On Polycyclic Group–Based Cryptography

Carmine **Monetta**

Università degli Studi di Salerno

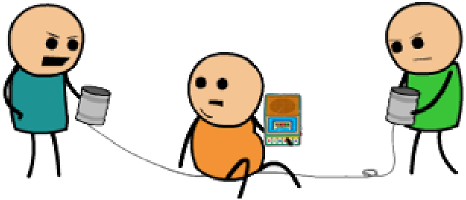
joint work (in progress)
with Antonio **Tortora**

Workshop BunnyTN 7

November 16, 2016

Background

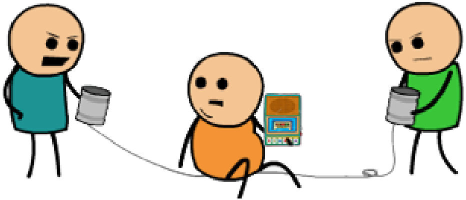
In cryptography, one of the most studied problems is how to share a secret key over an insecure channel.



Key exchange methods are usually based on one-way functions, that is functions which are easy to compute but whose inverses are difficult to determine.

Background

In cryptography, one of the most studied problems is how to share a secret key over an insecure channel.



Key exchange methods are usually based on one-way functions, that is functions which are easy to compute but whose inverses are difficult to determine.

There are several ways in which group theory can be used to construct one-way functions.

In 1999, I. Anshel, M. Anshel and D. Goldfeld introduced a key exchange protocol whose platform is a nonabelian group G.

There are several ways in which group theory can be used to construct one-way functions.

In 1999, I. Anshel, M. Anshel and D. Goldfeld introduced a key exchange protocol whose platform is a nonabelian group G.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS
  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.
  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.
- PRIVATE KEYS
  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.
  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.
- EXCHANGED INFORMATION
  Alice computes $b'_1 = b_1^A, \ldots, b'_k = b_k^A$, and sends them to Bob.
  Bob computes $a'_1 = a_1^B, \ldots, a'_l = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b_1' = b_1^A, \ldots, b_k' = b_k^A$, and sends them to Bob.

  Bob computes $a_1' = a_1^B, \ldots, a_l' = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b_1' = b_1^A, \ldots, b_k' = b_k^A$, and sends them to Bob.

  Bob computes $a_1' = a_1^B, \ldots, a_l' = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b'_1 = b_1^A, \ldots, b'_k = b_k^A$, and sends them to Bob.

  Bob computes $a'_1 = a_1^B, \ldots, a'_l = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b'_1 = b_1^A, \ldots, b'_k = b_k^A$, and sends them to Bob.

  Bob computes $a'_1 = a_1^B, \ldots, a'_l = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b'_1 = b_1^A, \ldots, b'_k = b_k^A$, and sends them to Bob.

  Bob computes $a'_1 = a_1^B, \ldots, a'_l = a_l^B$, and sends them to Alice.

Anshel-Anshel-Goldfeld

Circumstances: Alice and Bob want to agree on a common key.

Platform: let G be a nonabelian group

- PUBLIC KEYS

  Alice chooses $a_1, \ldots, a_l$ in G and makes them PUBLIC.

  Bob chooses $b_1, \ldots, b_k$ in G and makes them PUBLIC.

- PRIVATE KEYS

  Alice chooses $A \in \langle a_1, \ldots, a_l \rangle$.

  Bob chooses $B \in \langle b_1, \ldots, b_k \rangle$.

- EXCHANGED INFORMATION

  Alice computes $b_1' = b_1^A, \ldots, b_k' = b_k^A$, and sends them to Bob.

  Bob computes $a_1' = a_1^B, \ldots, a_l' = a_l^B$, and sends them to Alice.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a_1', \ldots, a_l') = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a'_1, \ldots, a'_l) = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a_1', \ldots, a_l') = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a'_1, \ldots, a'_l) = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \dots, a_l)$ as a word in $a_1, \dots, a_l$.
  2. Compute

$$A^{-1}w(a'_1, \dots, a'_l) = A^{-1}w(a_1^B, \dots, a_l^B)$$

$$= A^{-1}w(a_1, \dots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a'_1, \ldots, a'_l) = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a_1', \ldots, a_l') = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a_1', \ldots, a_l') = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

The shared key

- The shared key is $K = [A, B] = A^{-1}B^{-1}AB$.
- Alice determine K via:
  1. Write $A = w(a_1, \ldots, a_l)$ as a word in $a_1, \ldots, a_l$.
  2. Compute

$$A^{-1}w(a_1', \ldots, a_l') = A^{-1}w(a_1^B, \ldots, a_l^B)$$

$$= A^{-1}w(a_1, \ldots, a_l)^B = A^{-1}A^B = [A, B] = K.$$

- Bob uses the dual approach to determine K.

Eavesdropping

Since the conversation is not protected, an eavesdropper could obtain $b'_1, \dots b'_k$, and $a'_1, \dots a'_l$ as well.

Using the public data and the stolen information, one way to break the algorithm is the following:

$$\text{find } C \in \langle a_1, \dots, a_l \rangle \text{ such that } \begin{cases} b_1^C = b'_1 \\ \dots \\ b_k^C = b'_k. \end{cases}$$

Eavesdropping

Since the conversation is not protected, an eavesdropper could obtain $b'_1, \dots b'_k$, and $a'_1, \dots a'_l$ as well.

Using the public data and the stolen information, one way to break the algorithm is the following:

$$\text{find } C \in \langle a_1, \dots, a_l \rangle \text{ such that } \begin{cases} b_1^C = b'_1 \\ \dots \\ b_k^C = b'_k. \end{cases}$$

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b'_j = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a'_1, \ldots, a'_l) = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b_j' = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a_1', \ldots, a_l') = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

  obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b_j' = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a_1', \ldots, a_l') = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

  obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b'_j = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \dots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \dots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \dots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a'_1, \dots, a'_l) = C^{-1}v(a_1^B, \dots, a_l^B) = C^{-1}v(a_1, \dots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b_j' = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a_1', \ldots, a_l') = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b_j' = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a_1', \ldots, a_l') = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

obtaining the shared key.

Breaking AAG

- Note that $C = xA$ for some $x \in C_G(B)$:

  $b_j^C = b'_j = b_j^A$ implies $b_j^{CA^{-1}} = b_j$, that is $CA^{-1} \in C_G(b_j)$ for every $j = 1, \ldots, k$.

  Therefore, $CA^{-1} \in C_G(b_1, \ldots, b_m) \subset C_G(B)$.

- Write $C = v(a_1, \ldots, a_l)$ as word in the generators a_i, and compute

$$C^{-1}v(a'_1, \ldots, a'_l) = C^{-1}v(a_1^B, \ldots, a_l^B) = C^{-1}v(a_1, \ldots, a_l)^B$$

$$= C^{-1}C^B = (xA)^{-1}B^{-1}(xA)B = A^{-1}x^{-1}B^{-1}xAB$$

$$= A^{-1}B^{-1}AB = [A, B]$$

  obtaining the shared key.

In order to break AAG, one needs to solve:

Word Problem

Let G be a finitely presented group. If you are given an element g in G, decide whether $g = 1$.

Multiple Conjugacy Search Problem

Let $x_1, \dots, x_n, y_1, \dots, y_n$ be elements of G and suppose that there exists $C \in G$ such that

$$\begin{cases} x_1^C = y_1 \\ \dots \\ x_n^C = y_n. \end{cases}$$

Find such a C.

In order to break AAG, one needs to solve:

Word Problem

Let G be a finitely presented group. If you are given an element g in G, decide whether $g = 1$.

Multiple Conjugacy Search Problem

Let $x_1, \ldots, x_n, y_1, \ldots, y_n$ be elements of G and suppose that there exists $C \in G$ such that

$$\begin{cases} x_1^C = y_1 \\ \ldots \\ x_n^C = y_n. \end{cases}$$

Find such a C.

In order to break AAG, one needs to solve:

Word Problem

Let G be a finitely presented group. If you are given an element g in G, decide whether $g = 1$.

Multiple Conjugacy Search Problem

Let $x_1, \dots, x_n, y_1, \dots, y_n$ be elements of G and suppose that there exists $C \in G$ such that

$$\begin{cases} x_1^C = y_1 \\ \dots \\ x_n^C = y_n. \end{cases}$$

Find such a C.

What features should a group G have to be suitable for AAG?

- G requires fast multiplication and comparison of elements.
- G should have a difficult multiple conjugacy search problem.

What features should a group G have to be suitable for AAG?

- G requires fast multiplication and comparison of elements.
- G should have a difficult multiple conjugacy search problem.

What features should a group G have to be suitable for AAG?

- G requires fast multiplication and comparison of elements.
- G should have a difficult multiple conjugacy search problem.

Recently, B. Eick and D. Kahrobaei investigated the algorithmic properties of a special class of groups, namely

Polycyclic Groups

Recently, B. Eick and D. Kahrobaei investigated the algorithmic properties of a special class of groups, namely

Polycyclic Groups

Polycyclic Groups

A group G is said to be polycyclic if it has a chain of subgroups

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

in which each G_{i+1} is a normal subgroup of G_i, and the quotient group G_i/G_{i+1} is cyclic.

Such a chain of subgroups is called a polycyclic series.

Polycyclic Groups

A group G is said to be polycyclic if it has a chain of subgroups

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

in which each G_{i+1} is a normal subgroup of G_i, and the quotient group G_i/G_{i+1} is cyclic.

Such a chain of subgroups is called a polycyclic series.

Let $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ be a polycyclic series for G.

As G_i/G_{i+1} is cyclic, for every index i there exists $x_i \in G_i$ such that

$$\langle x_i G_{i+1} \rangle = G_i/G_{i+1}. \quad (1)$$

$X = [x_1, \ldots, x_n]$ is said to be a polycyclic sequence for G if (1) holds for $i = 1, \ldots, n$.

The sequence of relative orders for X is the sequence

$$R(X) = (r_1, \ldots, r_n)$$

defined by $r_i = |G_i : G_{i+1}| \in \mathbb{N} \cup \{\infty\}$.

Moreover, we define $I(X)$ as the set of $i \in \{1, \ldots, n\}$ such that r_i is finite.

Let $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ be a polycyclic series for G.

As G_i/G_{i+1} is cyclic, for every index i there exists $x_i \in G_i$ such that

$$\langle x_i G_{i+1} \rangle = G_i/G_{i+1}. \tag{1}$$

$X = [x_1, \ldots, x_n]$ is said to be a polycyclic sequence for G if (1) holds for $i = 1, \ldots, n$.

The sequence of relative orders for X is the sequence

$$R(X) = (r_1, \ldots, r_n)$$

defined by $r_i = |G_i : G_{i+1}| \in \mathbb{N} \cup \{\infty\}$.

Moreover, we define $I(X)$ as the set of $i \in \{1, \ldots, n\}$ such that r_i is finite.

Let $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ be a polycyclic series for G.

As G_i/G_{i+1} is cyclic, for every index i there exists $x_i \in G_i$ such that

$$\langle x_i G_{i+1} \rangle = G_i/G_{i+1}. \tag{1}$$

$X = [x_1, \ldots, x_n]$ is said to be a polycyclic sequence for G if (1) holds for $i = 1, \ldots, n$.

The sequence of relative orders for X is the sequence

$$R(X) = (r_1, \ldots, r_n)$$

defined by $r_i = |G_i : G_{i+1}| \in \mathbb{N} \cup \{\infty\}$.

Moreover, we define $I(X)$ as the set of $i \in \{1, \ldots, n\}$ such that r_i is finite.

Let $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ be a polycyclic series for G.

As G_i/G_{i+1} is cyclic, for every index i there exists $x_i \in G_i$ such that

$$\langle x_i G_{i+1} \rangle = G_i/G_{i+1}. \qquad (1)$$

$X = [x_1, \ldots, x_n]$ is said to be a polycyclic sequence for G if (1) holds for $i = 1, \ldots, n$.

The sequence of relative orders for X is the sequence

$$R(X) = (r_1, \ldots, r_n)$$

defined by $r_i = |G_i : G_{i+1}| \in \mathbb{N} \cup \{\infty\}$.

Moreover, we define $I(X)$ as the set of $i \in \{1, \ldots, n\}$ such that r_i is finite.

Polycyclic Presentation

A presentation $\langle x_1, \ldots, x_n \mid R \rangle$ is called a polycyclic presentation if there exist a sequence $S = (s_1, \ldots, s_n)$ with $s_i \in \mathbb{N} \cup \{\infty\}$ and integers $a_{i,k}, b_{i,j,k}, c_{i,j,k}$ such that R consists of the following relations:

$$x_i^{s_i} = R_{i,i} := x_{i+1}^{a_{i,i+1}} \cdots x_n^{a_{i,n}} \quad \text{for } 1 \leq i \leq n, \textbf{ if } s_i \textbf{ is finite};$$

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

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

Polycyclic Presentation

A presentation $\langle x_1, \ldots, x_n \mid R\rangle$ is called a polycyclic presentation if there exist a sequence $S = (s_1, \ldots, s_n)$ with $s_i \in \mathbb{N} \cup \{\infty\}$ and integers $a_{i,k}, b_{i,j,k}, c_{i,j,k}$ such that R consists of the following relations:

$$x_i^{s_i} = R_{i,i} := x_{i+1}^{a_{i,i+1}} \cdots x_n^{a_{i,n}} \quad \text{for } 1 \leq i \leq n, \textbf{ if } s_i \textbf{ is finite};$$

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

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

Polycyclic Presentation

A presentation $\langle x_1, \ldots, x_n \mid R \rangle$ is called a polycyclic presentation if there exist a sequence $S = (s_1, \ldots, s_n)$ with $s_i \in \mathbb{N} \cup \{\infty\}$ and integers $a_{i,k}, b_{i,j,k}, c_{i,j,k}$ such that R consists of the following relations:

$$x_i^{s_i} = R_{i,i} := x_{i+1}^{a_{i,i+1}} \cdots x_n^{a_{i,n}} \quad \text{for } 1 \leq i \leq n \text{, if } s_i \text{ is finite;}$$

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

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

Polycyclic Presentation

A presentation $\langle x_1, \ldots, x_n \mid R \rangle$ is called a polycyclic presentation if there exist a sequence $S = (s_1, \ldots, s_n)$ with $s_i \in \mathbb{N} \cup \{\infty\}$ and integers $a_{i,k}, b_{i,j,k}, c_{i,j,k}$ such that R consists of the following relations:

$$x_i^{s_i} = R_{i,i} := x_{i+1}^{a_{i,i+1}} \cdots x_n^{a_{i,n}} \quad \text{for } 1 \leq i \leq n, \textbf{ if } s_i \textbf{ is finite};$$

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

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

Polycyclic Presentation

A presentation $\langle x_1, \ldots, x_n \mid R \rangle$ is called a polycyclic presentation if there exist a sequence $S = (s_1, \ldots, s_n)$ with $s_i \in \mathbb{N} \cup \{\infty\}$ and integers $a_{i,k}, b_{i,j,k}, c_{i,j,k}$ such that R consists of the following relations:

$$x_i^{s_i} = R_{i,i} := x_{i+1}^{a_{i,i+1}} \cdots x_n^{a_{i,n}} \quad \text{for } 1 \leq i \leq n, \textbf{ if } s_i \textbf{ is finite};$$

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

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

Word Problem

Suppose that G is given by a pc-presentation.

Let $G_i = \langle x_i, \ldots, x_n \rangle$ for $1 \leq i \leq n+1$.

Consistency

A pc-presentation is consistence if $s_i = |G_i : G_{i+1}|$ for every $i \in I(X)$.

Normal Form in a Consistence PC-Presentation

For each $g \in G$ there exists a unique vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ with $0 \leq e_i < s_i$ if $i \in I(X)$ such that

$$g = x_1^{e_1} \ldots x_n^{e_n}.$$

Word Problem

Suppose that G is given by a pc-presentation.

Let $G_i = \langle x_i, \ldots, x_n \rangle$ for $1 \leq i \leq n+1$.

Consistency

A pc-presentation is consistence if $s_i = |G_i : G_{i+1}|$ for every $i \in I(X)$.

Normal Form in a Consistence PC-Presentation

For each $g \in G$ there exists a unique vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ with $0 \leq e_i < s_i$ if $i \in I(X)$ such that

$$g = x_1^{e_1} \ldots x_n^{e_n}.$$

Collection

Suppose an element g is given as a word in $x_1, \ldots, x_n$.

The collection algorithm determines the normal form of g by an iterated rewriting of the word using the relations of the polycyclic presentation.

Efficiency

The collection algorithm is generally effective in practical applications.

- For **finite** groups, collection was shown to be **polynomial** by Leedham-Green and Soicher.
- For **infinite** groups, Gebhardt showed that the complexity depends on the exponents occurring during the collection process, so it has **no bound**.

Collection

Suppose an element g is given as a word in $x_1, \ldots, x_n$.

The collection algorithm determines the normal form of g by an iterated rewriting of the word using the relations of the polycyclic presentation.

Efficiency

The collection algorithm is generally effective in practical applications.

- For **finite** groups, collection was shown to be **polynomial** by Leedham-Green and Soicher.
- For **infinite** groups, Gebhardt showed that the complexity depends on the exponents occurring during the collection process, so it has **no bound**.

Collection

Suppose an element g is given as a word in $x_1, \ldots, x_n$.

The collection algorithm determines the normal form of g by an iterated rewriting of the word using the relations of the polycyclic presentation.

Efficiency

The collection algorithm is generally effective in practical applications.

- For **finite** groups, collection was shown to be **polynomial** by Leedham-Green and Soicher.
- For **infinite** groups, Gebhardt showed that the complexity depends on the exponents occurring during the collection process, so it has **no bound**.

Collection

Suppose an element g is given as a word in $x_1, \ldots, x_n$.

The collection algorithm determines the normal form of g by an iterated rewriting of the word using the relations of the polycyclic presentation.

Efficiency

The collection algorithm is generally effective in practical applications.

- For **finite** groups, collection was shown to be **polynomial** by Leedham-Green and Soicher.
- For **infinite** groups, Gebhardt showed that the complexity depends on the exponents occurring during the collection process, so it has **no bound**.

Conjugacy Search Problem

Multiple conjugacy search problem can be reduced to finitely many iterations of single conjugacy search problem and centralizers computation.

Conjugacy Search Problem (CSP)

If g and h are conjugate elements of G, find $u \in G$ such that

$$g^u = h.$$

How to solve CSP

Let G be given by a consistent pc-presentation. Let $g, h \in G$ and $U \leq G$:

Problems

- Decide if g and h are conjugate in U.
- If g and h are conjugate, determine a conjugating element in U.
- Compute $C_U(g)$.

How to solve CSP

Let G be given by a consistent pc-presentation. Let $g, h \in G$ and $U \leq G$:

Problems

- Decide if g and h are conjugate in U.
- If g and h are conjugate, determine a conjugating element in U.
- Compute $C_U(g)$.

How to solve CSP

Let G be given by a consistent pc-presentation. Let $g, h \in G$ and $U \leq G$:

Problems

- Decide if g and h are conjugate in U.
- If g and h are conjugate, determine a conjugating element in U.
- Compute $C_U(g)$.

How to solve CSP

Let G be given by a consistent pc-presentation. Let $g, h \in G$ and $U \leq G$:

Problems

- Decide if g and h are conjugate in U.
- If g and h are conjugate, determine a conjugating element in U.
- Compute $C_U(g)$.

"Privileged"

Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Leedham-Green and Soicher.
- **Conjugacy Search Problem**: can be solved using induction on a refinement of the lower central series, as shown by Sims.

Virtually Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Du Sautoy.
- **Conjugacy Search Problem**: no special solution is known so far.

"Privileged"

Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Leedham-Green and Soicher.
- **Conjugacy Search Problem**: can be solved using induction on a refinement of the lower central series, as shown by Sims.

Virtually Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Du Sautoy.
- **Conjugacy Search Problem**: no special solution is known so far.

"Privileged"

Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Leedham-Green and Soicher.
- **Conjugacy Search Problem**: can be solved using induction on a refinement of the lower central series, as shown by Sims.

Virtually Nilpotent

- **Word Problem**: can be solved evaluating polynomials, as shown by Du Sautoy.
- **Conjugacy Search Problem**: no special solution is known so far.

Virtually Nilpotent Polycyclic Groups

Growth Rate

Let G be a finitely generated group. The growth rate of G is the asymptotic behaviour of its growth function $\gamma : \mathbb{N} \to \mathbb{R}$ defined as

$$\gamma(n) = |\{w \in G \ : \ l(w) \leq n\}|,$$

where $l(w)$ is the length of w as a word in the generators of G.

Remark

Wolf and Milnor proved that polycyclic groups have polynomial growth rate if and only if they are virtually nilpotent.

Being the secret key a word in the group, the faster the growth rate the larger the key space.

Non-virtually nilpotent polycyclic groups seem to be good candidates to use as platform groups, having exponential growth rate.

Virtually Nilpotent Polycyclic Groups

Growth Rate

Let G be a finitely generated group. The growth rate of G is the asymptotic behaviour of its growth function $\gamma : \mathbb{N} \to \mathbb{R}$ defined as

$$\gamma(n) = |\{w \in G \ : \ l(w) \leq n\}|,$$

where $l(w)$ is the length of w as a word in the generators of G.

Remark

Wolf and Milnor proved that polycyclic groups have polynomial growth rate if and only if they are virtually nilpotent.

Being the secret key a word in the group, the faster the growth rate the larger the key space.

Non-virtually nilpotent polycyclic groups seem to be good candidates to use as platform groups, having exponential growth rate.

Classes of Groups

$$\{\text{Polycyclic}\}$$
$$\cup$$
$$\{\text{Virtually Nilpotent Polycyclic}\}$$
$$\cup$$
$$\rightarrow \{\text{Supersoluble}\} \leftarrow$$
$$\cup$$
$$\{\text{Finitely Generated Nilpotent}\}$$

What about Supersoluble?

A group G is said to be supersoluble if it has a chain of subgroups

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

in which each G_i is a normal subgroup of G, and the quotient group G_i/G_{i+1} is cyclic.

A Special Subgroup in Supersolubles

For any $1 \leq i \leq n$, we can consider

$$C_G(G_i/G_{i+1}) = \{g \in G \mid [g, x] \in G_{i+1} \text{ for every } x \in G_i\}.$$

The intersection of all these centralizers

$$H = \bigcap_{i=1}^{n} C_G(G_i/G_{i+1})$$

is a normal nilpotent subgroup of G such that G/H is finite abelian.

A Special Subgroup in Supersolubles

For any $1 \leq i \leq n$, we can consider

$$C_G(G_i/G_{i+1}) = \{g \in G \mid [g,x] \in G_{i+1} \text{ for every } x \in G_i\}.$$

The intersection of all these centralizers

$$H = \bigcap_{i=1}^{n} C_G(G_i/G_{i+1})$$

is a normal nilpotent subgroup of G such that G/H is finite abelian.

Achievements

Recently, we focused our attention on the algorithmical properties of supersoluble groups, and we achieved a solution for MCSP in supersoluble groups.

Let G be a supersoluble group, and let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G.

Proposition

Let x and y be elements of G. Then x and y are conjugate in G if and only if x and y^{t_i} are conjugate in H for some $i \in \{1, \ldots, r\}$.

Proof.

If x and y^{t_i} are conjugate in H for some i, then of course x and y are conjugate in G.

Viceversa, suppose that x and y are conjugate in $G = \bigcup_{i=1}^{r} t_i H$. Therefore, there exist $u \in H$ and $i \in \{1, \ldots, r\}$ such that $x = y^{t_i u} = (y^{t_i})^u$. □

Let G be a supersoluble group, and let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G.

Proposition

Let x and y be elements of G. Then x and y are conjugate in G if and only if x and y^{t_i} are conjugate in H for some $i \in \{1, \ldots, r\}$.

Proof.

If x and y^{t_i} are conjugate in H for some i, then of course x and y are conjugate in G.

Viceversa, suppose that x and y are conjugate in $G = \bigcup_{i=1}^{r} t_i H$. Therefore, there exist $u \in H$ and $i \in \{1, \ldots, r\}$ such that $x = y^{t_i u} = (y^{t_i})^u$. □

Let G be a supersoluble group, and let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G.

Proposition

Let x and y be elements of G. Then x and y are conjugate in G if and only if x and y^{t_i} are conjugate in H for some $i \in \{1, \ldots, r\}$.

Proof.

If x and y^{t_i} are conjugate in H for some i, then of course x and y are conjugate in G.

Viceversa, suppose that x and y are conjugate in $G = \bigcup_{i=1}^{r} t_i H$. Therefore, there exist $u \in H$ and $i \in \{1, \ldots, r\}$ such that $x = y^{t_i u} = (y^{t_i})^u$. □

Let G be a supersoluble group, and let $T = \{t_1, \dots, t_r\}$ be a transversal to H in G.

Proposition

Let x and y be elements of G. Then x and y are conjugate in G if and only if x and y^{t_i} are conjugate in H for some $i \in \{1, \dots, r\}$.

Proof.

If x and y^{t_i} are conjugate in H for some i, then of course x and y are conjugate in G.

Viceversa, suppose that x and y are conjugate in $G = \bigcup_{i=1}^{r} t_i H$. Therefore, there exist $u \in H$ and $i \in \{1, \dots, r\}$ such that $x = y^{t_i u} = (y^{t_i})^u$. □

Let G be a supersoluble group, and let $T = \{t_1, \dots, t_r\}$ be a transversal to H in G.

Proposition

Let x and y be elements of G. Then x and y are conjugate in G if and only if x and y^{t_i} are conjugate in H for some $i \in \{1, \dots, r\}$.

Proof.

If x and y^{t_i} are conjugate in H for some i, then of course x and y are conjugate in G.

Viceversa, suppose that x and y are conjugate in $G = \bigcup_{i=1}^{r} t_i H$. Therefore, there exist $u \in H$ and $i \in \{1, \dots, r\}$ such that $x = y^{t_i u} = (y^{t_i})^u$. □

If $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ is a normal cyclic series of G, we can consider

$$G \geq H = H_1 \geq \ldots \geq H_n \geq H_{n+1} = 1$$

where $H_i = H \cap G_i$. So for any i

- $H_i \lhd G$,
- G/H is finite abelian,
- H_i/H_{i+1} is cyclic,
- $H_i/H_{i+1} \leq Z(H/H_{i+1})$.

If $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ is a normal cyclic series of G, we can consider

$$G \geq H = H_1 \geq \ldots \geq H_n \geq H_{n+1} = 1$$

where $H_i = H \cap G_i$. So for any i

- $H_i \lhd G$,
- G/H is finite abelian,
- H_i/H_{i+1} is cyclic,
- $H_i/H_{i+1} \leq Z(H/H_{i+1})$.

If $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ is a normal cyclic series of G, we can consider

$$G \geq H = H_1 \geq \ldots \geq H_n \geq H_{n+1} = 1$$

where $H_i = H \cap G_i$. So for any i

- $H_i \lhd G$,
- G/H is finite abelian,
- H_i/H_{i+1} is cyclic,
- $H_i/H_{i+1} \leq Z(H/H_{i+1})$.

If $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ is a normal cyclic series of G, we can consider

$$G \geq H = H_1 \geq \ldots \geq H_n \geq H_{n+1} = 1$$

where $H_i = H \cap G_i$. So for any i

- $H_i \triangleleft G$,
- G/H is finite abelian,
- H_i/H_{i+1} is cyclic,
- $H_i/H_{i+1} \leq Z(H/H_{i+1})$.

If $G = G_1 \geq G_2 \geq \ldots \geq G_{n+1} = 1$ is a normal cyclic series of G, we can consider

$$G \geq H = H_1 \geq \ldots \geq H_n \geq H_{n+1} = 1$$

where $H_i = H \cap G_i$. So for any i

- $H_i \lhd G$,
- G/H is finite abelian,
- H_i/H_{i+1} is cyclic,
- $H_i/H_{i+1} \leq Z(H/H_{i+1})$.

CSP in Supersoluble

1. Compute each centralizer $C_G(G_i/G_{i+1})$ as kernel of some homomorphisms between polycyclic groups.
2. Consider $H = \bigcap_{i=1}^{n} C_G(G_i/G_{i+1})$.
3. Since H is nilpotent, use well-known methods to check whether x and y^{t_i} are conjugate in H.

CSP in Supersoluble

1. Compute each centralizer $C_G(G_i/G_{i+1})$ as kernel of some homomorphisms between polycyclic groups.
2. Consider $H = \bigcap_{i=1}^{n} C_G(G_i/G_{i+1})$.
3. Since H is nilpotent, use well-known methods to check whether x and y^{t_i} are conjugate in H.

CSP in Supersoluble

1. Compute each centralizer $C_G(G_i/G_{i+1})$ as kernel of some homomorphisms between polycyclic groups.

2. Consider $H = \bigcap_{i=1}^{n} C_G(G_i/G_{i+1})$.

3. Since H is nilpotent, use well-known methods to check whether x and y^{t_i} are conjugate in H.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1} h_{i_1}, \ldots, t_{i_m} h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j} h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1}h_{i_1}, \ldots, t_{i_m}h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j}h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \dots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1}h_{i_1}, \dots, t_{i_m}h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j}h_{i_j}} = v$$

for any $j = 1, \dots, m$.

- Determine $S = \{i \in \{1, \dots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1} h_{i_1}, \ldots, t_{i_m} h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j} h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1} h_{i_1}, \ldots, t_{i_m} h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j} h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1} h_{i_1}, \ldots, t_{i_m} h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

$$v^{t_{i_j} h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

In order to solve the Multiple Conjugacy Search Problem, we should be able to compute $C_U(v)$ for any $v \in G$ and any $U \leq G$.

It becomes easy if we manage to compute $C_G(v)$, since $C_U(v) = U \cap C_G(v)$.

We found an algorithm which works as follows.

Let $T = \{t_1, \ldots, t_r\}$ be a transversal to H in G. Then, $\{t_{i_1} h_{i_1}, \ldots, t_{i_m} h_{i_m}\}$ is a transversal to $C_H(v)$ in $C_G(v)$, where

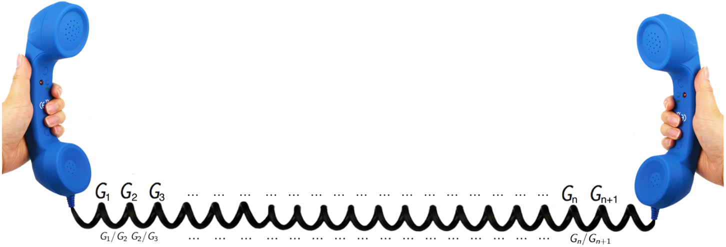
$$v^{t_{i_j} h_{i_j}} = v$$

for any $j = 1, \ldots, m$.

- Determine $S = \{i \in \{1, \ldots, n\} \mid v^{t_i h_i} = v\}$
- $C_G(v) = \langle C_H(v), t_i h_i \mid i \in S \rangle$.

Aims

We are now interested in studying the MCSP in virtually nilpotent groups hoping to extend the supersoluble case.



BIBLIOGRAPHY

I. Anshel, M. Anshel, D. Goldfeld
An algebraic method for public-key cryptography,
Math. Res. Let., 6:287-291, 1999

B. Eick and D. Kahrobaei
Polycyclic groups: a new platform for cryptography,
preprint arxiv: math.gr/0411077. Technical report, 2004

V. Gebhardt
Efficient collection in infinite polycyclic groups,
J. Symbolic Comput., 34(3):213-228, 2002

J. Wolf
Growth of finitely generated solvable groups and curvature of Riemannian manifolds,
Journal of Differential Geometry, pages 421-446, 1968

J. Milnor
Growth of finitely generated solvable groups,
J. Differential Geom., 2(4):447-449,1968

M. Du Sautoy
Polycyclic groups, analytic groups and algebraic groups,
Proc. London Math. Soc. (3), 85(1):62-92, 2002.

C. ,C. Sims
Computation with finitely presented groups,
Enciclopedia of mathematics and its application

C. Leedham-Green, L. Soicher
Symbolic collection using deep thought,
LMS J. Comput. Math.,1:9-24, 1998

C. R. Leedham-Green, L. H. Soicher
Collection from the left and other strategies,
J. Symbolic Comput., 9(5-6):665-675, 1990. Computational group theory, Part 1.

J. Gryak, D. Kahrobaei
The status of polycyclic group-based cryptography: a survey and open problems,
arXiv:1607.05819 [cs.CR], 2016

Thank you for the attention!