := [g];
    add_gen := Image(endomorphism,g);
    cur_gen := add_gen*g;
    subgroup := Subgroup(G, result_gen_list);

    # While the group is extended by g_{i-1}, proceed to g_i
    while not cur_gen in subgroup do
        Add(result_gen_list,cur_gen);
        subgroup := Subgroup(G, result_gen_list);
        add_gen := Image(endomorphism,add_gen);
        cur_gen := add_gen*cur_gen;
    od;

    return result_gen_list;
end;

find_key := function(G,endomorphism,gm_in_gen_set,gn);

    input
    G                 : public group G
    endomorphism      : known endomorphism in G
    gm_in_gen_set     : Alice's public key in terms of the gen_set
    gn                 : Bob's public key

    result := Identity(G);
    for i in [1..Length(gm_in_gen_set)] do
        left := gn;

        # \phi^{i_j}(g_n)
        for j in [1..gm_in_gen_set[i]] do
            left := Image(endomorphism,leftPart);
        od;

        #the rest of multiplication in the iteration
        result := result * (left * gen_set[gm_in_gen_set[i]] * gn^-1);
    od;

    result := result * gn;
    return result;
end;
```

Note that when the group is finite, negative exponents can be regarded as $(-1 \mod n)$, where n is an integer.

3.3 Time Complexity Analysis

3.3.1 Membership Search Problem

To analyze the efficiency of the nonlinear decomposition attack, one can look at the worst-case time complexity analysis. Since there are three independent parts in the nonlinear decomposition attack, we only need to consider the process which requires the highest complexity. In the analysis we assume G to be a finite polycyclic group. Consider finding the generating set in the presented protocols, Ko-Lee protocol and semidirect product protocol. We first consider the complexity of the membership search problem, then apply it to each protocol to find the general time complexity of those protocols. We follow the algorithm of solving the membership search problem in Chapter 2.

First, note that every row operation on the associated exponent matrix requires a proper collection process. As M.F. Newman and A.C. Niemeyer proposed in [13], the upper bound of the collection algorithm on G with derived length d and maximum normal word length N is N^{3d-1}, using collection from the left.

Lemma 3.3.1. Let $G = G_1 \supseteq G_2 \supseteq G_3 \supseteq \cdots \supseteq G_{n-1} \supseteq G_n \supseteq G_{n+1} = 1$ be a polycyclic series of G, with $(g_1, \ldots, g_n)$ the polycyclic generating sequence of G. Assume U is the generating sequence of H and A the associated exponent matrix of U. Suppose the subgroup H is generated by s elements. Then the total time complexity of solving the membership search problem on H is $O(n^5 b(\log_\varphi(b))^2)$

Proof. The row reduction process is essentially the Euclidean algorithm applied to the row which has the same generating element as its leading term on the associated exponent matrix. By [2], the worst case time complexity of Euclid's algorithm is $\log_\varphi(a)$ where a is the larger exponent and $\varphi = (1+\sqrt{5})/2$ is the golden ratio. Let b by the upper bound over all exponents of a collected word in G. Since G can be generated by n elements, the subgroup of G can have at most n polycyclic generating elements

by Corollary 2.1.20. Therefore, there can be at most n^2 Euclidean Algorithms of complexity $\log_\varphi(b)$ applied during the row operations, so the time complexity of the row reduction process is $O(n^2 \log_\varphi(b))$. Then, permuting the rows and replacing the corner entries with positive terms take at most $O(n\log(n) + n) = O(n\log(n))$ steps, thus the overall complexity remains the same.

Next, we need to make all the corner entries divide m_k, the order of the cyclic group G_k/G_{k+1}. (Since we assume the group to be finite, there exist such m_i's for all $i \in \{1, \ldots, n\}$.) Since we again use the extended Euclidean algorithm to calculate $\beta = \gcd(A_{ij}, m_k) = pA_{ij} + qm_k$ where A_{ij} is a corner entry, its time complexity is $O(\log_\varphi(b))$. Then as we add the resulting element exponent vector to the exponent matrix and repeat the above procedure, we have $O(n^2 \log_\varphi(b) \cdot \log_\varphi(b)) = O(n^2(\log_\varphi(b))^2)$.

Then, we need the sequence U to be full. Since there are at most n^2 elements on which to test the first condition (for $1 \leq i < j \leq s$ the set $S(U)$ contains $g_i^{-1}g_jg_i$) and n elements on which to test the second condition (if A_{ij} is a corner entry and $j \in I$, then $g_i^q \in S(U)$ where $i \in \{1, \ldots, n\}$ and $q = m_j/A_{ij}$), and each test is done by the POLY_MEMBER algorithm. The definition of *admissible sequence of exponents* of U, denoted $E(U)$, is the sequence $(\beta_1, \ldots, \beta_s)$ of integers such that if A_{ij} is a corner entry, then $0 \leq b_i < m_j/A_{ij}$, if $i \in I$. So, the process to acquire $h = 1$ (h is the tested element after cancellation during POLY_MEMBER) takes at most $n(b-1)$ cancellations. Thus the time complexity of making U full is $O(n^2(nb) + n(nb)) = O(n^3b)$. Moreover, if we add another element not in $S(U)$, we iterate through the row reduction process again. Therefore the total time complexity would be $O(n^3b \cdot n^2(\log_\varphi(b))^2) = O(n^5b(\log_\varphi(b))^2)$. □

Since the above procedures are piecewise independent, the total time complexity of solving the membership search problem is $O(n^5b(\log_\varphi(b))^2)$.

3.3.2 Ko-Lee Protocol

We need to analyze two steps: finding the generating set of the subgroup g^A (or g^B, but assume we are searching for the subgroup containing Alice's public key) and finding the shared key.

Claim 3.3.2. Suppose $A = \{1\}$, and suppose we add successively other elements $g \notin \langle A \rangle$ until $\langle A \rangle = G$. Let $|G| = p_1^{\alpha_1} \cdots p_n^{\alpha_n}$, where p_i's are distinct primes. Then the process takes at most $\alpha := \sum_{i=1}^{n} \alpha_i$ iterations.

Proof. Let $|G_i/G_{i+1}| = p_1^{\beta_{i_1}} \cdots p_n^{\beta_{i_n}}$ be the prime factorization of the order of G_i/G_{i+1}. Then, suppose the new element is in G_j, such that its image of the natural mapping to G_j/G_{j+1} is not the identity. Also, the order of the image in G_j/G_{j+1} is some multiple of those primes $p_1^{\gamma_{j_1}} \cdots p_n^{\gamma_{j_u}}$ where $\gamma_{j_l} \leq \beta_{j_l}$. Then consider the process on G_j/G_{j+1}: Consider the multiset $X = \{p_1^{\gamma_{j_1}}, p_2^{\gamma_{j_2}}, \ldots, p_n^{\gamma_{j_n}}\}$ which means X contains p_i exactly γ_{j_i} times, and $\gamma_j = \sum_{i=1}^{n} \gamma_{j_i}$

```
z = 1; #z \in G_j/G_{j+1}
gen_set = []; #the obtained generating set of G_j/G_{j+1}
for i in [1..\gamma_j] do
    x = Random(X);
    X = X - x; # take one element out of X
    z = z * x; # multiply z by x
    g = element_of_order z; take element of order z in G_j/G_{j+1}
    Add(gen_set, z); # add g to gen_set
od;
```

Since the group is cyclic, any multiplication of the elements in the subgroup generated by *gen_set* will have order of the least common multiple of those elements, bounded above by the order of the last element added to the subgroup. Therefore, for all iterations, because the group is cyclic, none of the new elements are in the previous

generating set. If the new g is in the subgroup generated by the previous set of elements, then we arrive at a contradiction, having an element of order greater than the upper bound provided above. Moreover, the process cannot take more than $\sum_{i=1}^{n} \beta_{j_i}$ steps, since then it means there is at least one step where we add an element of order equal to another element's order by the pigeonhole principle. Since the group is cyclic, this means that we added an element already in the group generated by the previous elements, which contradicts our assumption.

Thus, this process takes $\sum_{i=1}^{n} \gamma_{j_i}$ iterations for the image of the elements in the subgroup to fully contain G_j/G_{j+1}. Moreover, $\sum_{i=1}^{n} \gamma_{j_i} = \beta_j$, and $\sum_{i=1}^{n} \sum_{j=1}^{n} \beta_{i_j} = \sum_{i=1}^{n} \alpha_i = \alpha$. Therefore, the maximum iterations required for the images to fully contain every coset $G_i/G_{i+1}, i \in \{1, \ldots, n\}$ is α. Therefore, it requires at most α iterations for the process to terminate. □

In finding the generating set of g^A, we first set $g^{A_0} = \{g\}$, and test for each iteration starting from $i = 1$ whether g^{A_i} extends L_{i-1}, where $g^{A_i} = \{g^a |$ the word length of $a \leq i\}$. Each iteration takes $(2a)^i$ membership decision tests, where $|A| = a$. So the total number of subgroup membership search tests is bounded above by $\sum_{i=1}^{\alpha+1} (2a)^i$ where α is the total number of prime factors of $|G|$ as in Claim 3.3.2 (since $a \leq n$, the upper bound becomes $\sum_{i=1}^{\alpha+1} (2a)^i$). Also, each subgroup test has time complexity of $O(n^5 b(\log_\varphi(b))^2)$ as well as $2i$ multiplications for each α^i iterations on i^{th} iteration. The total time complexity of finding the generating set is

$$O(n^5 b(\log_\varphi(b))^2 (\sum_{i=1}^{\alpha+1} 2^i n^i N^{3d-1})) = O(n^5 b(\log_\varphi(b))^2 \cdot 2^{\alpha+1} n^{(\alpha+1)} N^{3d-1})$$
$$= O(2^{\alpha+1} n^{5+(\alpha+1)} b(\log_\varphi(b))^2 N^{3d-1}).$$

Now, as we are going to solve the membership search problem in a different generating sequence than the original sequence, in Ko-Lee protocol, we need to keep track of each polycyclic generating sequence in terms of the original generating set (to calculate

the shared key). Therefore, in order to analyze the time complexity of the nonlinear decomposition attack, we need the maximum word length of an element $g^a \in g^A$ in terms of the original generating set $(g^{c_1}, \ldots, g^{c_s})$.

Claim 3.3.3. Suppose we have generating sequence (not necessarily polycyclic) $V = (h_1, \ldots, h_s)$ of H. Then the maximum length of any element $h \in H$ in terms of those element by converting the sequence and applying the membership search test is $(n^2 + n) \cdot b^{n \log_\varphi(b)}$.

Proof. Noting that there are only additions of $g_i^{-1} g_j g_i$'s and g_i^p's while making the sequence full, during the POLY_MEMBER algorithm, the elementary row operation that makes the resulting sequence the longest in terms of the original generating sequence is replacing g_i with $g_i g_j^\beta$ where β is an integer. In our case, there are a maximum of n generating elements, and for each step in row reduction, we have an upper bound of $\log_\varphi(b)$ operations, and each time the β is bounded above by b. Thus each row reduction process increases the length of the word in terms of the original generating set of the element represented by the row in the exponent matrix by $b^{\log_\varphi(b)} \cdot l_i + l_j$, where l_i, l_j are the word lengths of the elements represented by the i^{th} and j^{th} rows in terms of the original generating set before the row operation.
Now, consider the last row of the exponent matrix A. Denoting the word length of the i^{th} row as l_i, the $(i+1)^{st}$ row of A will have length $b^{\log_\varphi(b)} \cdot l_i + l_{i+1}$, by induction. Thus the last (n^{th}) row will have length at most $b^{n \log_\varphi(b)} + l_n = b^{n \log_\varphi(b)} + 1 \approx b^{n \log_\varphi(b)}$. Then, we ensure each corner entry A_{ij} divides m_j by multiplying integers, bound by b. To finalize and make A full, we apply the POLY_MEMBER algorithm of $n^2 + n$ elements on $S(U)$, and in the worst case, $n^2 + n$ elements will be added to the sequence V over the process. As a result, there will be $n^2 + n$ additional row reduction iterations. Therefore, the upper bound of the word length of an element in the generating sequence in terms of the original generating set is $(n^2+n) \cdot b^{n \log_\varphi(b)}$. □

(We noticed during the test cases, even for fairly small b's, that putting the words in terms of the generating set often required large amount of memory and was a severe restriction on testing the algorithm.)

Finally, we need to calculate the shared key. The shared key calculation is fairly simple, as it only needs to calculate the conjugation $(g^b)^{c_i}$, where $\langle g^{c_i} \rangle = g^A$. The maximum number of generating element for g^A is α, and each conjugation is two multiplications, which is equivalent to two collection processes. Also, the maximum word length of $a \in A$ in terms of g^{c_i} is, by Claim 3.3.3, $(n^2+n) \cdot b^{n \log_\varphi(b)}$, so two collections for each g^{c_i} in the word. Therefore we need $2 \cdot (n^2+n) \cdot b^{n \log_\varphi(b)}$ collection processes, which equals to $2 \cdot (n^2+n) \cdot b^{n \log_\varphi(b)} \cdot N^{3d-1} = O(n^2 b^{n \log_\varphi(b)} N^{3d-1})$.

In summation, the overall attack has time complexity of

$$\begin{aligned} &O(2^{\alpha+1} n^{5+(\alpha+1)} b (\log_\varphi(b))^2 N^{3d-1} + (n^2+n) \cdot b^{n \log_\varphi(b)} + n^2 b^{n \log_\varphi(b)} N^{3d-1}) \\ &= O(2^{\alpha+1} n^{5+(\alpha+1)} b (\log_\varphi(b))^2 N^{3d-1} + n^2 b^{n \log_\varphi(b)} N^{3d-1}). \end{aligned}$$

3.3.3 Semidirect Product Based Protocol

On this protocol, note that $g_i = \phi^{i-1}(g) \cdot \cdots \cdot \phi(g) \cdot g$ By Claim 3.3.2, we need a maximum of $\sum_{i=1}^{n} \alpha_i = \alpha$ iterations, provided that $|G| = p_1^{\alpha_1} \cdots p_n^{\alpha_n}$, to obtain the generating set of the subgroup containing Alice and Bob's key. Note that since the public endomorphism ϕ is operation-preserving, we have $\phi(g) = \phi(u(g_1, \ldots, g_n)) = u(\phi(g_1), \ldots, \phi(g_n))$. Also, the generating elements g_i's can be mapped to any element with word length bounded above by nb, and the word length of all $\phi^j(g)$'s are bounded above by N, the maximum normal word length. In each step, the maximum length of the image of the public element $\phi^i(g)$ after the endomorphism ϕ is bounded above by N^2. To collect the elements, we need to apply N collections, each taking at most N^{3d-1} steps.

The i^{th} iteration has some basic steps:

- Calculate $\phi^{i-1}(g)$. This can be done efficiently by applying the endomorphism once on $\phi^{i-2}(g)$ from the previous step.
- Multiply the calculated $\phi^{i-1}(g)$ by the previously calculated g_{i-1} to obtain g_i.
- Apply the subgroup test to see if g_i extends the subgroup generated by $\{g, g_1, \ldots, g_{i-1}\}$.
- If the subgroup is extended, proceed to $(i+1)^{st}$ iteration.

First, we have $\phi^{i-2}(g)$ already collected. Therefore, we need to compute $N \cdot N^{3d-1}$ to get $\phi^{i-1}(g)$ in a collected form. Then we multiply $\phi^{i-1}(g)$ and g_{i-1}, taking at most N^{3d-1} steps, and apply POLY_MEMBER on the resulting element g_i, which takes $O(n^5 b(\log_\varphi(b))^2)$ time. Then we add it to the subgroup, depending on whether it extends the group. The total time complexity of each step is then bounded above by $(1+N)N^{3d-1} + n^5 b(\log_\varphi(b))^2$. Therefore, the number of steps of the whole procedure is bounded above by $\alpha((1+N)N^{3d-1} + n^5 b(\log_\varphi(b))^2)$.

After solving the membership search problem for Alice's key g_m, we calculate the shared key by

$$\prod_{j=1}^{k} (\phi^{i_j}(g_n) \cdot g_{i_j} \cdot g_n^{-1})^{\epsilon_j} \cdot g_n,$$

where $g_m = \prod_{j=1}^{k} g_{i_j}^{\epsilon_j}$.

The number i_j is bounded above by α, the number of iterations taken to find the generating set. Since we can calculate and store $\phi^{i_j}(g)$'s, it takes $\alpha \cdot (1+N)N^{3d-1}$ to calculate and store those elements. Then since the upper bound of word length in the subgroup $H = \langle g_i | i \in \{1, \ldots, \alpha\} \rangle$ is bounded above by $(n^2+n)b^{n \log_\varphi(b)}$, we have $k \leq (n^2+n)b^{n \log_\varphi(b)}$. Therefore the complexity of finding the shared key for the semidirect product protocol is bounded above by $\alpha((1+N)N^{3d-1} + n^5 b(\log_\varphi(b))^2)) + (n^2 + n)b^{n \log_\varphi(b)}(2N^{3d-1}) + N^{3d-1} = (\alpha(1+N) + 2(n^2+n)b^{n \log_\varphi(b)} + 1)N^{3d-1} + \alpha n^5 b(\log_\varphi(b))^2)$.

3.4 Test Results

The test results were done in a linux OpenSUSE machine, Dell Optiplex 9020 with an Intel Core i7 processor and 16 GB of RAM .

Ko-Lee protocol was used as the testing. The platform group G was $P \operatorname{wr} C_2$, where P was a Heisenberg Group of order p^3 and $\mathbb{Z}_p$ as the field for each entry.

The time measurements are average time in 100 trials with random public element g, Alice's public key, and Bob's public key to calculate the shared key.

p	Alice and Bob	Eve
5	∼0(ms)	15.92(ms)
7	∼0	38.56
11	∼0	719.44
13	∼0	5112.64 (5 sec)
17	∼0	11982.76
19	∼0	23941.04
23	∼1/4	113164.88 (2 min)
29	∼1/4	907250.44
31	∼1/4	626605.6
37	∼1/4	3090570.68(45 min)
41	∼1/2	3304778.24 (50 min)

Note that the computation time for Eve, the adversary, does not consistently grow in accordance with p, and the group order $|G|$, most apparent between $p = 29$ and $p = 31$. The reason was quite unclear, but we noticed that for a few of the test cases the computation time were much longer than the average of all test cases. Observing the relations among the elements a, b, and g could be the way to find the source of the difference, but it was not apparent why there were so much gap between some test cases and the others.

Chapter 4

Future Research

4.1 More Platform Groups

Even though we only tested for some specific polycyclic groups, the process of implementing and testing the nonlinear decomposition attack on other types of polycyclic groups should not be much different. As the nonlinear decomposition attack depends largely on the membership search problem of the group, any group with an efficient membership search problem would be vulnerable under nonlinear decomposition attack, if it is used as a platform group for a conjugacy problem based cryptosystem.

4.2 Better Upper Bounds

Since we only assumed the worst case on analyzing the complexity of these algorithms, one could easily apply more constraints and come up with a better upper bound for the computational complexity of the attack.

Appendix A

Group Theory

A.1 Commutator Subgroups

Since polycyclic groups are strongly related to the derived series and the lower central series, below are some of the relevant propositions and properties of those series from [15]. Note: The ***derived subgroup*** of G is the group $[G, G]$

Proposition A.1.1. The derived subgroup G' is normal in G and the quotient G/G' is abelian in G. If $N \trianglelefteq G$ and N is abelian in G, then $N \supseteq G'$.

Remark A.1.2. Suppose $H_1, H_2, K_1, K_2 \subseteq G$ such that $H_1 \subseteq H_2$ and $K_1 \subseteq K_2$. Then $[H_1, K_1] \subseteq [H_2, K_2]$.

Corollary A.1.3. If $H \subseteq G$, then $H^{(i)} \subseteq G^{(i)}$ and $\gamma_i(H) \subseteq \gamma_i(G)$.

Proof. Apply induction on the above proposition. □

Proposition A.1.4. If $f : G_1 \to G_2$ is a group homomorphism, then $[f(H), f(K)] = f([H, K])$ for all $H, K \subseteq G$

Corollary A.1.5. If $f : G_1 \to G_2$ is a group homomorphism, then $f(G)^{(i)} = f(G^{(i)})$ and $\gamma_i(f(G_1)) = f(\gamma_i(G_2))$

Proof. Applying induction on the above proposition. □

Corollary A.1.6. If $N \trianglelefteq G$ and g is the natural homomorphism from G to G/N. Then $g([H,K]) = [g(H), g(K)]$

Proposition A.1.7. If $H, K \trianglelefteq G$, then $[H,K] \trianglelefteq G$ and $[H,K] \subseteq H \cap K$.

Proposition A.1.8. For any elements $x, y, z \in G$, the following hold.

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

Proposition A.1.9. If $H, K, L \subseteq G$ and $N \trianglelefteq G$ such that $[K, L, H], [L, H, K] \subseteq N$, then $[H, K, L] \subseteq N$.

Proposition A.1.10. For all $i \geq 1$ and $j \geq 1$,$[\gamma_i(G), \gamma_j(G)] \subseteq \gamma_{i+j}(G)$.

Corollary A.1.11. $G^{(i)} \subseteq \gamma_{2^i}(G)$ for all $i \geq 0$.

A.2 Implementation of the Membership Search Algorithm

```
word_solve_new := function(G, gen_set, gm) ;
    result := [];
    H := Subgroup(G, gen_set);
    MyFreeGroup := FreeGroup(Length(gen_set));

    gen_bundle := IgsParallel(gen_set, GeneratorsOfGroup(MyFreeGroup));

    gen_elems := gen_bundle[1];
    gen_reference := gen_bundle[2];

    h := gm;
    while not h=Identity(G) do
        h_exp := GenExpList(h);
```

```
        for i in [1..Length(gen_elems)] do
            if GenExpList(gen_elems[i])[1] = h_exp[1] then
                w := gen_elems[i];
                index := i;
                break;
            fi;
        od;
        exp := 0;

        for j in [1..Order(GeneratorsOfGroup(G)[h_exp[1]])] do
            if Order(GeneratorsOfGroup(G)[h_exp[1]]) <> infinity then
                exp := (exp + GenExpList(w)[2])
                        mod Order(GeneratorsOfGroup(G)[h_exp[1]]);
            else
                exp := (exp + GenExpList(w)[2]);
            fi;

            if exp = GenExpList(h)[2] then

                exp := j;
                break;
            fi;
        od;

        v := w^(exp);
        h := v^-1*h;
        Add(result,index);
        Add(result,exp);
    od;

    # converting the format according to LetterRepAssocWord format
    result2 := [];
    for o in [1..(Length(result)/2)] do
        index := (o*2) - 1;
        for p in [1..result[index+1]] do
            Add(result2, LetterRepAssocWord(gen_reference[result[index]]));
        od;
    od;

    result := Flat(result2);

    return result;
end;
```

Bibliography

[1] Groups, algorithms, and programming. https://www.gap-system.org/.

[2] What is the time complexity of euclid's algorithm?, Dec 2012. https://math.stackexchange.com/users/50776/mario-carneiro.

[3] Gap package polycyclic, 2013. https://www.gap-system.org/Manuals/pkg/polycyclic-2.11/doc/manual.pdf.

[4] Cavallo, B., and Kahrobaei, D. A family of polycyclic groups over which the uniform conjugacy problem is np-complete. *International Journal of Algebra and Computation 24*, 04 (2014), 515530. https://arxiv.org/abs/1403.4153v2.

[5] Eick, B., and Kahrobaei, D. Polycyclic groups: A new platform for cryptology? .

[6] Gryak, J., and Kahrobaei, D. The status of polycyclic group-based cryptography: A survey and open problems. *Groups Complexity Cryptology 8*, 2 (Jan 2016).

[7] Habeeb, M., Kahrobaei, D., Koupparis, C., and Shpilrain, V. Public key exchange using semidirect product of (semi)groups. *Applied Cryptography and Network Security Lecture Notes in Computer Science* (2013), 475486.

[8] Kahrobaei, D., and Khan, B. Nis05-6: A non-commutative generalization of elgamal key exchange using polycyclic groups. *IEEE Globecom 2006* (2006).

[9] Ko, K. H., Lee, S. J., Cheon, J. H., Han, J. W., Kang, J., and Park, C. Advances in cryptology crypto 2000. *Lecture Notes in Computer Science* (2000), 166–183.

[10] Myasnikov, A., and Romankov, V. A linear decomposition attack. *Groups Complexity Cryptology 7*, 1 (Jan 2015).

[11] Myasnikov, A. D., and Ushakov, A. Length based attack and braid groups: Cryptanalysis of anshel-anshel-goldfeld key exchange protocol. *Public Key Cryptography PKC 2007 Lecture Notes in Computer Science* (2007), 7688.

[12] Myasnikov, A. G., Shpilrain, V., and Ushakov, A. *Group-based cryptography*. Birkhauser Verlag, 2008.

[13] Newman, M., and Niemeyer, A. C. On complexity of multiplication in finite soluble groups. *Journal of Algebra 421* (2015), 425430.

[14] RomanKov, V. A nonlinear decomposition attack. *Groups Complexity Cryptology 8*, 2 (Jan 2016), 197207.

[15] Sims, C. C. *Computation with finitely presented groups*. Cambridge University Press, 2010.