AMSI SUMMER RESEARCH SCHOLARSHIPS 2025–26

Get a taste for Research this Summer

Post-quantum Cryptography

Ziyan Chen

Supervised by Nalini Joshi

University of Sydney

Jane Street®

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Abstract

Public-key encryption (PKE) enables secure communication over insecure networks without requiring a pre-shared secret key. In this report, we study algebraic methods to reach PKE based on non-commutative groups and their corresponding actions. We first review braid-group-based constructions, including Anshel-Anshel Goldfield type schemes, Diffie–Hellman-like conjugacy systems, and other conjugacy search problem based variants. We discuss their algebraic basic knowledge and known attack techniques, such as summit-set methods and length-based attacks.

We then introduce tensor-action-based cryptographic constructions. We explain the tensor isomorphism problem, its relation to multilinear algebra, and recent developments on weak-key structures and strong-key sampling. Also, we provide empirical and experimental evidence to compare the computational cost of tensor isomorphism with classical problems, namely the integer factoring problem. The report also aims to present an overall view of group-based and tensor-based public-key constructions, highlighting both their potential advantages and their known limitations.

1 Introduction

Public-key encryption (PKE) is one of the central problems of modern cryptography. It allows a sender to encrypt a message using a publicly available key, known as the public key, while only the holder of a private key can decrypt to recover the original message. Different from symmetric encryption, where two parties reach the same agreement, it removes the need to find out the actual secure key.

A major challenge in cryptography is to design systems whose security relies on problems that remain hard under quantum computation. This has motivated research into algebraic and non-commutative cryptographic constructions. One important direction is non-commutative-group-based cryptography, where secret information is hidden, typically in conjugacy-type relations. Braid groups provide a practical and well-studied platform for this approach [1]. For the braid groups, they have efficient normal (unique) forms [2] and support encryption schemes such as Anshel-Anshel Goldfield (AAG) [3] and Ko–Lee-type systems [4].

However, braid-based constructions also have challenges in security. Methods such as super summit sets and length-based attacks show that the algebraic structure of braid group can sometimes be exploited to recover secret conjugators [5, 6]. This motivates the search for candidates of alternative algebraic hardness assumptions.

A second direction explored in this report is tensor-based cryptography [7]. Instead of group conjugation, these constructions rely on tensor actions and the hardness of the tensor isomorphism problem. For tensors of order three or higher, the isomorphism problem appears significantly more complex than matrix isomorphism problem. Recent works also identify weak-key structures and propose strong-key sampling techniques to avoid structural vulnerabilities [8, 9, 10].

The goal of this report is to study these two algebraic paradigms in details, including their constructions, security assumptions, and known attacks. We present their mathematical foundations, describe concretely the public-key constructions, discuss known attacks, and examine the feasibility of tensor-based approaches as a future direction in post-quantum cryptography.

Statement of Authorship. I declare that this report is my own work. All definitions, constructions, and theoretical discussions are based on existing literature and are properly cited throughout the report. The only original contributions in this report are the experimental comparison between integer factoring and the 3-tensor isomorphism problem, as well as the illustrative examples provided in the appendix.

2 Public Key Encryption (PKE)

Traditional cryptography system generally refers to symmetric setting, where two parties, denoted to be A and B, would like to communicate with each other through a mutual key, say K. In a symmetric setting, two parties must agree on a shared secret key before any confidential communication can take place, which is difficult to achieve at scale over an insecure network. PKE, on the other hand, enables any sender to encrypt to a recipient using only a publicly available key, while only the recipient can decrypt using a private key. Before introducing the definition of PKE, the definition of negligible function is given to specify the goal of a PKE scheme.

Definition 2.1 (Negligible function)**.** A function $\mu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible* if for every polynomial $p(\cdot)$, there exists $N \in \mathbb{N}$ such that for all $k \geq N$,

$$\mu(k) \leq \frac{1}{p(k)}.$$

Equivalently, $\mu(k)$ vanishes faster than the inverse of any polynomial in the security parameter k.

Definition 2.2 (Public-key encryption scheme)**.** A *public-key encryption scheme* is a triple of algorithms

$$\Pi = (G, E, D)$$

such that the following conditions hold.[1]

Fix a security parameter $k \in \mathbb{N}$, which decides on the size of private key space. Consider two parties: the receiver A and the sender B. Party A generates keys and keeps the private key, while B encrypts messages to A using A's public key.

Key generation. On input 1^k, the algorithm G outputs a pair

$$(e, d) \leftarrow G(1^k),$$

where e is the public key and d is the private key. Party A publishes e and keeps d secret.

Encryption. On input $(1^k, e, m)$, where $m \in \{0,1\}^k$ is a message chosen by B, the algorithm E outputs a ciphertext

$$c \leftarrow E(1^k, e, m), \qquad c \in \{0,1\}^*.$$

where $\{0,1\}^*$ means any string consists of 0 or 1.

[1] In standard cryptographic definitions, G, E, D are required to be *probabilistic* polynomial-time algorithms (PPT). The randomness is necessary because deterministic encryption cannot achieve standard semantic security, and random coins ensure that encrypting the same message twice typically yields different ciphertexts.

Decryption. On input $(1^k, d, c)$, the algorithm D outputs a plaintext

$$m' \leftarrow D(1^k, d, c), \qquad m' \in \{0,1\}^*.$$

Correctness. For every $k \in \mathbb{N}$ and every message $m \in \{0,1\}^k$, the decryption error is negligible:

$$\Pr\big[D(1^k, d, E(1^k, e, m)) \neq m \ : \ (e,d) \leftarrow G(1^k)\big] \leq \mu(k),$$

for some negligible function $\mu(\cdot)$.

3 Braid Based Cryptography

3.1 Braid groups and normal forms

Definition 3.1 (braid group)**.** For $n \geq 2$, the *n-strand braid group* B_n is defined by the presentation

$$B_n \ = \ \left\langle \sigma_1, \ldots, \sigma_{n-1} \ \middle| \ \begin{array}{ll} \sigma_i\sigma_j = \sigma_j\sigma_i & \text{if } |i-j| \geq 2, \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{if } |i-j| = 1 \end{array} \right\rangle.$$

A word in the letters $\sigma_i^{\pm 1}$ is called an *n-strand braid word*. Two braid words are *equivalent* (where we denoted $\equiv$) if one can be transformed into the other using the relations mentioned above, indicating they represent the same braid. A demonstration for the meaning of $\sigma_i^{\pm 1}$ and one example of braid and its corresponding braid word is shown in Fig. 1. Intuitively, σ_i means to put the $(i+1)^{th}$ strand over the i^{th}, while σ_i^{-1} means the inverse, and the group operation, known as concatenation, means to connect two n-strand braids together.

Note that the representation (i.e., the braid word corresponds to a braid structure) of a braid is not unique because of equivalence relation. As a simple example, the braid represented by $\sigma_1\sigma_2\sigma_1$ can also be represented by $\sigma_2\sigma_1\sigma_2$ due to the equivalence relation in braid definition. This motivates the derivation of a normal form (of representation), which exists and is unique for any braid.

We start by giving the definition of positive braids, which is a key concept in building up one of its normal form definitions. Then we define Garside elements and simple braids.

Definition 3.2 (Positive braid monoid)**.** Let B_n^+ be the submonoid (i.e., subgroup without inverse) of B_n generated by $\sigma_1, \ldots, \sigma_{n-1}$, namely $B_n^+ = \langle \sigma_1, \ldots, \sigma_{n-1} \rangle$. Its elements are called *positive braids*.

Definition 3.3 (Garside element)**.** Define $\Delta_n \in B_n^+$, the Garside element in B_n inductively by

$$\Delta_1 = 1, \qquad \Delta_{n+1} = \Delta_n \, \sigma_n\sigma_{n-1}\cdots\sigma_1.$$

Definition 3.4 (Simple braids)**.** A positive braid $b \in B_n^+$ is *simple* if it is a *left divisor* of Δ_n, i.e. if there exists $c \in B_n^+$ with $\Delta_n = bc$.

For example, in B_4, the braid $b = \sigma_1\sigma_2$ is simple, since by picking $c = \sigma_1\sigma_3\sigma_2\sigma_1$, we can verify:

$$bc = (\sigma_1\sigma_2)(\sigma_1\sigma_3\sigma_2\sigma_1) = \Delta_4$$

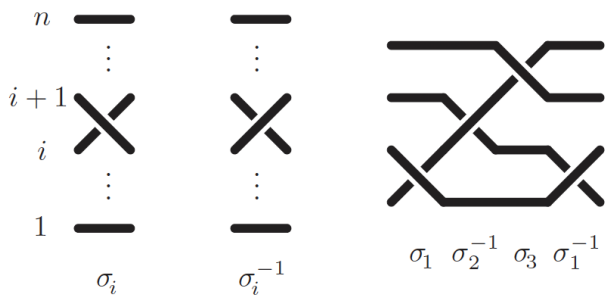

Figure 1: A demonstration and example for a braid from [1]

By contrast, in B_3, the braid $y = \sigma_1^2$ is not simple. This can be understood that the pair of strands $(1, 2)$ crosses twice in σ_1^2, so y cannot be a divisor of Δ_3 (in $\Delta_3 = \sigma_1\sigma_2\sigma_1$, each pair of strands crosses at most once).

Armed with the Def. 3.4, the normal sequence motivated by the normal form [2], is defined:

Definition 3.5 (Greedy Normal sequence)**.** A finite sequence $(k; b_1, \ldots, b_r)$ with $k \in \mathbb{Z}$ and b_i simple braids is called (Garside) *normal* if:

- each b_i is neither 1 nor Δ_n; and
- for each i, the braid b_i is the *maximal simple left divisor* of the product $b_i b_{i+1} \cdots b_r$ (equivalently, b_i is the left greatest common divisor (gcd) of $b_i \cdots b_r$ and Δ_n).

Where in [2], the uniqueness and existence of normal form is proposed and proved as below:

Proposition 3.6 (Greedy (Garside) normal form (GNF))**.** *Every braid $b \in B_n$ admits a unique decomposition*

$$b \;=\; \Delta_n^k\, b_1 b_2 \cdots b_r,$$

where $(k; b_1, \ldots, b_r)$ is a normal sequence (defined in Def. 3.5). The integer r is called the canonical length *(or* complexity*) of b.*

As an example of how a braid representation is converted into normal form, consider the braid group B_3 with Garside element Δ with $w \;=\; \sigma_1^{-1} \;\in B_3$. We convert w into Garside (greedy) normal form:

$$w = \sigma_1^{-1} = \Delta^{-1}(\Delta\sigma_1^{-1}) = \Delta^{-1}(\sigma_1\sigma_2\sigma_1\sigma_1^{-1}) = \Delta^{-1}\sigma_1\sigma_2$$

We can verify, from Def. 3.4 that $\sigma_1\sigma_2$ is simple, which finishes the conversion with $k = -1$ and $b_1 = \sigma_1\sigma_2$. Note that there is an algorithmic method for this conversion introduced in [2].

3.2 Cryptography Schemes on Braid Group

In this part, we introduce several existing cryptography schemes, including Anshel-Anshel Goldfield proposed in [3], Diffie-Hellman-like (DH) proposed in [11], and conjugator search problem (CSP) under ElGamal (CSP-ElG) proposed in [12, 13, 14, 4] that could be applied to braid group to create PKE scheme.

3.2.1 The AAG PKE scheme

Algorithm 3.1 (AAG-based PKE)**.** *Fix a group G (e.g. B_n) and public subsets $P = \{p_1, \dots, p_k\} \subseteq G$ and $Q = \{q_1, \dots, q_\ell\} \subseteq G$. Let $H : G \to \{0,1\}^N$ be a hash function (such that $H(g_1) + H(g_2) = \mathbf{0}$ if $g_1 = g_2 \in G$ because of its binary nature.*

A user (Alice) samples a secret word u in the subgroup $\langle P \rangle$ and sets

$$s := u(p_1, \dots, p_k) \in \langle P \rangle.$$

Then she publishes the tuple

$$\mathsf{pk} = \big(P, Q, Q^s\big), \qquad Q^s := (sq_1s^{-1}, \dots, sq_\ell s^{-1}),$$

and keeps $\mathsf{sk} = s$, where pk and sk represent public key and private key respectively.

To encrypt a message $m \in \{0,1\}^N$ to Alice, Bob samples a secret word v in $\langle Q \rangle$ and sets

$$r := v(q_1, \dots, q_\ell) \in \langle Q \rangle.$$

He computes the conjugated P-tuple

$$P^r := (rp_1r^{-1}, \dots, rp_kr^{-1}),$$

and the shared element

$$K := v(Q^s)\, r^{-1}, \qquad \text{where } v(Q^s) \text{ means evaluating the same word } v \text{ on } (sq_is^{-1})_{i=1}^{\ell}.$$

He outputs the ciphertext and send it to Alice.

$$c := \big(P^r,\ m \oplus H(K)\big).$$

To decrypt $c = (P^r, c_2)$, Alice evaluates u on P^r:

$$u(P^r) := u(rp_1r^{-1}, \dots, rp_kr^{-1}),$$

and reconstructs

$$K' := s\, u(P^r)^{-1}.$$

She outputs $m' := c_2 \oplus H(K')$ to get the original message.

The security of the AAG scheme relies on the hardness of solving a multiple conjugacy search problem, which is given public tuples $(q_1, \dots, q_\ell)$ and $(q_1^s, \dots, q_\ell^s)$ with $q_i^s = sq_is^{-1}$, an adversary is required to recover the conjugator s. This task is essentially a conjugacy search problem, but with the additional difficulty that s is only determined up to right multiplication by elements of the centralizer $C_G(Q^s)$, since the conjugator is not unique (every elements inside the centralizer are all candidates). Therefore, breaking the AAG scheme amounts to solving a conjugacy problem and the centralizer of the conjugated public set Q^s, which is believed to be computationally infeasible in appropriately chosen noncommutative groups.

Proposition 3.7 (Correctness)**.** *The Decryption recovers the original plaintext, i.e.* $m' = m$.

Proof. Since $s = u(P)$ and $P^r = (rp_ir^{-1})_i$, evaluating u on P^r gives

$$u(P^r) = r\,u(P)\,r^{-1} = rsr^{-1}.$$

Hence, the reconstruction K' of the shared element K gives:

$$K' = s\,u(P^r)^{-1} = s\,(rsr^{-1})^{-1} = s\,r\,s^{-1}\,r^{-1}.$$

On the other hand, because $v(Q^s) = v(sQs^{-1}) = s\,v(Q)\,s^{-1} = srs^{-1}$, we have

$$K = v(Q^s)\,r^{-1} = (srs^{-1})r^{-1} = s\,r\,s^{-1}\,r^{-1} = K'.$$

Therefore $m' = c_2 \oplus H(K') = (m \oplus H(K)) \oplus H(K) = m$. □

3.2.2 The Diffie–Hellman-like conjugacy and Ko–Lee PKE

Next we describe a Diffie–Hellman-like key agreement in braid groups based on commuting subgroups [11], and then an ElGamal-like public-key encryption scheme which was introduced by Ko *et al.* [4].

Let B_n be the braid group. Let LB_n and UB_n be commuting subgroups, and let $h : B_n \to \{0,1\}^N$ be a collision-free (we can understand that to be injective) one-way hash (a.k.a. virtually injective one-way hash). Normally, we define the lower group $LB_n = \{\sigma_1, \ldots, \sigma_{m-1}\}$ and the upper group $UB_n = \{\sigma_m, \ldots \sigma_{m+1}\}$ to be the two commuting subgroups, where $m = \lfloor \frac{n}{2} \rfloor$. This commuting property can be directly interpreted from the equivalence relationship in Def.3.1 since the index difference between LB_n and UB_n is at least 2.

Definition 3.8 (Ko–Lee–Cheon–Han–Kang–Park PKE)**.** Alice samples $s \in LB_n$ and a public braid $p \in B_n$, keeping $\mathsf{sk} = s$ to be the private key and publishes:

$$\mathsf{pk} = (p, p'), \qquad p' := sps^{-1},$$

To encrypt $m \in \{0,1\}^N$, Bob samples $r \in UB_n$ and sends to Alice c defined below:

$$c := \big(p'',\ m \oplus h(rp'r^{-1})\big), \qquad p'' := rpr^{-1}.$$

To decrypt $c = (p'', c_2)$, Alice computes

$$m' := c_2 \oplus h(sp''s^{-1}).$$

The security of the Diffie–Hellman-like conjugacy schemes, for example, the Ko–Lee public-key encryption scheme, is basically based on the hardness of the conjugacy search problem. Given public elements p and $p' = sps^{-1}$, an adversary is required to recover the secret conjugator s from a single conjugacy instance.

Proposition 3.9 (Correctness)**.** *The Decryption recovers the original plaintext, i.e.* $m' = m$.

Proof. Because $r \in UB_n$ and $s \in LB_n$ commute, we have

$$sp''s^{-1} = s(rpr^{-1})s^{-1} = r(sps^{-1})r^{-1} = rp'r^{-1}.$$

Therefore $m' = c_2 \oplus h(sp''s^{-1}) = (m \oplus h(rp'r^{-1})) \oplus h(rp'r^{-1}) = m$. □

3.2.3 CSP-PKE: the basic CSP-ElG scheme

Wang *et al.* [14] proposed PKE constructions from the hardness of conjugacy-search assumptions over noncommutative monoids. Below we give the *basic* ElGamal-like construction CSP-ElG described in [14, §4.1].

Definition 3.10 (CSP-ElG PKE scheme)**.** Let G be a noncommutative monoid and let F be the conjugation-based map, where $F_a(b) = aba^{-1}$ on invertible a. Fix invertible $a \in G^{-1}$ and $b \in G$, and a finite set $T \subset \mathbb{Z}$. For $i \in T$, write $F_{a^i}(b)$ and denote $K[a,b] := \{F_{a^i}(b) : i \in T\}$, representing the set of b conjugated with all powers of a with powers in T.

Let k be a security parameter, message space $M = \{0,1\}^k$, and a hash function $H : K[a,b] \to M$. Alice chooses $s \in T$, set:

$$\mathsf{pk} := F_{a^s}(b), \qquad \mathsf{sk} := s.$$

On input pk and $m \in M$, Bob chooses $t \in T$ and sends to Alice:

$$c := \big(F_{a^t}(b),\ m \oplus H(F_{a^t}(\mathsf{pk}))\big).$$

Based on input $\mathsf{sk} = s$ and $c = (c_1, c_2)$, Alice recovers the message with output:

$$m' := c_2 \oplus H(F_{a^s}(c_1)).$$

The security of the CSP-ElG public-key encryption scheme is based on the assumed hardness of the conjugacy search problem in non-commutative groups. Given public parameters a, b and a public key $\mathsf{pk} = F_{a^s}(b) = a^s b a^{-s}$, an adversary who can recover the secret exponent s (or equivalently a valid conjugator a^s) can decrypt. Moreover, the encryption scheme tries to mask the message using the shared element:

$$F_{a^t}(\mathsf{pk}) = F_{a^{t+s}}(b),$$

so breaking this scheme amounts to computing this shared conjugate from publicly available data, which reduces to solving conjugacy-type problems (recovering s or t, or directly deriving $F_{a^{t+s}}(b)$).

Proposition 3.11 (Correctness)**.** *The Decryption recovers the original plaintext, i.e.* $m' = m$.

Proof. Using the power law, $F_{a^{s+t}}(b) = F_{a^s}(F_{a^t}(b))$ for conjugation-based systems [14], we obtain:

$$F_{a^t}(\mathsf{pk}) = F_{a^t}(F_{a^s}(b)) = F_{a^{t+s}}(b) \quad \text{and} \quad F_{a^s}(c_1) = F_{a^s}(F_{a^t}(b)) = F_{a^{s+t}}(b).$$

Thus $F_{a^t}(\mathsf{pk}) = F_{a^s}(c_1)$, so the two parties hash the same element and $c_2 \oplus H(F_{a^s}(c_1)) = (m \oplus H(F_{a^t}(\mathsf{pk}))) \oplus H(F_{a^s}(c_1)) = m$. □

3.3 Cryptoanalysis on Braid Group

We can tell from the previous schemes that the encryption method can be insecure once CSP can be solved efficiently, which is the kernel to encryption security. Note that for multiple conjugacy problem, given tuples

$(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ with $b_i = xa_ix^{-1}$, any solution is determined only up to right multiplication by elements of the centralizer:

$$C_G(b_1, \ldots, b_n) := \{\, g \in G \mid gb_i = b_ig \text{ for all } i = 1, \ldots, n \,\}$$

Hence solving the multiple conjugacy problem amounts to solving n single conjugacy problems together with computing (or approximating) the centralizer of the conjugated set [15].

Below we introduce two main attacks that aims to solve CSP efficiently.

3.3.1 Super summit set (SSS) for the conjugacy problem

Definition 3.12 (Garside invariants: inf and sup)**.** Let B_n with Garside element Δ. Every braid $x \in B_n$ has a (left) normal form:

$$x = \Delta^k x_1 x_2 \cdots x_r,$$

where each x_i is simple and $x_i \neq 1, \Delta$. Define

$$\inf(x) := k, \qquad \sup(x) := k + r.$$

Definition 3.13 (Summit set and super summit set)**.** For $x \in B_n$, its conjugacy class is:

$$[x] := \{g^{-1}xg : \ g \in B_n\}.$$

Define the *summit infimum* and *summit supremum* by

$$\inf_s(x) := \max\{\inf(y) : y \in [x]\}, \qquad \sup_s(x) := \min\{\sup(y) : y \in [x]\}.$$

The *super summit set* of x is

$$\mathrm{SSS}(x) := \{y \in [x] : \inf(y) = \inf_s(x) \text{ and } \sup(y) = \sup_s(x)\}.$$

Intuitively, we can understand the SSS of x as the set of braids conjugating to x with the minimum canonical length. Below we introduce cycling and decycling algorithm, which is the key in calculating $\mathrm{SSS}(x)$.

Algorithm 3.2 (Cycling and Decycling)**.** *Let B_n be equipped with the classical Garside structure and Garside element Δ. Let*

$$x = \Delta^k x_1 x_2 \cdots x_r$$

be the (left) normal form of a braid x, where each x_i is a simple braid. The cycling *operation on x is defined as the conjugation:*

$$\mathbf{c}(x) := x_1^{-1} x x_1 = \Delta^k x_2 x_3 \cdots x_r \, \tau^k(x_1),$$

where τ is the automorphism of B_n given by

$$\tau(y) := \Delta^{-1} y \Delta.$$

and actually, τ maps σ_i to σ_{n-i}. The decycling *operation on x is defined as the conjugation:*

$$\mathbf{d}(x) := x_r x x_r^{-1} = \Delta^k \tau^k(x_r)\, x_1 x_2 \cdots x_{r-1}.$$

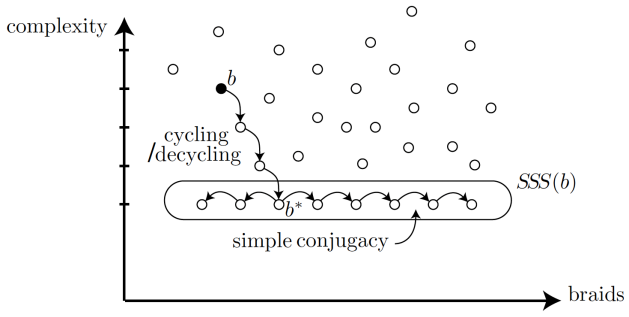

Figure 2: Enter SSS and enumerate on SSS from [1]

From the algorithm of cycling and decycling, it can be seen that the SSS set can be computed by iteratively cycling and decycling. In practice, one solves conjugacy search problem by searching inside SSS(x), which contains reaching SSS(x) and enumerating on SSS(x), shown in Fig. 2. Given a CSP instance (a, b) with $b = g^{-1}ag$, one standard approach is: compute an element of SSS(a) and an element of SSS(b) (using cycling/decycling), and then search for a conjugator within the SSS. An example is shown in App. A for demonstration.

3.3.2 Length-based attack (LBA) for braid-group CSP

Let G be a group (typically B_n) with a computable *length* function

$$\ell : G \to \mathbb{Z}_{\geq 0},$$

for example the canonical length coming from the Garside normal form, or the length of a reduced word. Consider a conjugacy search instance

$$b = x^{-1}ax,$$

where $a, b \in G$ are public and the secret conjugator x is assumed to lie in a subgroup generated by a public set $S = \{s_1, \ldots, s_m\} \subset G$.

Heuristically, if x is expressed as a product of generators from $S^{\pm 1}$, then conjugating b by a correct *initial* factor of x tends to reduce the length of the resulting conjugate (measured by ℓ), whereas conjugating by an unrelated element tends not to reduce ℓ. Length-based attacks exploit this heuristic to "peel off" x, which is generated by S.

Algorithm 3.3 (LBA as a best-first search)**.** *Fix an integer beam width $W \geq 1$ and a maximum iteration bound $T \geq 1$. Maintain a set of candidates*

$$\mathcal{C} \subseteq G \times G,$$

whose elements are pairs (y, c) where y is a guessed prefix of x and $c = yby^{-1}$ is the current conjugate. Initialize

$$\mathcal{C}_0 := \{(1, b)\}.$$

For $t = 0, 1, \ldots, T-1$, construct a new pool of candidates by expanding each $(y, c) \in \mathcal{C}_t$ using all generators $s \in S^{\pm 1}$:

$$(y, c) := (ys,\ s^{-1}cs).$$

Keep only the W candidates with smallest score, where a typical score is

$$\text{score}(y, c) := \ell(c).$$

Intuitively, Denote the surviving set by $\mathcal{C}_{t+1}$. Output any (y, c) such that $c = a$ (then y is a conjugator mapping b to a), or declare failure.

Note that if at some step a candidate satisfies $c = a$, then

$$c = yby^{-1} = a \quad \Longrightarrow \quad b = y^{-1}ay,$$

so y^{-1} is a valid solution to the CSP instance. An example is included in App. B.

4 Tensor Action and Tensor-Based Cryptography

4.1 Tensor action

Let V be a finite-dimensional vector space over a finite field $\mathbb{F}_q$. For $d \geq 2$, we denote

$$\mathcal{T} = V^{\otimes d}$$

to be the space of d-tensors over V [7].

Let $G = \text{GL}(V)$. There is a natural group action of G on $\mathcal{T}$ defined by

$$g \cdot T := (g \otimes g \otimes \cdots \otimes g)(T), \qquad g \in G,\ T \in \mathcal{T}.$$

More generally, one may consider the action of

$$G_d := \text{GL}(V)^d$$

given by

$$(g_1, \ldots, g_d) \cdot T = (g_1 \otimes \cdots \otimes g_d)(T).$$

Remark 4.1. This action generalizes the matrix action $X \mapsto AXB^T$ in the $d = 2$ case. For $d \geq 3$, the structure becomes significantly more complex, and the corresponding inversion problems appear to be computationally hard.

4.2 Tensor Isomorphism Problem

Definition 4.2 (Tensor Isomorphism Problem (TIP))**.** Let $T, T' \in \mathcal{T}$. The tensor isomorphism problem asks to determine whether there exist invertible linear maps

$$(g_1, \ldots, g_d) \in \text{GL}(V)^d$$

such that

$$T' = (g_1 \otimes \cdots \otimes g_d)(T).$$

If such maps exist, one further asks to compute them.

Remark 4.3. When $d = 2$, the problem reduces to matrix equivalence, which can be solved efficiently using linear algebra. However, for $d \geq 3$, the tensor isomorphism problem is believed to be computationally difficult and is closely related to multilinear isomorphism and group action inversion problems.

Our goal of introducing tensor action lies in finding a candidate for hardness problem. The security of tensor-based cryptographic constructions relies on the assumption that, for suitably chosen parameters, recovering $(g_1, \ldots, g_d)$ from (T, T') is computationally infeasible.

Note that it is sufficient to consider the case $d = 3$ in tensor-based cryptographic constructions. Indeed, it has been shown that the general d-tensor isomorphism problem can be reduced in polynomial time to the 3-tensor isomorphism (3TI) problem [16]. Therefore, from a computational complexity perspective, the 3-dimensional tensor isomorphism problem already captures the full hardness of the multi-dimensional case. Consequently, using $d = 3$ does not weaken the security assumption, while keeping the algebraic structure and implementation significantly simpler for the encoder to encode the message. This reduction result justifies that focusing on order-3 tensors in cryptographic applications is enough for us.

4.3 A tensor-based authentication scheme

We outline a basic authentication mechanism derived from the hardness of the tensor isomorphism problem. Note that below, we restrict to the case that $d = 3$.

For authentication scheme, it basically asks the prover to prove its identity to the verifier that he knows the secret key without revealing that. The key process lies in:

- For the prover who does not have the secret key, his wrong identity can be told by the verifier; for those who have secret key, they can be verified.
- The secret key cannot be easily deduced from the message the prover sends.

Let $T \in \mathcal{T}$ be a public tensor. The prover samples a secret key

$$(g_1, \ldots, g_d) \in \mathrm{GL}(V)^d$$

which represents the tensor action, and computes

$$T' := (g_1 \otimes \cdots \otimes g_d)(T).$$

The public key is T', while the secret key is the action $(g_1, \ldots, g_d)$.

Commitment. The prover chooses random tensor action

$$(h_1, \ldots, h_d) \in \mathrm{GL}(V)^d$$

and computes how this action applied to the public key by:

$$C := (h_1 \otimes \cdots \otimes h_d)(T).$$

The value C is sent to the verifier.

Challenge. The verifier sends a random challenge bit $b \in \{0, 1\}$.

Response. If $b = 0$, the prover reveals $(h_1, \ldots, h_d)$ and the verifier checks

$$C = (h_1 \otimes \cdots \otimes h_d)(T').$$

Note that in the case when $b = 0$, the secret key $(g_1, \ldots, g_d)$ is not included in the message sent from the prover, meaning the verifier is verifying nothing. If $b = 1$, the prover reveals $(h_1 g_1^{-1}, \ldots, h_d g_d^{-1})$ and the verifier checks whether:

$$C = ((h_1 g_1^{-1}) \otimes \cdots \otimes (h_d g_d^{-1}))(T').$$

Remark 4.4. The soundness of this protocol relies on the hardness of the tensor isomorphism problem. When given the commitment $C := (h_1 \otimes \cdots \otimes h_d)(T)$ and the public tensor T, the hardness of isomorphism problem indicates the hardness of interpreting $(h_1 \otimes \cdots \otimes h_d)$. Therefore, this indicates the hardness of recovering the secret action $(g_1, \ldots, g_d)$ when given the response $(h_1 g_1^{-1}, \ldots, h_d g_d^{-1})$ from the prover.

4.4 Pitfalls in key generation and a remedy

Although the tensor isomorphism problem is believed to be hard on average, recent works demonstrate that naive public-key sampling may lead to weak keys. In particular, random cubic tensors (here we assume $d = 3$) may exhibit special geometric structures, such as "triangles", or low-rank initializations, which can be exploited to recover the secret isomorphism more efficiently than generic attacks [10, 9].

Moreover, degeneracy testing in higher dimensions is itself believed to be computationally hard, since hyperdeterminant calculation is believed to be NP hard [10]. Therefore, the classical strategy of sampling a random tensor and rejecting degenerate instances is infeasible in practice.

To avoid the above pitfalls, it is desirable to sample from non-degenerate tensors, while ensuring that the distribution remains computationally indistinguishable from uniform under the tensor isomorphism assumption. A principled approach of initialization T is to restrict to diagonal tensor initialization T_0, then scramble it by a randomly sampled action. This *strong-key sampling* eliminates weak-key attacks without requiring an increase in the field size. In particular, it enables instantiation over smaller finite fields while maintaining security guarantees [10].

4.5 Experimental comparison: naive 3TI vs integer factoring

To obtain empirical evidence on comparative hardness of 3TI problem, we implemented a toy experiment comparing a brute-force solver for the 3TI with a naive integer factoring (IF) algorithm. Note that this experiment is purely illustrative and does not represent cryptographic-scale hardness. We use toy examples for better visualization and basic demonstration only.

Problem setup. For the tensor side, we fix a prime field $\mathbb{F}_q$ and a local dimension d, and sample a random tensor

$$T \in (\mathbb{F}_q^d)^{\otimes 3}.$$

We then sample random invertible matrices

$$A, B, C \in \mathrm{GL}(d, \mathbb{F}_q),$$

and form

$$T' = (A \otimes B \otimes C)(T).$$

The solver attempts to recover a triple (A', B', C') by exhaustive enumeration of $\mathrm{GL}(d, \mathbb{F}_q)^3$. For the factoring side, we generate two random primes p, q of fixed decimal length and set

$$N = pq.$$

Note that for the brute-force factoring method, it is performed by iterating over possible divisors to test correctness.

Hyperparameter choice. To make the comparison computationally feasible, we use small parameters:

$$q = 3, \quad d = 2,$$

so that $|\mathrm{GL}(2, \mathbb{F}_3)|$ remains enumerable (otherwise it takes incredibly long time to run), and primes of fixed digit length (not binary length) for N. Both solvers are purely brute-force: the tensor solver enumerates triples in $\mathrm{GL}(d, \mathbb{F}_q)^3$, while factoring tests divisors up to $\lfloor \sqrt{N} \rfloor$.

We repeat the experiment for multiple independent runs and record the prefix average time.

Observed result. Figure 3 plots the prefix mean running time as the number of runs increases.

Analysis. The experiment shows that, under comparable toy parameters, the naive 3TI solver consistently requires significantly more time than naive factoring. Moreover, while factoring stabilizes quickly, the tensor search maintains a substantially higher average running time due to the cubic enumeration over $\mathrm{GL}(d, \mathbb{F}_q)^3$, which grows faster than the linear enumeration used in IF. Although these parameters are far below cryptographic scale, the trend suggests that tensor isomorphism exhibits rapid growth in search complexity comparing to classical IF under naive algorithms.

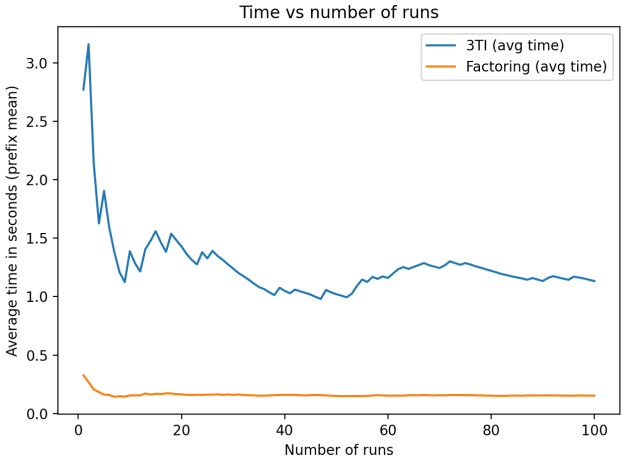

Figure 3: Hardness between 3TI and IF

5 Discussion and Conclusion

This report reviewed two algebraic approaches to public-key encryption: braid-based constructions built from conjugacy problems, and tensor-based constructions built from tensor action inversion. For braid groups, we presented representative schemes and explained how their security reduces to conjugacy search problems. We also discussed practical attack methods, which show that algebraic structure must be handled carefully.

We then introduced tensor actions and the tensor isomorphism problem as an alternative hardness assumption. Compared with matrix equivalence, higher-order tensor isomorphism appears more complex, and recent work shows that weak-key structures can be avoided through strong-key sampling. Preliminary experiments suggest that even small tensor instances exhibit rapid growth in search complexity under naive attacks.

These observations suggest several directions for future research. First, a fully formal public-key encryption scheme based on tensor action should be developed, including precise security assumptions and reductions. Second, parameter selection and strong-key generation must be studied carefully to prevent structural weaknesses. Third, efficiency and practical performance need to be evaluated for realistic dimensions and field sizes. Finally, it would be valuable to compare algebraic tensor assumptions with other post-quantum candidates in a unified framework.

In summary, braid-based systems provide important lessons about algebraic cryptography, while tensor-based constructions offer a promising and less explored direction. Further theoretical and experimental work is needed to determine whether tensor isomorphism can serve as a strong foundation for future public-key encryption.

Acknowledgements

The author would like to thank Professor Nalini Joshi for valuable suggestions, encouragement, and insightful comments during the development of this work. Her guidance and support are greatly appreciated.

References

[1] Patrick Dehornoy. Braid-based cryptography. In *Group Theory, Statistics, and Cryptography*, volume 360 of *Contemporary Mathematics*, pages 5–33. American Mathematical Society, 2004. Survey. Online copy: `https://dehornoy.lmno.cnrs.fr/Surveys/Dgw.pdf`.

[2] F. A. Garside. The braid group and other groups. *The Quarterly Journal of Mathematics*, 20(1):235–254, 1969.

[3] Iris Anshel, Michael Anshel, and Dorian Goldfeld. An algebraic method for public-key cryptography. *Mathematical Research Letters*, 6:287–291, 1999.

[4] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jong Woo Han, Jin Soo Kang, and Choonsik Park. New public-key cryptosystem using braid groups. In *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 166–184. Springer, 2000.

[5] Volker Gebhardt. A new approach to the conjugacy problem in garside groups. *Journal of Algebra*, 292(1):282–302, 2005.

[6] James Hughes and Allen Tannenbaum. Length-based attacks for certain group based encryption rewriting systems. In *SECI02 – Sécurité des Communications sur Internet*, 2002.

[7] Zhengfeng Ji, Youming Qiao, Fang Song, and Andy Yun. General linear group action on tensors: A candidate for post-quantum cryptography. In *Theory of Cryptography Conference (TCC 2019), Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 251–281. Springer, 2019.

[8] Devarshi Browne and Anand Kumar Narayanan. Easy orbits for isomorphism problems. In *Proceedings of the 65th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024.

[9] Daniele Ran and Svetla Samardjiska. Triangle structures and weak keys in tensor isomorphism cryptography. *Preprint*, 2025.

[10] Anand Kumar Narayanan. Strong keys for tensor isomorphism cryptography. In *Proceedings of MFCS 2025*, LIPIcs. Schloss Dagstuhl, 2025.

[11] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[12] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[13] V. M. Sidelnikov, M. A. Cherepnev, and V. Y. Yashchenko. Systems of open distribution of keys on the basis of noncommutative semigroups. *Doklady Akademii Nauk*, 332(5), 1993. English translation: Doklady Mathematics 48(2) (1994) 384–386.

[14] Lihua Wang, Licheng Wang, Zhenfu Cao, Eiji Okamoto, and Jun Shao. New constructions of public-key encryption schemes from conjugacy search problems. In *Inscrypt 2010*, volume 6584 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011.

[15] James Hughes and Andrew Tannenbaum. Length-based attacks for certain group based encryption rewriting systems. *Workshop SECI02 Securité de la Communication sur Internet*, 2002. Explicitly discusses reducing multiple conjugacy problems to sequential single conjugacy problems modulo centralizers.

[16] Joshua A. Grochow and Youming Qiao. Algorithms for tensor isomorphism and polynomial equivalence. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1113–1126. ACM, 2017.

Appendices

A An Example of Super Summit Set

Here we give an example in B_3 with Garside element Δ. We define x in normal form with: $x = \sigma_1\sigma_2$. From Def. 3.12, we have $\inf(x) = 0, \sup(x) = 1$ since x does not contain Δ and is itself simple. We start by conjugate x by σ_1 (this step should be replaced with cycling / decycling in practice), yields:

$$x' := \sigma_1^{-1} x \sigma_1 = \sigma_1^{-1}(\sigma_1\sigma_2)\sigma_1 = \sigma_2\sigma_1 \in [x]$$

Note that the braid $x' = \sigma_2\sigma_1$ is also simple, with left normal form

$$x' = \Delta^0\,(\sigma_2\sigma_1), \qquad \inf(x') = 0, \quad \sup(x') = 1.$$

We observe that for the conjugacy of x, its sup cannot be smaller than 1 since $x \neq 1$, and x itself is not Δ, hence:

$$\inf_s(x) = 0, \qquad \sup_s(x) = 1.$$

Therefore every conjugate $y \in [x]$ belongs to $\mathrm{SSS}(x)$ if and only if

$$\inf(y) = 0 \quad \text{and} \quad \sup(y) = 1,$$

and in particular both x and x' lie in $\mathrm{SSS}(x)$. Actually, in this example,

$$\mathrm{SSS}(x) = \{\sigma_1\sigma_2,\ \sigma_2\sigma_1\}.$$

B An Example of Length Based Attack

We illustrate a length-based attack (LBA) on a conjugacy search problem (CSP) instance, where we work in B_3 and use the reduced word length $\ell(\cdot)$ (number of letters after free cancellations) as the length function.

Let the conjugator $x := \sigma_1\sigma_2$, $a := \sigma_1\sigma_2^{-1}$ and $b := x^{-1}ax$ Since $x^{-1} = \sigma_2^{-1}\sigma_1^{-1}$, we compute:

$$\begin{aligned} b &= (\sigma_2^{-1}\sigma_1^{-1})(\sigma_1\sigma_2^{-1})(\sigma_1\sigma_2) \\ &= \sigma_2^{-1}(\sigma_1^{-1}\sigma_1)\sigma_2^{-1}\sigma_1\sigma_2 \\ &= \sigma_2^{-2}\sigma_1\sigma_2. \end{aligned}$$

The public CSP instance is (a, b), and the goal is to recover a conjugator mapping b to a.

For LBA, it tests conjugation of b by generators in $\{\sigma_1^{\pm1}, \sigma_2^{\pm1}\}$ and compare lengths. One useful move is conjugation by σ_2:

$$c_1 := \sigma_2\, b\, \sigma_2^{-1} = \sigma_2(\sigma_2^{-2}\sigma_1\sigma_2)\sigma_2^{-1} = \sigma_2^{-1}\sigma_1.$$

and in terms of reduced word length, we have:

$$\ell(b) = 4, \qquad \ell(c_1) = 2,$$

it can be verified that conjugated by σ_2 drops to the minimum length. However, $c_1 \neq a$, so one iteration does not solve the instance. Then from $c_1 = \sigma_2^{-1}\sigma_1$, we conjugate by generatros in $\{\sigma_1^{\pm 1}, \sigma_2^{\pm 1}\}$ again and we finally find σ_1 to be the correct answer:

$$c_2 := \sigma_1\, c_1\, \sigma_1^{-1} = \sigma_1(\sigma_2^{-1}\sigma_1)\sigma_1^{-1} = \sigma_1\sigma_2^{-1} = a.$$

Thus after two iterations, LBA finds a conjugator

$$(\sigma_1\sigma_2)\, b\, (\sigma_1\sigma_2)^{-1} = a,$$

so the recovered secret conjugator is $x = \sigma_1\sigma_2$.