Knot-based Key Exchange protocol

Silvia Sconza* and Arno Wildi†

University of Zurich, Zurich, Switzerland

Abstract

We propose a new key exchange protocol based on the Generalised Diffie-Hellman Key Exchange. In the latter, instead of using a group-action, we consider a semigroup action. In our proposal, the semigroup is the set of oriented knots in $\mathbb{S}^3$ with the operation of connected sum. As a semigroup action, we choose the action of the semigroup on itself through the connected sum. For the protocol to work, we need to use knot invariants, which allow us to create the shared secret key starting from the same knot represented in two different ways. In particular, we use finite type invariants. The security of the protocol is guaranteed by the hardness of decomposing knots in the semigroup.

Keywords: Generalised Diffie-Hellman Key Exchange, Public Key Cryptography, Knot Theory, Connected Sum, Finite Type Invariants, Semigroup Action, Semigroup-based Cryptography.

1 Introduction

Knots, which are ambient isotopy types of embeddings $\mathbb{S}^1 \hookrightarrow \mathbb{S}^3$ (see Figure 2 and Definition 2.1), were used by humankind since ancient times, at the latest since the invention of the shoelace. The mathematical study of knots however started with Lord Kelvin, hypothesizing that atoms are actually knots and molecules are links flowing in the aether. His collaborator Peter Tait then initiated the field of knot theory. The basic problem being: Given two knots, are they the same or not? Following the development of topology in the early 20th century, numerous knot invariants like the Alexander polynomial [39] were developed to give answers to this problem. The interest in knot theory rose when deep connections were found to the study of 3- and 4-manifolds. For example, knots were used to prove that there are exotic $\mathbb{R}^4$, i.e. manifolds that are homeomorphic but not diffeomorphic to $\mathbb{R}^4$ [14]. Jones and Witten revolutionized the field with the discovery of the Jones polynomial [19] and its relation to Quantum field theory [41] from which quantum topology emerged. These breakthroughs were followed by the discovery of Khovanov homology [21] and knot Floer homology [35], which vastly generalize the Jones and the Alexander polynomial and provide active fields of research.

In this paper we are mainly interested in two aspects of knot theory. The first is an operation called the connected sum (see Figure 5) that takes two oriented knots, cuts them open and glues

*silvia.sconza@math.uzh.ch

†arno.wildi@math.uzh.ch

them together, respecting the orientation, to produce a new oriented knot. It turns out that, with this operation, oriented knots form the abelian semigroup **Knots**. The second is a class of knot invariants (see Section 2.1.3) called finite type invariants. They have many interesting features, one of them being that they are efficiently computable.

In the last 30 years, applications of knot theory to other scientific disciplines were found. In chemistry, we can decide whether a molecule is chiral or not. Molecules can have very different properties depending on their chirality. In biology, one can study the knottedness of the DNA in a cell. And in quantum computing, one studies anyons which give naturally a braid that can be studied in knot-theoretic terms.

In the paper, we propose an application of knot theory in cryptography, in particular we propose a Key Exchange Protocol. We know of three instances where knots were considered in cryptography. After getting the fields medal, Jones was asked whether knotted antennas would help sending messages securely. Together with Przytycki they investigated the problem. More promising attempts were the protocol of Marzuoli and Palumbo in [28] and the secret-key agreement proposed by Zucker in [42]. In the first one, they are proposing a symmetric protocol, the weakness of which is that they have to agree in secret on as much information as is used to describe the message. The second one is more close to our proposal since it is also a Key Exchange protocol, but the author is using knots deriving just from braids. Moreover, the use of the Jones Polynomial allows us to break it, since it is multiplicative. We will discuss in Section 4 why it is necessary to avoid such invariants.

In cryptography, we are concerned with exchanging messages securely, i.e. in such a way that an eavesdropper outside the conversation cannot obtain the original message. To do this, the message is "reformed" using a cryptographic key and only the designated recipient can retrieve the original message again using a key (not necessarily the same one). One of the main problems facing cryptography is how to generate and exchange these keys (Key Exchange Problem). One of the most widely used ways is the Diffie-Hellman Key Exchange, proposed in [12] and described below in Protocol 2.20. The original protocol works for a finite cyclic group G (in the first proposal $G = \mathbb{F}_q^\times$), but it can be generalised using a group action or a semigroup action, obtaining the Generalised Diffie-Hellman Key Exchange, described below in Protocol 2.24. In our case, the semigroup that we consider is **Knots**. The semigroup action is the action of the semigroup on itself, through the connected sum.

In the past, several cryptosystems based on group-actions have been proposed. The most famous are the one proposed by Anshel, Anshel and Goldfeld [2] and Ko et al. [22], where the groups considered are the braid groups and the action is the conjugation, and the one proposed by Castryck et al. [10], known as CSIDH (Commutative Supersingular Isogeny Diffie-Hellman). The latter is part of Isogeny-based Cryptography; the set considered is that of $\mathbb{F}_p$-isomorphism classes of supersingular elliptic curves over $\mathbb{F}_p$, characterised by the fact that the ring of endomorphisms is an order $\mathcal{O}$ in an imaginary quadratic field. The acting group is the ideal-class group $\mathrm{cl}(\mathcal{O})$ which acts through the application of isogenies.

As we will see also in our case, the use of semigroup actions instead of group actions brings certain advantages. First of all, given the poorer algebraic structure, many attacks cannot be applied or generalised. Generic algorithms for the group case like Pollard's rho [37], Pollard's lambda [37] and Shank's baby-step-giant-step [40] require at some point the existence of inverses, which is obviously not ensured in the case of semigroups.

The choice of considering **Knots** and the semigroup action on itself brings certain advantages. First of all, referring to the aforementioned attacks, no knot admits an inverse except for the trivial

one, i.e. the unknot $\mathcal{U}$, as stated in Proposition 2.6. Furthermore, in order to propose a well-defined cryptosystem, one must find a mathematical problem which is supposed to be computationally hard. Our cryptosystem is based on the difficulty of factoring a connected sum of knots while knowing one of the two knots used (Problem 2.14). The particularity of this problem is that it admits a unique solution, thanks to Proposition 2.7.

The paper is organised as follows. In Section 2 we give all the necessary preliminaries. It is divided in two subsections: in the first one, we give the preliminaries of Knot Theory and in the second one, there are the preliminaries related to Public-Key Cryptography. In Section 3, we describe our proposed new cryptosystem, with an indication of the size of the keys. Section 4 is dedicated to the cryptoanalysis; there we analyse both generic attacks for the semigroup action problem and knot theoretic attacks. Moreover, we briefly discuss the efficiency. Finally, we give a possible choice of parameters for a 128-bit security level and we conclude with some open questions.

Acknowledgement The authors would like to thank their respective advisors Joachim Rosenthal and Anna Beliakova for their generous support and their helpful suggestions. We would like to thank Léo Ducas for useful comments on a first draft of the paper. The second author would like to thank Peter Feller, Józef Przytycki and Daniel Tubbenhauer for interesting conversations on the use of knot invariants in cryptography.

2 Preliminaries

In this section we introduce the required topics for understanding the proposed cryptosystem. A reader coming from the knot-theory-side may skip the part on knots (although one might want to refresh the memory on finite type invariants from Section 2.1.3), while a reader coming from the cryptography-side may skip the part on cryptography.

2.1 Crash course on knots

We discuss the relevant notions and results from knot theory, but leave out most of the proofs. For a general introduction to knot theory we refer to [39].

2.1.1 Basic notions and problems from knot theory

Intuitively, knots are exactly what one expects them to be. We think of them as tied ropes where we glue the ends together and allow the rope to "wiggle". Let us start by giving a mathematically precise definition.

Definition 2.1. A *knot* is described by either of the following (equivalent) definitions.

1. Knots are smooth embeddings $\mathbb{S}^1 \hookrightarrow \mathbb{R}^3$, considered up to ambient isotopy [1]
2. Knots are (finite) polygonal closed lines in $\mathbb{R}^3$, considered up to the Δ-move seen in Figure 1.

[1] Two embeddings $g, h\colon N \hookrightarrow M$ are called ambient isotopic, if there is a continuous map (called ambient isotopy) $F\colon M \times [0,1] \to M$ such that $F_0 = id_M$ and $F_1 \circ g = h$.

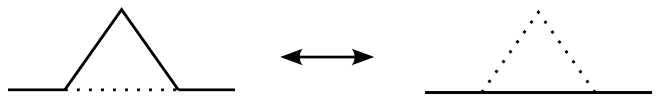


Figure 1: A local picture of the Δ-move.

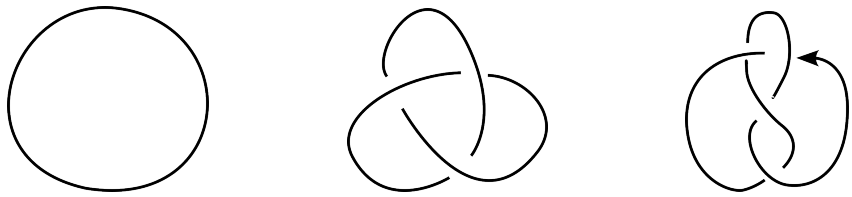


Figure 2: Some examples of knots (diagrams). The unknot $\mathcal{U}$, the (right-handed) Trefoil knot and the oriented Figure-Eight knot.

If one allows several components, one speaks of a link. Usually, we picture knots by *knot diagrams*. They are generic projections of the knot to a plane allowing only singularities of a certain kind, called *crossings*. The complexity of a knot diagram can be measured by its number of crossings. Of course, a knot has many different diagrams, depending on its position in $\mathbb{R}^3$ and the chosen projection. The *Reidemeister theorem* allows us to handle this problem.

Theorem 2.2 (Reidemeister)**.** *Two knot diagrams represent the same knot if and only if they are related by planar isotopies and a finite sequence of the Reidemeister moves, represented in Figure 3.*

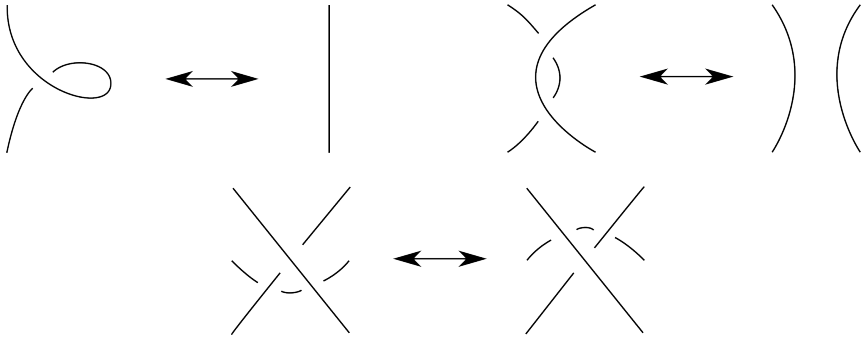


Figure 3: The three Reidemeister moves for knot diagrams.

Definition 2.3. When a knot is endowed with an orientation, it is called an *oriented knot*. There is a corresponding Reidemeister theorem for oriented knots. We distinguish two types of crossings called the positive and the negative crossing seen in Figure 4. The set of oriented knots is denoted by **Knots**.

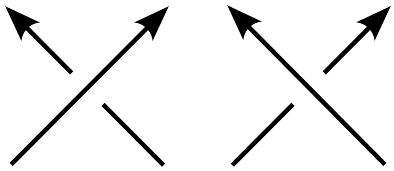


Figure 4: The positive and the negative crossing.

We consider the following operation on oriented knots.

Definition 2.4. Given two oriented knots K and K' we define the *connected sum* $K\#K'$ by cutting open the two knots and glueing the corresponding ends (given by the orientation) together as in Figure 5 to get a new knot.

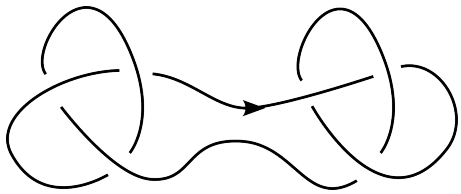


Figure 5: The connected sum of two Trefoil knots.

The connected sum of two knots is well defined, because cutting the knots at different spots results in isotopic knots. Note that it is not well defined for unoriented knots, since we have a choice, i.e. which arcs we glue together in the unoriented case. It is obvious that taking the connected sum with the unknot $\mathcal{U}$ yields an isotopic knot. The following Proposition is immediate.

Proposition 2.5. $(\mathbf{Knots}, \#, \mathcal{U})$ *form an abelian semigroup.*

We call a knot that cannot be written in a non-trivial way as the connected sum of two other knots *prime*. The semigroup of oriented knots has the following properties proven in [39].

Proposition 2.6. *Apart from the unknot, no knot has an inverse with respect to the connected sum.*

Proposition 2.7. *For any knot there is a unique (up to reordering) decomposition into prime knots.*

In general, it is hard to tell whether a knot is prime or not. But for a certain subset of knots, it is straightforward.

Definition 2.8. A knot diagram is called *alternating*, if the crossings alternate between under- and overpasses when travelling along the knot. A knot is called alternating, if it has an alternating diagram.

Not all knots are alternating. There are even prime knots which are not alternating. There are the following useful fact about alternating knots proven in [31] and [33].

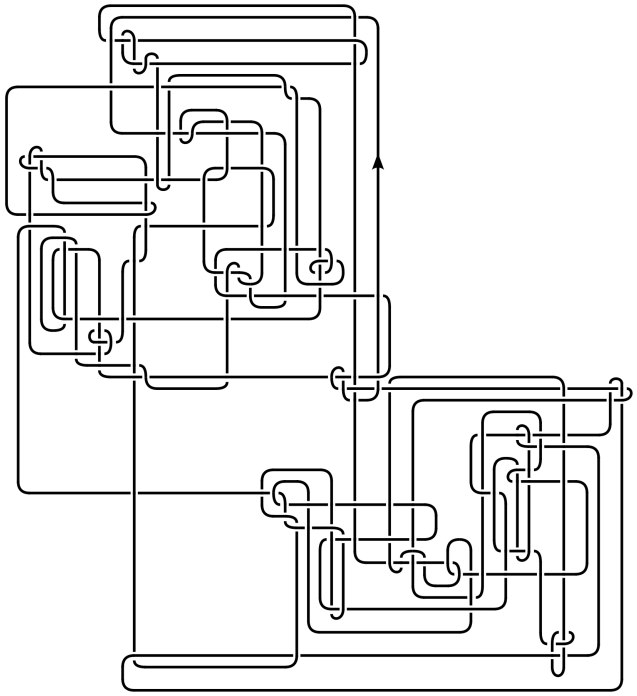


Figure 6: A randomly produced alternating prime knot with 151 crossings.

Proposition 2.9. *An alternating knot diagram represents a prime knot if and only if the diagram "looks prime" which means that there is no circle that intersects the diagram in two points such that both, in the interior and the exterior of the circle there are crossings. Moreover, such an alternating prime diagram is minimal. i.e. there is no diagram of the knot with less crossings.*

This gives a method to generate random prime knots. With Schaeffer's algorithm, one can generate a random planar graph with n vertices in $\mathcal{O}(n)$ time. This will give a graph representing a link in general, but one can take the biggest component which is a knot. There is a unique (up to mirroring) way of making the diagram alternating. Then, one can take the biggest prime summand and this will give in practice a prime knot with $\frac{n}{2}$ to n crossings. This is implemented in the "SnapPy" program. An example of a randomly produced knot is seen in Figure 6. A question to ask is, whether one gets enough knots this way, or if the set of alternating prime knots is comparably small. While Table 1 computes the number of small (alternating) prime knots, the following estimate, proven in [1], answers the previous question in our favour.

Proposition 2.10. *Let $K(n)$ be the number of knots with at most n crossings, $PK(n)$ be the number of prime knots with at most n crossings, and $APK(n)$ be the number of alternating prime knots with at most n crossings. We have the following estimate.*

$$4.45 \leq \liminf APK(n)^{\frac{1}{n}} \leq \liminf PK(n)^{\frac{1}{n}} \leq \liminf K(n)^{\frac{1}{n}} \leq 10.40.$$

In cryptography, one is usually interested in problems that are hard to solve. Topology is full of such problems and, in particular, we have examples in knot theory.

n	1	2	3	4	5	6	7	8	9	10	11	12
# n crossings prime knots	0	0	1	1	2	3	7	21	49	165	552	2176
# n crossings alternating prime knots	0	0	1	1	2	3	7	18	41	123	367	1288

13	14	15	16	17	18	19	20	21
9988	46972	253293	1388705	8053393	48266466	294130458	unknown	unknown
4878	19536	85263	379799	1769979	8400285	40619385	199631989	990623857

Table 1: The list of the number of (alternating) prime knots with n crossings.

Problem 2.11 (Unknotting Problem)**.** Given a diagram D. Does it represent the unknot?

Problem 2.12 (Recognition Problem)**.** Given two knot diagrams D and D'. Do they represent the same knot?

Problem 2.13. Given a knot diagram D of a knot K. Are there non-trivial knots K_1 and K_2 such that $K_1 \# K_2 = K$? Can you find (diagrams of) K_1 and K_2?

Problem 2.14. Given a knot diagram D of a knot $K = K_1 \# K_2$ and assume you know (a diagram of) K_1. Find (a diagram of) K_2.

It was shown in [17] that Problem 2.11 is in *NP* (which stands for Nondeterministic Polynomial time). It is the set of decision problems verifiable in polynomial time by a deterministic Turing machine. So far the best unknotting algorithms are exponential ([8], [9], [25], [13]). In [13], it was shown that Problem 2.13 and Problem 2.14 are in *NP*. However, the presented algorithm is very ineffective for knots with many crossings. The hardness of Problem 2.14 is what we use to our advantage in the construction of the cryptosystem.

2.1.2 Encoding knots

The following part is dedicated to transferring the cryptosystem into the world of computers. Since we are using knots, we need to have some way of encoding these topological objects. We will introduce the so called *PD* (planar diagram) *notation*.

PD notation. Consider a knot diagram D. We can see it as a 4-regular graph with the additional information about which strand goes over which one at every vertex. The goal now is to encode this information efficiently. Choose a starting point on an arc of the knot and the direction along the knot, given by the orientation. Start by labelling this arc with 1. The next will get labelled with 2 and so on until every arc has a number associated to it. Now, we associate to each crossing a quatern with the labels of the four arcs attached to it: we start with the label of the unique incoming undergoing arc and we conclude with the other three in counter-clockwise order. At the end, form a list with all these quaterns. This holds the information about the underlying 4-regular graph. An example is seen in Figure 7.

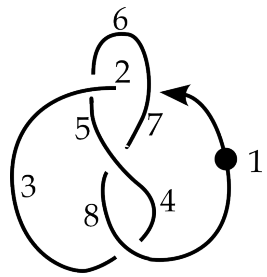


Figure 7: The PD notation of the Figure Eight knot pictured above is the following list: $[(1,6,2,7),(5,2,6,3),(3,1,4,8),(7,5,8,4)]$.

PD notation and connected sum. Notice that the PD notation in Figure 8 for the connected sum of twice the Figure Eight knot splits into two sublists where in the first one there are all the numbers between 1 and 9 and in the second one there are the the remaining ones from 9 over 16 to 1. This fact is true in general for the code of the connected sum, if the starting point for the code lies on one of the connecting arcs. If the starting point lies somewhere else, we can get it to lie on the connecting arc by shifting every number by a fixed shift modulo twice the number of crossings.

To perform the connected sum of two knots we can use the previous observation to split the first arc of the two knots and combine the list of tuples, shifting the numbers of the second list.

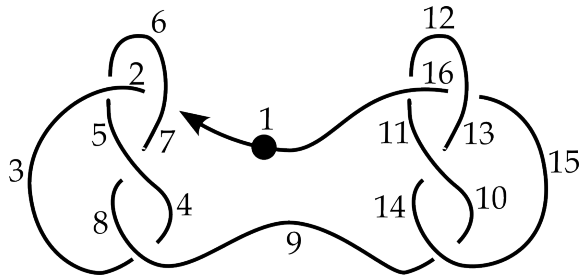


Figure 8: The PD notation for the connected sum of two Figure Eight knots is $[(1,6,2,7),(5,2,6,3),(3,9,4,8),(7,5,8,4),(9,15,10,14),(13,11,14,10),(11,16,12,1),(15,12,16,13)]$.

PD notation and Reidemeister moves. We analyze how Reidemeister moves performed on the diagram change the PD notation. A visualization is seen in Figure 9. There are several possibilities for each move, depending on the orientation of the arcs and the signs of the crossings. In the following we will show one case per move.

- Reidemeister 1+ (R1p): It adds a crossing with tuple $(i, i+1, i+1, i+2)$.
- Reidemeister 1- (R1m): It deletes a crossing with tuple $(i, i+1, i+1, i+2)$.
- Reidemeister 2+ (R2p): It adds two crossings between arcs belonging to the same region. The crossings are of the form $(i, j, i+1, j+1)$ and $(i+1, j+2, i+2, j+1)$.
- Reidemeister 2- (R2m): It deletes two crossings of the form $(i, j, i+1, j+1)$ and $(i+1, j+2, i+2, j+1)$.

– Reidemeister 3 (R3): This move changes three crossings $(j, i, j+1, i+1), (j+1, k+2, j+2, k+1), (i+1, k+1, i+2, k)$ to $(i, k+2, i+1, k+1), (j+1, i+1, j+2, i+2), (j, k+1, j+1, k)$ (see Figure 9).

This gives us the tools to apply random Reidemeister moves to a knot diagram using the PD notation, because we can see from the code where we can do which move. For this, we use a weight system that assigns weights to (R1p,R1m,R2p,R2m,R3) according to which we randomly decide which kind of move to perform. An example of this is seen in Figure 10.

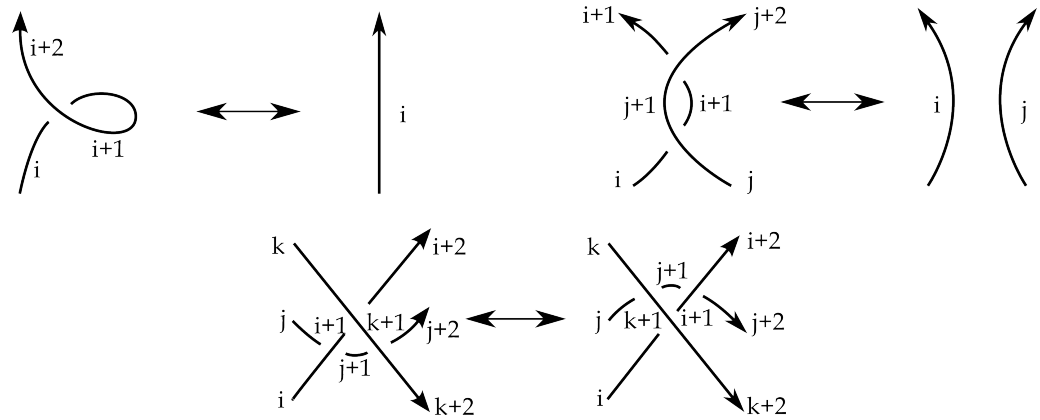


Figure 9: A visualization of how the PD notation transforms under the Reidemeister moves.

PD notation and coding. It is important to note that PD notation is supported by "SnapPy" and the Mathematica package "KnotTheory" from Bar Natan and Morrison [4].

2.1.3 Finite type invariants

To take knots apart we use functions defined on diagrams that do not change under planar isotopies and Reidemeister moves. These functions are called knot invariants. Prominent examples include the knot signature, the Jones polynomial, the Alexander polynomial and the finite type invariants studied in this section. Note that these invariants usually do not distinguish all knots and it is common to trade calculability of the invariant for its power to detect knots.

We now consider finite type invariants. An introduction can be found in [3]. Let V (for Vassiliev invariant) be an invariant of knots with values in $\mathbb{Z}$, i.e. a function

$$V\colon \{\textit{Knot Diagrams}\} \to \mathbb{Z}$$

that does not change under Reidemeister moves. Extend V to an invariant of oriented 1-singular knots, which are knots that have a singularity locally looking like , using the (local) formula

$$V\Big(\ \Big) = V\Big(\ \Big) - V\Big(\ \Big). \tag{1}$$

Further extend V to the set of oriented m-singular knots by using 1 repeatedly.

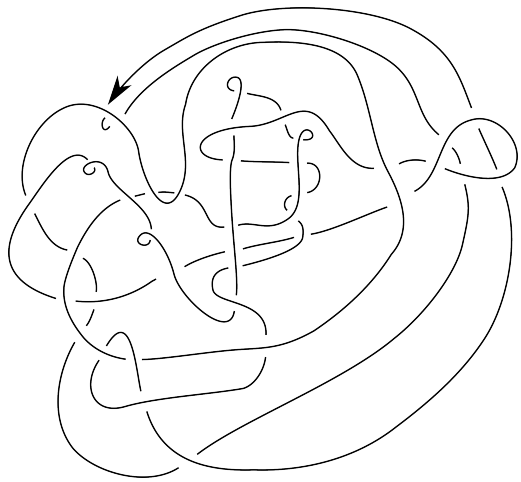


Figure 10: An example of the output of the code in Appendix B where random Reidemeister moves were performed to the connected sum of the Trefoil and the Figure Eight with weight system (R1p=5, R1m=5, R2p=8, R2m=8, R3=30).

Definition 2.15. We say that V is a *finite type invariant* (of degree m), if its extension to oriented $(m+1)$-singular knots vanishes.

Example 2.16 (Conway polynomial)**.** The (Alexander-)Conway polynomial Δ provides a source of finite type invariants. It is a polynomial (in the formal variable z) knot invariant, with value 1 on the unknot, satisfying the following local (skein) relation

$$\Delta(\times) - \Delta(\times) = z \cdot \Delta(\,)(\,)$$

It is immediate from the definition that the coefficient of z^m in the Conway polynomial is a finite type invariant of degree m.

In some sense, finite type invariants forget a lot of data. They carry a "finite" amount of information in the sense of Theorem A.3. Nevertheless there is the following conjecture.

Conjecture 2.17. The set of all finite type invariants distinguishes knots.

The following property of finite type invariants is crucial for the purpose of our paper. We postpone the proof to Appendix A.

Corollary 2.18. *A finite type invariant of degree m can be computed in polynomial time $\mathcal{O}(c^m)$ (depending on the number of crossings c).*

The proof will give an explicit way of constructing any finite type invariant. Parts of this are already encoded in Bar-Natan's, Amir-Khosravi's, and Sankarans Program "VasCalc". In Table 2, we see the list of the number of linearly independent finite type invariants.

Remark 2.19. Finite type invariants are supported in the "KnotTheory" package of Mathematica [4].

m	0	1	2	3	4	5	6	7	8	9	10	11	12
#Degree m finite type invariants	1	1	2	3	6	10	19	33	60	104	184	316	548

Table 2: The list of the number of linearly independent finite type invariants of a given degree m.

2.2 Crash course on cryptography

The aim of this section is to introduce the notions of cryptography, particularly relating to key exchanges, which we will use in the main part of the article.

The purpose of cryptography is to find ways (protocols) to communicate securely, assuming the presence of eavesdroppers. In particular, we want to transform our messages (*encryption phase*) in such a way that opponents will find it to be unintelligible text and only the predestined receiver will be able to trace the original message (*decryption phase*).

In order to carry out encryption and decryption, we need so-called *cryptographic keys*. This brings us to the *Key Exchange Problem*: how can two parties exchange keys in such a way as to establish a secure communication channel? This process is called the *key exchange protocol* and, in the next section, we will propose one based on knots.

There are two main types of cryptography: *symmetric-key* cryptography (or *single-key* cryptography) and *asymmetrical* cryptography (or *public-key* cryptography). In the first one, the two parties (Alice and Bob) use the same secret key to encrypt and decrypt the message. The main problem in this case is that they need to communicate this key to each other via a secure channel. The commonly used symmetric cryptosystem is the Advanced Encryption Standard (AES) [11, 18].

In public-key crytography, there are two keys involved: a public one known to everybody and a private one known only to the owner; depending on the algorithm, one will be used to encrypt and the other to decrypt. One of the earliest and most famous examples of Public-Key Exchange (PKE) is the *Diffie-Hellman Key Exchange* [12], proposed by Diffie and Hellman in 1976 and described below in Protocol 2.20. It allows the two parties to establish a shared secret key over an insecure channel, i.e. even in the presence of eavesdroppers who can monitor the channel.

Protocol 2.20 (Diffie-Hellman Key Exchange)**.**

1. Alice and Bob publicly agree on a cyclic finite group G and a generator g.
2. Alice chooses $a \in \{1, \dots, \mathrm{ord}(G)\}$, computes g^a and sends it to Bob. Her secret key is a.
3. Bob chooses $b \in \{1, \dots, \mathrm{ord}(G)\}$, computes g^b and sends it to Alice. His secret key is b.
4. Alice computes $(g^b)^a = g^{ba}$.
5. Bob computes $(g^a)^b = g^{ab}$.

The secret common key is $g^{ba} = g^{ab}$.

The security of the system is based on the following hard problem:

Problem 2.21 (Diffie-Hellman Problem (DHP))**.** Let G be a finite cyclic group and let g be a generator. Given g^a and g^b, find g^{ab}.

We will say that a mathematical problem is *easy* if there exists a polynomial-time algorithm which can solve it. If there are no deterministic or probabilistic polynomial-time algorithms that can solve it, we will call it *hard*.

Actually, if someone is able to solve the Discrete Logarithm Problem (DLP): *given G an abelian group and $a, b \in G$ such that $b = a^m$ for some $m \in \mathbb{Z}$, find $0 \leq n < \operatorname{ord}(G)$ such that $a^n = b$*, they can also solve the DHP.

Remark 2.22. In a cryptosystem, we need the computations required for implementation to be feasible, and those needed to break it to be not. In the Diffie-Hellman Key Exchange 2.20, we have that g^a can be computed in $\mathcal{O}(\log a)$ group multiplications, while the best algorithm to solve the DLP requires $\mathcal{O}(\sqrt{\operatorname{ord}(G)})$.

Notice that we define the Diffie-Hellman protocol using the group action of $\mathbb{Z}^{\times}_{\operatorname{ord}(G)}$ over G given by

$$\begin{aligned} \mathbb{Z}^{\times}_{\operatorname{ord}(G)} \times G &\to G \\ (n, g) &\mapsto g^n. \end{aligned}$$

Therefore it is possible to naturally extend the Diffie-Hellman protocol from a generic group action and even from a semigroup action, as shown in [32].

Definition 2.23. Let G be a semigroup and let S be a set. The semigroup G *acts on* S if there exists a map

$$\begin{aligned} G \times S &\to S \\ (g, s) &\mapsto g \cdot s, \end{aligned}$$

satisfying $(gh) \cdot s = g \cdot (h \cdot s)$ for all $g, h \in G$ and all $s \in S$.

If the semigroup is abelian, the map is called a *G-action* on the set S.

We can then define the *Generalized Diffie-Hellman Key Exchange*.

Protocol 2.24 (Generalized Diffie-Hellman Key Exchange)**.**

1. Alice and Bob publicly agree on a G-action on a finite set S and an element $s \in S$.
2. Alice chooses $a \in G$, computes $a \cdot s$ and sends it to Bob. Her secret key is a.
3. Bob chooses $b \in G$, computes $b \cdot s$ and sends it to Alice. His secret key is b.
4. Alice computes $a \cdot (b \cdot s)$.
5. Bob computes $b \cdot (a \cdot s)$.

The secret common key is $a \cdot (b \cdot s) = (ab) \cdot s = (ba) \cdot s = b \cdot (a \cdot s)$.

In order to obtain a secure cryptosystem, we need to choose an action that makes the following mathematical problem hard.

Problem 2.25 (Diffie-Hellman Semigroup Action Problem (DHSAP))**.** Let G be an abelian semigroup, S a finite set and $\cdot$ a G-action on S. Given $x, y, z \in S$ such that $y = g \cdot x$ and $z = h \cdot x$ for some $g, h \in G$, find $(gh) \cdot x$.

Notice that, on the other hand, in order to be able to calculate the key, we need the action to be computationally feasible, i.e. we need to be able to easily calculate every $g \cdot s$ and also every multiplication in G.

There is a generalised version of the DLP which use a generic semigroup action, the so-called Semigroup Action Problem.

Problem 2.26. (Semigroup Action Problem (SAP)) Let G be an abelian semigroup, S a finite set and $\cdot$ a G-action on S. Given $x, y \in S$ such that $y = g \cdot x$ for some $g \in G$, find $h \in G$ such that $y = h \cdot x$.

As with the DLP and the DHP, if one is able to solve the SAP, then he can automatically solve the DHSAP; indeed, let $\tilde{g} \in G$ such that $g \cdot x = \tilde{g} \cdot x$, therefore we could solve the DHSAP computing

$$\tilde{g} \cdot (h \cdot x) = (\tilde{g}h) \cdot x = (h\tilde{g}) \cdot x = h \cdot (\tilde{g} \cdot x) = h \cdot (g \cdot x) = (hg) \cdot x = (gh) \cdot x.$$

It is still an open question if the DHSAP and the SAP are equivalent, therefore all the attacks considered attempt to solve the Semigroup Action Problem.

Remark 2.27. If we are considering a semigroup G, we can always take $S = G$ and consider as a semigroup action just the semigroup operation. This is actually what we will do in the next section with $S = G = \mathbf{Knots}$.

3 Cryptosystem

The proposed cryptosystem is built by using the semigroup of oriented knots in the (Generalized Diffie-Hellman Key Exchange) Protocol 2.24 with the choices as in Remark 2.27. The security of the protocol is ensured by the fact that the general Problem 2.26 in our case translates to Problem 2.14, which is believed to be hard.

Protocol 3.1 (Knot-based Diffie-Hellman)**.** All knots in this protocol are presented in PD notation, as described in Subsection 2.1.2.

1. Alice and Bob publicly agree on a (randomly generated) alternating prime knot K, a positive integer n and a finite type invariant V taking values in $\mathbb{Z}$.

2. Alice chooses a (randomly generated) alternating prime knot A with at most n crossings, computes $A\#K$, complexifies the obtained knot by applying random Reidemeister moves and sends it to Bob. Her secret key is A.

3. Bob chooses a (randomly generated) alternating prime knot B with at most n crossings, computes $B\#K$, complexifies the obtained knot by applying random Reidemeister moves and sends it to Alice. His secret key is B.

4. Alice computes $V(A\#(B\#K))$.

5. Bob computes $V(B\#(A\#K))$.

The secret common key is $V(A\#B\#K) = V(B\#A\#K) \in \mathbb{Z}$.

There are two things about this protocol that are worth dwelling on. At first, there is the fact that we complicate the knot $A\#K$ (respectively $B\#K$). The reason is that otherwise, it is very easy to see where the connected sum was made and we can decompose the knot by looking at its PD notation. The other thing that makes our protocol not quite a standard Diffie-Hellman protocol is that, in the end, we compute a knot invariant. The reason is that even though we know that Alice and Bob share the same knot $A\#B\#K = B\#A\#K$, they have very different presentations of that knot. Thus, the presentations themself are useless for doing encryption, so we compute a knot invariant to get an integer, which is the same for both Alice and Bob.

Notice that the bottleneck of the protocol is the computation of the finite type invariant, which requires $\mathcal{O}(c^m)$ operations, where c is the number of crossings of the final knots $A\#B\#K$ and $B\#A\#K$ (recall that in their description these two knots can have a different number of crossings) and m is the degree of the chosen finite type invariant V. The other operations that we are doing are a juxtaposition of integer strings to perform the connected sum and addition/removal of integers in strings to apply Reidemeister moves.

Key size. Assume that in the complexification phase of the knot, both Alice and Bob apply random Reidemeister moves until they obtain an equivalent knot with at most $2n$ crossings (recall that we know exactly how many crossings we are adding/removing with each move).

In order to describe a knot with n crossings, we need all the positive integers from 1 to $2n$, just to enumerate all the edges. Following the encoding procedure, we have to write each integer between 1 and $2n$ twice in the string of integers, since each edge is related to two crossings. Recall that we need at most (up to a constant)

$$\sum_{i=1}^{\lfloor \log_2(n) \rfloor} i2^{(i-1)} - 1 = 2^{\lfloor \log_2(n) \rfloor} \lfloor \log_2(n) \rfloor - 2^{\lfloor \log_2(n) \rfloor} \simeq n(\log_2(n) - 1) \text{ bits}$$

to describe all the integers from 1 to n.

Summing up, given a knot with n crossings, we need roughly $2n(\log_2(2n) - 1)$ bits to describe all the integers from 1 to $2n$ that represent the edges and, since we need to use each of them twice, we need roughly $4n(\log_2(2n) - 1)$ bits to describe Alice's private key and Bob's private key respectively.

About the private common key, recall that it is an integer number that we obtain computing a finite type invariant of a knot with at most $3n$ crossings, since K as at most n crossings and both Alice and Bob complexify their secret knots to obtain an equivalent knot with at most $2n$ crossings. Therefore, thanks to Corollary A.4, we need at most $\lfloor 3n \log_2(M) \rfloor$ bits to describe the common secret key.

4 Cryptanalysis

In the following, we analyse the security and the efficiency of the proposed protocol. We explain how other invariants than the ones of finite type fail and discuss a similar, but in some sense very different, protocol of Marzuoli–Palumbo [28].

4.1 Security analysis

We discuss several possible attacks on the proposed protocol and either explain why they fail or give our opinion on how likely they would work to break the protocol.

Notice that the underlying mathematical problem is the following one.

Problem 4.1. Given V a finite type invariant and the values $V(K)$, $V(A\#K)$ and $V(B\#K)$, find $V(A\#B\#K)$.

Notice that if, for example, we would have chosen the Jones polynomial [19] as our knot invariant the protocol would break. This is because the Jones polynomial J is multiplicative with respect to the connected sum, so one can compute $J(A) = \frac{J(A\#K)}{J(K)}$ and $J(B) = \frac{J(B\#K)}{J(K)}$, which leads to $J(A\#B\#K) = J(A)J(B)J(K)$. Thus, we want to avoid knot invariants which have a connected sum formula. We don't know of any such general formula for finite type invariants. But it may be (and we actually know it for some of them as discussed in Section 4.3) that the specific finite type invariant we choose has such a formula. The strength of the proposed protocol however then lies in the fact that we are free to choose another finite type invariant for which we know no connected sum formula.

The study of an attack for a given finite type invariant is far from exhaustive. It is possible that some of them allow ad hoc attacks and should therefore be excluded.

There is a related mathematical problem, which is the SAP 2.26 on **Knots**.

Problem 4.2. Given two knots K and K', construct a knot A (if it exists) such that $K' = A\#K$.

Notice that, solving the previous problem allows an attacker to solve the underlying problem 4.1. Indeed, given K and $A\#K$ as in the protocol, if he finds A, since he also knows $B\#K$, he can compute $A\#B\#K$; at this point, he only has to calculate the finite type invariant of $A\#B\#K$ in order to get the secret shared key.

In the following, we discuss several possible attacks on the SAP on **Knots**. We divide them into two families: those related to the generic SAP and those that explicitly use the Knot Theory.

4.1.1 Generic attacks for the semigroup action problem

Here we analyse the generic attacks on the SAP described in [30] and [29].

Feasibility of the SAP. In general, when we try to solve the SAP, we are looking for an $h \in G$ such that $g \cdot x = h \cdot x$, not necessarily for the specific g. For this reason, we are interested in the following set

$$G_{x,g} = \{h \in G \mid h \cdot x = g \cdot x\}.$$

The parameters G, S and x need to be chosen in such a way that $|G_{x,g}|$ is small with respect to $|G|$. In our case we have $G = S = \mathbf{Knots}$, $x = K \in S$ and the action is just the connected sum. Thanks to Proposition 2.7, we know that, given K and $K'\#K$, the knot K' is unique, therefore $G_{K,K'} = \{K'\}$ and $|G_{K,K'}| = 1$ for all $K, K' \in \mathbf{Knots}$.

Structure of the semigroup. Various attacks on DH realised through a group action are known and it is therefore legitimate to ask whether some of them can also be applied or generalised to

semigroup actions. First, notice that we can partition the semigroup G, following the notation in [29], as $G = G_0 \cup G_1$, where

$$G_1 = \{g \in G \mid g^{-1} \text{ exists}\} \quad \text{and} \quad G_0 = G \smallsetminus G_1.$$

Practically, if G has a large subgroup, it can be a problem. Indeed, we can try to solve the SAP in G_0 by brute force, with an exhaustive search. If we found no solutions, then we can restrict the SAP to G_1, which is a group. Now all the attacks that we know for groups are applicable. In our case, this strategy does not apply, because $G_1 = \{\mathcal{U}\}$, thanks to Proposition 2.6.

4.1.2 Knot theoretic attacks

Attacking the basic structure and the complexification phase. In our proposed protocol there are three steps which need our attention. Taking the connected sum, complexifying the diagram, and the evaluation via a knot invariant. We have already discussed above, at the begginning of the section, the choice of the finite type invariant. The connected sum poses no problems for security. Provided that the knot is in a general position, it is commonly accepted that decomposition as in Problem 2.14 is very hard. However, if you just take the connected sum and leave the knot as it is, it is easy to decompose it by looking at its PD notation. So the question becomes, whether the algorithm using random Reidemeister moves achieves a position for the knot which is random enough that one cannot decompose the knot easily. This is still an open question.

Brute force attack. Of course, the first attack that one can always try is the brute force attack, which means that the attacker has to compute $A'\#K$, for all $A' \in$ **Knots** with at most n crossings, until he finds the correct one, which is $A\#K$. But recall that he doesn't have a way to compare directly $A'\#K$ with $A\#K$, since the second one is complexifyied in the protocol and Problem 2.12 is hard. The best thing that he could do is computing a fixed invariant of $A'\#K$ and $A\#K$ and check if it is the same. In general, in order to avoid brute force attacks, it is sufficient to set the parameters of the cryptosystem appropriately. We will give an example of a possible choice of parameters in Subsection 4.4. But recall that we don't have a computable complete invariant, so it could happen that more than one knot A' satisfy that $A'\#K$ and $A\#K$ have the same fixed invariant. In that case, the attacker could change the invariant and compute it only for previously acceptable candidates or for each of them (if they are few) he can compute the finite type invariant of $A'\#B\#K$ and check which one works as the cryptographic key.

Problem with small knots. One thing to keep in mind is that knots with up to around 15 crossings can be taken apart with knot invariants. This forces us to use bigger knots. Otherwise one could compare the invariants of the public knots to a table of knots with their invariants and try the corresponding knot. The number of (alternating prime) knots however increases very fast as seen in Proposition 2.10, which means that this procedure is not possible for bigger knots.

Attack using invariants with a connected sum formula. If the knots chosen by Alice and Bob in the protocol are sums of smaller knots, there is the following attack. In Subsection 4.3.2 we will see that a lot of knot invariants have a connected sum formula. Take for example the Alexander polynomial Δ. It satisfies $\frac{\Delta(A\#K)}{\Delta(K)} = \Delta(A)$. Hence, one can get the invariant of the secret knot. If this knot is composed of small knots, one can factor $\Delta(A)$ and only consider prime knots with

Alexander polynomial equal to one of the factors (including the constant polynomial 1). This will lead to a combinatorial problem that is more approachable, given that one can compute the Alexander polynomial on all prime knots with a given crossing number. This attack also generalizes to other knot invariants with a connected sum formula. This attack is the reason why we work with prime knots in the protocol.

Attack for braid group cryptography. Knots and braids (see Figure 11) are intimately related by the Alexander and Markov Theorem proven in [7]. The basic idea is that closing up a braid yields a knot (or a link). But the group structure on braids is not related to the semigroup structure on knots. There are cryptographic protocols involving braid groups (see [22, 15]) which are now broken. They were successfully attacked by using the faithful Lawrence representation of the braid group, where one could transfer the underlying problem into the world of matrices (compare [26], [6], and [23]). Contrary to braid groups, we know no reasonable representation of the semigroup of knots, so attacks of this form are not available.

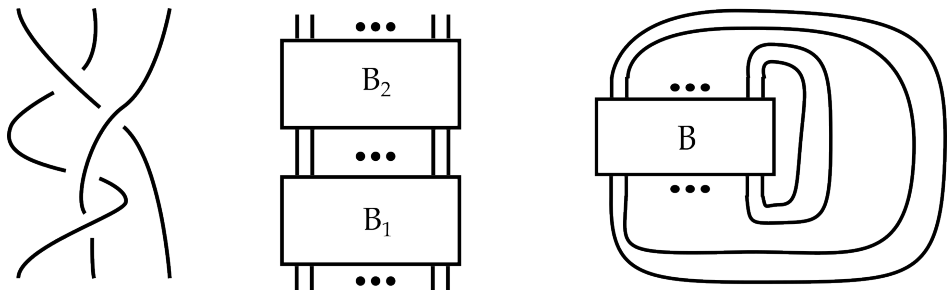


Figure 11: An example of a braid with three strands, the group structure of the braid group given by stacking and the closure of a braid.

Knots seen as graphs. In general, a linear representation of the problem could be a weakness; indeed, if we can represent the action as a matrix action on a vector space, then the SAP may be solved easily. One can think of a knot as a graph (keeping track of over- or undercrossing) and to a graph we can associate the adjacency matrix. Anyway, we can't represent the connected sum as a matrix action. Moreover, if we consider a complicated knot and the adjacency matrix associated to it, we can't obtain enough information to understand which knot it is.

We are therefore unable to represent the action as a linear one, or at least we are unable to do so by exploiting the one related to braid groups nor the point of view of graphs. This obviously does not represent an exhaustive study of all possible ways in which one can try to represent the connected sum as a linear action.

4.2 Efficiency analysis

We show here that our proposed protocol can be ran in polynomial time, depending on the crossing number. We have three knots involved K, A, and B with the number of crossings c_K, c_A, and c_B. Taking the connected sum is an easy operation. Complicating the diagram is not very costly as well, since the application of Reidemeister moves in terms of the PD notation is efficient.

This leaves the computation of the finite type invariant which uses $\sim (c_K + 2c_A + 2c_B)^m$ steps for a finite type invariant of degree m, thanks to Corollary 2.18.

4.3 Other invariants

In principal, one can choose any knot invariant for the protocol to work. Maybe, there is a knot invariant that we do not know and that fits the protocol better then the finite type ones. However there are some features of certain knot invariants that make the protocol very slow or even attackable. We discuss them here.

4.3.1 Computability

First, there is the problem of computability. Low-dimensional topology is notorious for having complicated (and powerful) constructions which are hard to calculate. This rules out basically all knot homologies like the Khovanov homology [21] and the knot Floer homology [35], as well as any invariant coming from them, like the Rassmussen invariant [38], the Υ [36] and the τ invariant [34]. Even the Jones polynomial is too costly to calculate, since its computation time grows exponentially in the number of crossings [20]. The genus, braid index and similar invariants have no algorithms to calculate them at all [39].

4.3.2 Connected sum formula

As already explained before, the protocol is attackable when the used invariant has a reasonably nice connected sum formula. Then the attacker does not have to decompose the knot itself, but rather has to find the invariant of the individual knots. We show here which invariants this excludes. The HOMFLY, Jones, and Alexander knot polynomials [27] are all multiplicative with respect to the connected sum and are therefore immediately excluded. Even the "Polynomial time knot polynomial" of Bar Natan [5] has a nice enough connected sum formula that it can be attacked. The signature is additive [39] and the number of 3-colorings is multiplicative [39].

We show here how even some finite type invariants fail. Let us consider f_m the m-th coefficient of the Conway polynomial C considered in Example 2.16, i.e. $C(K) = \sum_{m\in\mathbb{N}} f_m z^m$. The polynomial itself is multiplicative, i.e. $C(K\#A) = C(K)C(A)$. While the finite type invariant f_m is not multiplicative, it has the following formula

$$f_m(K\#A) = \sum_{i=0}^{m} f_i(K) f_{m-i}(A).$$

Thus, knowing $f_i(K)$ and $f_i(K\#A)$ for $0 \leq i \leq m$ (which are computable in not much more time than $f_m(K\#A)$) allows an attacker to solve for $f_i(A)$ (and similarly $f_i(B)$), which is enough to compute $f_m(K\#A\#B)$.

With exactly the same reasoning, one can also exclude the finite type invariants given by the m-th coefficient of $J(e^x)$ seen as a formal power series in x. This tells us that we have to choose our finite type invariant carefully enough to avoid these specific ones. But there are plenty of other finite type invariants.

4.4 Choice of Parameters

Here we will give a possible choice of parameters in order to reach a 128-bit security level, which means that the attacker would have to perform at least 2^{128} operations to break the protocol. The unique parameter that we have to choose is the number of crossings n. As far as we know, the best attack is the brute force attack described above in this chapter, since the knots chosen by Alice and Bob are prime.

So we need to look at the number of alternating prime knots with at most n crossings. For this we look at Table 1 and Proposition 2.10. From Table 1, we know $APK(21)$, the number of alternating prime knots with crossing number 21. Assuming exponential growth after that as in Proposition 2.10, we have that $n = 67$ is a suitable parameter in order to reach 128-bit security level, since

$$2^{128} < APK(21) \cdot 4.45^{46}.$$

5 Conclusion and further development

In the paper, we propose a new Key-Exchange protocol based on the Generalised Diffie-Hellman Key Exchange protocol. We use the semigroup action given by

$$\begin{aligned} \#\colon \mathbf{Knots} \times \mathbf{Knots} &\to \mathbf{Knots} \\ (K_1, K_2) &\mapsto K_1 \# K_2, \end{aligned}$$

where # represents the connected sum of two knots.

Since in our protocol Alice and Bob get the same connected sum of three knots, but represented in two different ways, in the last step we need to compute an invariant of this knot in order to obtain the same shared secret key. We studied the different possibilities and concluded that the best choice are finite type invariants, since they give us a bounded positive integer, they can be computed in polynomial type and they do not admit a connected sum formula.

Furthermore, after the cryptanalysis, the best attack is the brute force attack and, based on this, we propose a possible choice of parameters for a 128-bit security level.

Open problems. The following problems are still open and are interesting for a future work:

- A much deeper study of finite type invariants is needed. We should study each of them for degree $m \geq 3$, to understand which is the most suitable one. It could be that some of them admit a connected sum formula, which means that we have to exclude them. We must also exclude those that only admit too few integer values. This research also is needed to understand which degree m is most suitable. This choice is fundamental, since the computational complexity of a finite type invariant of degree m is exactly $\mathcal{O}(c^m)$, where c is the number of crossings of the knot. This computation is the bottleneck of the protocol.
- As already mentioned, the weak point in our protocol is the use of the knot invariant. Even though finite type invariants seem to work in principal, probably they make the protocol too slow. So the main goal would be to replace them with a better invariant, i.e. an invariant with the same good properties as the finite type ones (no connected sum formula), but that could be computed in less time.

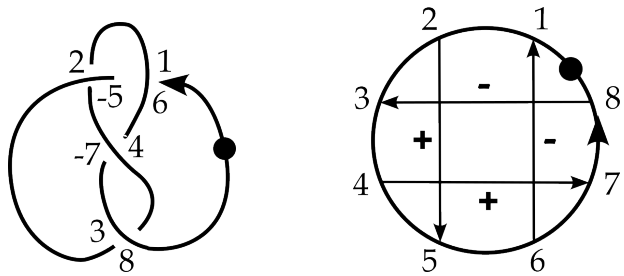


Figure 12: The Gauss diagram for the Figure Eight knot.

- Another unanswered question is: how many times do we have to apply Reidemester moves to get an equivalent knot that looks as random as possible?
- The choice of creating random alternating prime knots requires further investigation: does it weaken the protocol, if one works with alternating knots?
- Finally, no attempt has yet been made to implement our protocol.

A Finite type invariants are computable in polynomial time

The aim of this appendix is to show how one can compute finite type invariants in polynomial time. Additionally, we will see a way to construct any finite type invariant.

A.1 Gauss diagrams

We first consider Gauss diagrams. They are closely related to PD notation in the sense that the former are a visualization of the latter.

Start with a knot diagram with n crossings. Label its crossings starting at a basepoint on the knot, following the orientation, with the numbers 1 to $2n$ such that every crossing has two numbers assigned to it. To construct the Gauss diagram, we draw a circle with $2n$ dots with labels from 1 to $2n$ and connect the points belonging to the same crossing with a line and decorate it using the sign of the crossing. In addition, we give an orientation to the line to indicate which strand passed over the other one. See an example in Figure 12. We can consider arbitrary Gauss diagrams which do not need to come from knots. In fact, there are examples of Gauss diagrams that cannot be realised as knot in $\mathbb{R}^3$ without introducing singularities.

Definition A.1. The *space of Gauss diagrams* $\mathcal{D}$ is the $\mathbb{Q}$ vector space spanned by all Gauss diagrams.

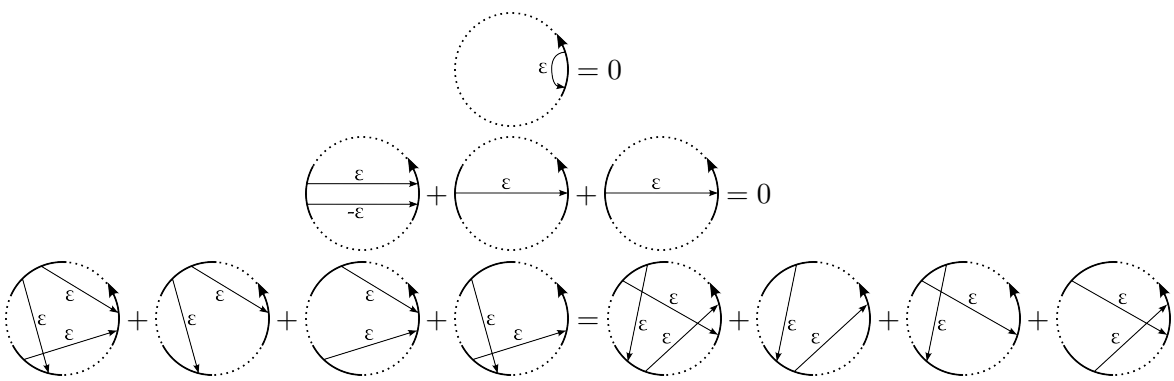
A *Gauss subdiagram* D' of D is a diagram in which some arcs of D have been deleted and we say $D' \subset D$. We can define the following endomorphism:

$$\begin{aligned} \phi\colon \mathcal{D} &\to \mathcal{D} \\ D &\mapsto \sum_{D_i \subset D} D_i \ . \end{aligned}$$

A.2 The Polyak space and the proof

Clearly, ϕ doesn't yield a knot invariant. Just introducing more crossings in a diagram changes its image under ϕ. To make ϕ an invariant, we need to introduce relations in the space of Gauss diagrams $\mathcal{D}$.

Definition A.2. The *Polyak space* $\mathcal{P}$ is defined to be $\mathcal{A}$ with the following relations:



The space $\mathcal{P}_m$ is obtained by setting all diagrams with more than m arcs to zero. It yields a finite dimensional vector space. The restriction of ϕ to $\mathcal{P}_m$ is denoted by ϕ_m. It is a universal finite type invariant in the sense of the next theorem.

Theorem A.3 ([16])**.** *V is a finite type invariant of degree m (with values in $\mathbb{Q}$) if and only if $V = f \circ \phi_m$, where $f\colon \mathcal{P}_m \to \mathbb{Q}$ is a linear functional.*

From this we can prove Corollary 2.18.

Proof of Corollary 2.18. To compute the map ϕ_m one needs to compute all $\sum_{i=1}^{m} \binom{c}{i}$ Gauss subdiagrams, which requires the stated polynomial amount of operations of Gauss diagrams:

$$\sum_{i=1}^{m} \binom{c}{i} \sim \sum_{i=1}^{m} c^i \sim c^m.$$

The evaluation of these subdiagrams via f doesn't add computational cost. □

Note that the previous proof allows us to construct any finite type invariant and thus, can be seen as a blueprint to construct finite type invariants. One can calculate a basis of the finite dimensional vector space P_m and then choose any linear functional on this basis. Note that the finite type invariant in principle takes values in $\mathbb{Q}$. It is more convenient to have an invariant that takes values in $\mathbb{Z}$, which is easily obtained by scaling the invariant by the least common multiple of the divisors of the values on the basis of Gauss subdiagrams, such that they evaluate to integers. The following corollary is immediate.

Corollary A.4. *Let the knot K have a diagram with c crossings and let V be a finite type invariant of degree m. Define $M \coloneqq \max_{D \text{ Gauss diagram}} V(D)$. We have the following bound:*

$$|V(K)| \leq M \cdot 2^c.$$

B Mathematica code for the complication algorithm

The code for the function "Randomeister" which performs random Reidemeister moves to a knot in PD notation was written by Kutluay [24] and modified by us.

```
Needs["KnotTheory`"];
ConnectedSumFigureEightTrefoilPD :=
  PD[X[20, 1, 21, 2], X[2, 11, 3, 12], X[3, 22, 4, 23],
   X[4, 10, 5, 9], X[16, 5, 17, 6], X[25, 6, 26, 7], X[7, 14, 8, 15],
   X[8, 23, 9, 24], X[10, 22, 11, 21], X[19, 13, 20, 12],
   X[13, 19, 14, 18], X[24, 16, 25, 15], X[26, 17, 1, 18]];
Upby2[m_, r_Integer] := If[m > r, m + 2, m];

Downby2[m_, r_Integer] := If[m > r, m - 2, m];

MaxArc[K_PD] := 2*Length[K];

Fig8PDs = {PD[X[1, 1, 2, 2]], PD[X[2, 1, 1, 2]], PD[X[2, 2, 1, 1]],
   PD[X[1, 2, 2, 1]]};

Fig8Q[K_PD] := MemberQ[Fig8PDs, K];

Fig8SpecialCase[r_Integer] :=
  If[r == 1, {X[1, 1, 2, 2] -> X[3, 1, 4, 4],
    X[2, 1, 1, 2] -> X[4, 3, 1, 4], X[2, 2, 1, 1] -> X[4, 4, 1, 3],
    X[1, 2, 2, 1] -> X[3, 4, 4, 1]}, {X[1, 1, 2, 2] -> X[1, 1, 2, 4],
    X[2, 1, 1, 2] -> X[4, 1, 1, 2], X[2, 2, 1, 1] -> X[4, 2, 1, 1],
    X[1, 2, 2, 1] -> X[1, 4, 2, 1]}];

R1Type[r_Integer, t_Integer] :=
  Switch[t, 1, X[r + 1, r + 1, r + 2, r], 2,
   X[r, r + 1, r + 1, r + 2], 3, X[r + 1, r, r + 2, r + 1], 4,
   X[r, r + 2, r + 1, r + 1]];

Reide1onFig8[K_PD, r_Integer, t_Integer] :=
  Append[K /. Fig8SpecialCase[r], R1Type[r, t]];

Reide1plusGeneric[K_PD, r_Integer, t_Integer] :=
  Append[K /. {X[i_, j_, k_, l_] ->
           X[Upby2[i, r], Upby2[j, r], Upby2[k, r],
            Upby2[l, r]]} /. {X[r, j_, r + 3, l_] ->
          X[r + 2, j, r + 3, l]} /. {X[i_, r, k_, r + 3] ->
         X[i, r + 2, k, r + 3]} /. {X[i_, r + 3, k_, r] ->
        X[i, r + 3, k, r + 2]} /. {X[i_, MaxArc[K], k_, 1] ->
       X[i, MaxArc[K] + 2, k, 1]} /. {X[MaxArc[K], j_, 1, l_] ->
      X[MaxArc[K] + 2, j, 1, l]} /. {X[i_, 1, k_, MaxArc[K]] ->
     X[i, 1, k, MaxArc[K] + 2]}, R1Type[r, t]];

R1p[K_PD, r_Integer, t_Integer] :=
  If[Fig8Q[K], Reide1onFig8[K, r, t], Reide1plusGeneric[K, r, t]];
Reide1minusGeneric[K_PD, r_Integer] :=
  DeleteCases[K,
    crs_ /; crs == X[r, r, r + 1, r - 1] ||
      crs == X[r - 1, r, r, r + 1] || crs == X[r, r - 1, r + 1, r] ||
```

```
      crs == X[r - 1, r + 1, r, r]] /. {X[i_, j_, k_, l_] ->
     X[Downby2[i, r], Downby2[j, r], Downby2[k, r], Downby2[l, r]]};
Reide1minusOnArc1[K_PD] :=
  DeleteCases[K,
        crs_ /; crs == X[1, 1, 2, MaxArc[K]] ||
          crs == X[MaxArc[K], 1, 1, 2] ||
          crs == X[1, MaxArc[K], 2, 1] ||
          crs == X[MaxArc[K], 2, 1, 1]] /. {X[i_, j_, k_, l_] ->
         X[i - 1, j - 1, k - 1, l - 1]} /. {X[MaxArc[K] - 1, j_, k_,
         l_] -> X[1, j, k, l]} /. {X[i_, MaxArc[K] - 1, k_, l_] ->
       X[i, 1, k, l]} /. {X[i_, j_, MaxArc[K] - 1, l_] ->
      X[i, j, 1, l]} /. {X[i_, j_, k_, MaxArc[K] - 1] ->
     X[i, j, k, 1]};
Reide1minusOnMaxArc[K_PD] :=
  DeleteCases[K,
        crs_ /; crs == X[MaxArc[K] - 1, MaxArc[K], MaxArc[K], 1] ||
          crs == X[MaxArc[K], MaxArc[K], 1, MaxArc[K] - 1] ||
          crs == X[MaxArc[K] - 1, 1, MaxArc[K], MaxArc[K]] ||
          crs == X[MaxArc[K], MaxArc[K] - 1, 1, MaxArc[K]]] /. {X[i_,
          j_, k_, l_] -> X[i, j, k, l]} /. {X[MaxArc[K] - 1, j_, k_,
         l_] -> X[1, j, k, l]} /. {X[i_, MaxArc[K] - 1, k_, l_] ->
       X[i, 1, k, l]} /. {X[i_, j_, MaxArc[K] - 1, l_] ->
      X[i, j, 1, l]} /. {X[i_, j_, k_, MaxArc[K] - 1] ->
     X[i, j, k, 1]};
R1m[K_PD, r_Integer] :=
  If[r != 1 && r != MaxArc[K], Reide1minusGeneric[K, r],
   If[r == 1, Reide1minusOnArc1[K], Reide1minusOnMaxArc[K]]];
HasQ[m_][X[i_, j_, k_, l_]] :=
  If[m == i || m == j || m == k || m == l, True, False];
FindCrossingsWith[K_PD, m_Integer] := Select[K, HasQ[m]];
EdgeReindexing[m_Integer] := If[m > 0, 2 m - 1, -2 m];
Friend[K_PD, a_Integer, b_Integer] :=
  Module[{m = K[[a, b]], l = FindCrossingsWith[K, K[[a, b]]]},
   If[Length[l] == 1, {a,
     If[Position[K[[a]], m][[1, 1]] == b, Position[K[[a]], m][[2, 1]],
       Position[K[[a]], m][[1, 1]]]},
    If[l[[1]] === K[[a]], {Position[K, l[[2]]][[1, 1]],
      Position[l[[2]], m][[1, 1]]}, {Position[K, l[[1]]][[1, 1]],
      Position[l[[1]], m][[1, 1]]}]]];
(*Finds the matching number's coordinates for the number in the a-th \
crossing's b-th entry*)
AddEdge[X[i_, j_, k_, l_], b_Integer] :=
  Module[{m = X[i, j, k, l][[b]]},
   Switch[b, 1, -m, 2, If[l - j == 1 || j - l > 1, -m, m], 3, m, 4,
    If[l - j == 1 || j - l > 1, m, -m]]];
GetFace[K_PD, a_Integer, b_Integer] :=
  Module[{MyFace = {AddEdge[K[[a]], b]}, n, DummyX = K[[a]], A = a,
    B = b}, {A, B} = Friend[K, A, Mod[B + 1, 4, 1]];
   DummyX = K[[A]];
   n = K[[A, B]];
   Join[MyFace,
    Flatten[Reap[While[n != K[[a, b]], Sow[AddEdge[DummyX, B]];
        {A, B} = Friend[K, A, Mod[B + 1, 4, 1]];
        DummyX = K[[A]];
```

```
        n = K[[A, B]];]][[2]]]]];
FindArc[K_PD, m_Integer] :=
  Module[{L = Length[FindCrossingsWith[K, m]],
    X1 = FindCrossingsWith[K, m][[1]]},
   Switch[L,
    1, {{Position[K, X1][[1, 1]],
      Position[X1, m][[1, 1]]}, {Position[K, X1][[1, 1]],
      Position[X1, m][[2, 1]]}},
    2, {{Position[K, X1][[1, 1]],
      Position[X1, m][[1,
       1]]}, {Position[K, FindCrossingsWith[K, m][[2]]][[1, 1]],
      Position[FindCrossingsWith[K, m][[2]], m][[1,
       1]]}}]]; (*This command can be replaced by Position[K,m]*)
Fig8Faces[K_PD] :=
  If[K[[1]] === X[1, 1, 2, 2] ||
    K[[1]] === X[2, 1, 1, 2], {{-1}, {1, -2}, {2}}, {{1}, {-1,
     2}, {-2}}];
Faces[K_PD] :=
  If[Fig8Q[K], Fig8Faces[K],
   Module[{EdgeV = ConstantArray[0, 2*MaxArc[K]], i, Halfway},
     Reap[For[i = 1, i <= MaxArc[K], i++,
       Module[{DummyF1 =
          GetFace[K, FindArc[K, i][[1, 1]], FindArc[K, i][[1, 2]]],
         DummyF2 =
          GetFace[K, FindArc[K, i][[2, 1]], FindArc[K, i][[2, 2]]]},
        If[MemberQ[EdgeV[[Map[EdgeReindexing, DummyF1]]], 1],
         Goto[Halfway], Sow[DummyF1];
         EdgeV[[Map[EdgeReindexing, DummyF1]]] += 1;];
        Label[Halfway];
        If[MemberQ[EdgeV[[Map[EdgeReindexing, DummyF2]]], 1],
         Continue[], Sow[DummyF2];
         EdgeV[[Map[EdgeReindexing, DummyF2]]] += 1;];]]]][[2, 1]]];
ExtraXingsType1[a_Integer, b_Integer] :=
  Module[{A = Abs[a], B = Abs[b]},
   Switch[{Sign[a], Sign[b]}, {1, 1},
    PD[X[B + 3, A + 1, B + 4, A],
     X[B + 2, A + 1, B + 3, A + 2]], {1, -1},
    PD[X[B + 2, A, B + 3, A + 1], X[B + 3, A + 2, B + 4, A + 1]], {-1,
      1}, PD[X[B + 3, A + 1, B + 4, A + 2],
     X[B + 2, A + 1, B + 3, A]], {-1, -1},
    PD[X[B + 2, A + 2, B + 3, A + 1], X[B + 3, A, B + 4, A + 1]]]];

ExtraXingsType2[a_Integer, b_Integer] :=
  Module[{A = Abs[a], B = Abs[b]},
   Switch[{Sign[a], Sign[b]}, {1, 1},
    PD[X[A, B + 3, A + 1, B + 4],
     X[A + 1, B + 3, A + 2, B + 2]], {1, -1},
    PD[X[A, B + 3, A + 1, B + 2], X[A + 1, B + 3, A + 2, B + 4]], {-1,
      1}, PD[X[A + 1, B + 4, A + 2, B + 3],
     X[A, B + 2, A + 1, B + 3]], {-1, -1},
    PD[X[A + 1, B + 2, A + 2, B + 3], X[A, B + 4, A + 1, B + 3]]]];

ExtraXings[a_Integer, b_Integer, t_Integer] :=
  If[t == 2, ExtraXingsType2[a, b], ExtraXingsType1[a, b]];
```

```
ReNumberR2p[m_Integer, a_Integer, b_Integer] :=
  If[Abs[b] < m, m + 4, If[Abs[a] <= m <= Abs[b], m + 2, m]];

R2pShifts[K_PD, a_Integer, b_Integer] :=
  Map[Function[x, ReNumberR2p[x, a, b]], K, {2}];

R2pXFixes[K_PD, a_Integer, b_Integer] :=
  Module[{A = Abs[a], B = Abs[b], MaxArc = MaxArc[K]},
   K /. {X[A - 1, j_, A + 2, l_] -> X[A - 1, j, A, l]} /. {X[i_,
                A - 1, k_, A + 2] -> X[i, A - 1, k, A]} /. {X[i_,
               A + 2, k_, A - 1] -> X[i, A, k, A - 1]} /. {X[B + 2,
              j_, B + 5, l_] -> X[B + 4, j, B + 5, l]} /. {X[i_,
             B + 2, k_, B + 5] -> X[i, B + 4, k, B + 5]} /. {X[i_,
            B + 5, k_, B + 2] -> X[i, B + 5, k, B + 4]} /. {X[
           MaxArc + 2, j_, 1, l_] -> X[MaxArc + 4, j, 1, l]} /. {X[
          i_, MaxArc + 2, k_, 1] -> X[i, MaxArc + 4, k, 1]} /. {X[i_,
          1, k_, MaxArc + 2] -> X[i, 1, k, MaxArc + 4]} /. {X[
         MaxArc + 4, j_, 3, l_] -> X[MaxArc + 4, j, 1, l]} /. {X[i_,
        MaxArc + 4, k_, 3] -> X[i, MaxArc + 4, k, 1]} /. {X[i_, 3, k_,
        MaxArc + 4] -> X[i, 1, k, MaxArc + 4]}];

R2pGeneric[K_PD, a_Integer, b_Integer, t_Integer] :=
  Join[R2pXFixes[R2pShifts[K, a, b], a, b], ExtraXings[a, b, t]];

Fig8SpecialCaseR2[K_PD, a_Integer, b_Integer] :=
  Switch[K, PD[X[1, 1, 2, 2]],
   Switch[{Abs[a], Abs[b]}, {1, 2}, PD[X[3, 1, 4, 6]], {1, 1},
    PD[X[5, 1, 6, 6]], {2, 2}, PD[X[1, 1, 2, 6]]], PD[X[2, 1, 1, 2]],
   Switch[{Abs[a], Abs[b]}, {1, 2}, PD[X[6, 3, 1, 4]], {1, 1},
    PD[X[6, 5, 1, 6]], {2, 2}, PD[X[6, 1, 1, 2]]], PD[X[2, 2, 1, 1]],
   Switch[{Abs[a], Abs[b]}, {1, 2}, PD[X[6, 4, 1, 3]], {1, 1},
    PD[X[6, 6, 1, 5]], {2, 2}, PD[X[6, 2, 1, 1]]], PD[X[1, 2, 2, 1]],
   Switch[{Abs[a], Abs[b]}, {1, 2}, PD[X[3, 6, 4, 1]], {1, 1},
    PD[X[5, 6, 6, 1]], {2, 2}, PD[X[1, 6, 2, 1]]]]];

Reide2onFig8[K_PD, a_Integer, b_Integer, t_Integer] :=
  Join[Fig8SpecialCaseR2[K, a, b], R2pGeneric[K, a, b, t][[{2, 3}]]];

R2p[K_PD, a_Integer, b_Integer, t_Integer] :=
  If[Fig8Q[K], Reide2onFig8[K, a, b, t], R2pGeneric[K, a, b, t]];
ReNumberR2m[m_Integer, a_Integer, b_Integer] :=
  If[b < m, m - 4, If[a < m < b, m - 2, m]];

R2mGeneric[K_PD, a_Integer, b_Integer] :=
  Module[{A = Abs[a], B = Abs[b], MaxArc = MaxArc[K]},
   DeleteCases[K,
     crs_ /; HasQ[A][crs] || HasQ[B][crs], {1}] /. {X[i_, j_, k_,
       l_] -> X[ReNumberR2m[i, A, B], ReNumberR2m[j, A, B],
       ReNumberR2m[k, A, B], ReNumberR2m[l, A, B]]}];

ReNumberR2mOnArc1[m_Integer, b_Integer] :=
  If[b < m, m - 3, If[1 < m < b, m - 1, m]];
```

```
R2mOnArc1[K_PD, b_Integer] :=
  Module[{B = Abs[b], MaxArc = MaxArc[K]},
   DeleteCases[K,
         crs_ /; HasQ[1][crs] || HasQ[B][crs], {1}] /. {X[i_, j_, k_,
           l_] -> X[ReNumberR2mOnArc1[i, B], ReNumberR2mOnArc1[j, B],
           ReNumberR2mOnArc1[k, B], ReNumberR2mOnArc1[l, B]]} /. {X[
          MaxArc - 3, j_, k_, l_] -> X[1, j, k, l]} /. {X[i_,
         MaxArc - 3, k_, l_] -> X[i, 1, k, l]} /. {X[i_, j_,
        MaxArc - 3, l_] -> X[i, j, 1, l]} /. {X[i_, j_, k_,
       MaxArc - 3] -> X[i, j, k, 1]}];

R2mOnMaxArc[K_PD, a_Integer] :=
  Module[{A = Abs[a], MaxArc = MaxArc[K]},
   DeleteCases[K,
         crs_ /; HasQ[A][crs] || HasQ[MaxArc][crs], {1}] /. {X[i_, j_,
            k_, l_] ->
          X[ReNumberR2m[i, A, MaxArc], ReNumberR2m[j, A, MaxArc],
           ReNumberR2m[k, A, MaxArc],
           ReNumberR2m[l, A, MaxArc]]} /. {X[MaxArc - 3, j_, k_, l_] ->
          X[1, j, k, l]} /. {X[i_, MaxArc - 3, k_, l_] ->
        X[i, 1, k, l]} /. {X[i_, j_, MaxArc - 3, l_] ->
       X[i, j, 1, l]} /. {X[i_, j_, k_, MaxArc - 3] -> X[i, j, k, 1]}];

R2m[K_PD, a_Integer, b_Integer] :=
  If[Abs[a] == 1, R2mOnArc1[K, b],
   If[Abs[b] == MaxArc[K], R2mOnMaxArc[K, a], R2mGeneric[K, a, b]]];
R3[K_PD, a_Integer, b_Integer, c_Integer] :=
  Module[{A = Abs[a], B = Abs[b], C = Abs[c], Xings,
    NewXings = PD[X[0, 0, 0, 0], X[0, 0, 0, 0], X[0, 0, 0, 0]]},
   Xings = Union[FindCrossingsWith[K, A], FindCrossingsWith[K, B],
     FindCrossingsWith[K, C]];
   Do[NewXings[[i, j]] =
     If[MemberQ[{A, B, C}, Xings[[i, j]]],
      Xings[[Friend[Xings, i, j][[1]],
       Mod[Friend[Xings, i, j][[2]] + 2, 4, 1]]],
      Xings[[i, Mod[j + 2, 4, 1]]]], {i, 3}, {j, 4}];
   Join[DeleteCases[K, crs_ /; MemberQ[Xings, crs]], NewXings]];
R1pPairs[K_PD] :=
  Module[{list = {}, i},
   For[i = 1, i <= MaxArc[K], i++,
    AppendTo[list, {{i, 1}, {i, 2}, {i, 3}, {i, 4}}];];
   Flatten[list, 1]];

R1pNumber[K_PD] := 4*MaxArc[K];

R1mFaces[K_PD] :=
  If[Fig8Q[K], {}, Cases[Faces[K], face_ /; Length[face] == 1]];

R1mNumber[K_PD] := Length[R1mFaces[K]];

R2pOnSingleArc[K_PD] :=
  Module[{list = {}, i},
   For[i = 1, i <= MaxArc[K], i++, AppendTo[list, {i, i}];
    AppendTo[list, {-i, -i}];];
```

```
   list];

R2pOnTwoArcs[K_PD] :=
  SortBy[#, Abs] & /@ (Flatten[Subsets[#, {2}] & /@ Faces[K], 1]);

R2pTriples[K_PD] :=
  Join[Append[#, 1] & /@ Join[R2pOnSingleArc[K], R2pOnTwoArcs[K]],
   Append[#, 2] & /@ Join[R2pOnSingleArc[K], R2pOnTwoArcs[K]]];

R2pPairs[K_PD] := Join[R2pOnSingleArc[K], R2pOnTwoArcs[K]];

R2pNumber[K_PD] :=
  Module[{l = Map[Length, Faces[K]], f},
   f[n_] := 2*(Binomial[n, 2] + n);
   Plus @@ Map[f, l]];

R2mFaces[K_PD] :=
  If[Length[K] == 2, {},
   Cases[Faces[K],
    face_ /;
     Length[face] == 2 &&
      EvenQ[Position[K, Abs[face[[1]]]][[1, 2]] -
        Position[K, Abs[face[[1]]]][[2, 2]]]];

R2mNumber[K_PD] := Length[R2mFaces[K]];

R3Faces[K_PD] :=
  Cases[Faces[K],
   face_ /;
    Length[face] == 3 &&
     Length[Union[FindCrossingsWith[K, Abs[face[[1]]]],
        FindCrossingsWith[K, Abs[face[[2]]]],
        FindCrossingsWith[K, Abs[face[[3]]]]]] ==
      3 && (EvenQ[
        Position[K, Abs[face[[1]]]][[1, 2]] -
         Position[K, Abs[face[[1]]]][[2, 2]]] ||
       EvenQ[Position[K, Abs[face[[2]]]][[1, 2]] -
         Position[K, Abs[face[[2]]]][[2, 2]]] ||
       EvenQ[Position[K, Abs[face[[3]]]][[1, 2]] -
         Position[K, Abs[face[[3]]]][[2, 2]]])];

R3Number[K_PD] := Length[R3Faces[K]];
ReideNumberVector[K_PD] := {R1pNumber[K], R1mNumber[K], R2pNumber[K],
   R2mNumber[K], R3Number[K]};(*This is not called for now.*)

OneRandomRmovewithweights[K_PD, L_Listwith5weights] :=
  Module[{r1p = R1pNumber[K], r1m = R1mFaces[K], r2p = R2pPairs[K],
    r2m = R2mFaces[K], r3 = R3Faces[K], move, list, weights = L, sum,
    signlist, wlist},
   list = {r1p, Length[r1m], Length[r2p], Length[r2m], Length[r3]};
   signlist = Sign /@ list;
   wlist = signlist*weights;
   sum = Plus @@ wlist;
   wlist = wlist/sum;
```

```
  move = RandomChoice[wlist -> {1, 2, 3, 4, 5}];
  Switch[move, 1,
   Module[{arc = Random[Integer, {1, MaxArc[K]}],
     type = Random[Integer, {1, 4}]}, R1p[K, arc, type]], 2,
   Module[{face = RandomChoice[r1m]}, R1m[K, Abs[face[[1]]]]], 3,
   Module[{pair = RandomChoice[r2p], type = RandomChoice[{1, 2}]},
    R2p[K, pair[[1]], pair[[2]], type]], 4,
   Module[{face = RandomChoice[r2m]}, R2m[K, face[[1]], face[[2]]]],
   5, Module[{face = RandomChoice[r3]},
    R3[K, face[[1]], face[[2]], face[[3]]]]]];

Randomeisterwithweights[K_PD, n_Integer, L_Listwith5weights] :=
  FoldList[OneRandomRmovewithweights, K, ConstantArray[L, n]];


Weightlist = {5, 5, 8, 8, 30};
```

References

[1] Ilya Alekseev, Anatolii Moiseevich Vershik, and Andrei Valer'evich Malyutin. On the growth of the number of prime knots. *Algebra i Analiz*, 36(1):17–39, 2024.

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

[3] Dror Bar-Natan. On the Vassiliev knot invariants. *Topology*, 34(2):423–472, 1995.

[4] Dror Bar-Natan, Scott Morrison, and et al. The Knot Atlas. http://katlas.org.

[5] Dror Bar-Natan and Roland van der Veen. A polynomial time knot polynomial. *Proceedings of the American Mathematical Society*, 147(1):377–397, 2019.

[6] Stephen Bigelow. Braid groups are linear. *Journal of the American Mathematical Society*, 14(2):471–486, 2001.

[7] Joan S. Birman and Tara E. Brendle. Braids: a survey. *Handbook of knot theory*, pages 19–103, 2005.

[8] Benjamin A Burton. Maximal admissible faces and asymptotic bounds for the normal surface solution space. *Journal of Combinatorial Theory, Series A*, 118(4):1410–1435, 2011.

[9] Benjamin A Burton. The Pachner graph and the simplification of 3-sphere triangulations. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 153–162, 2011.

[10] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 395–427. Springer, 2018.

[11] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[12] Whitfield Diffie and Martin Hellman. New directions in cryptography (1976). *IEEE Trans. Inform. Theory*, 22:644–654, 1976.

[13] Ivan Dynnikov. Arc-presentations of links: monotonic simplification. *Fundamenta Mathematicae*, 1(190):29–76, 2006.

[14] Michael H Freedman and Frank Quinn. Topology of 4-manifolds (pms-39), volume 39. 49, 2014.

[15] David Garber. Braid group cryptography. *Braids: Introductory lectures on braids, configurations and their applications*, pages 329–403, 2010.

[16] Mikhail Goussarov, Michael Polyak, and Oleg Viro. Finite-type invariants of classical and virtual knots. *Topology*, 39(5):1045–1068, 2000.

[17] Joel Hass, Jeffrey C Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *Journal of the ACM (JACM)*, 46(2):185–211, 1999.

[18] Daemen Joan and Rijmen Vincent. The design of rijndael: AES-the advanced encryption standard. *Information Security and Cryptography*, 2002.

[19] Vaughan FR Jones. A polynomial invariant for knots via von Neumann algebras. 1985.

[20] Louis H Kauffman. State models and the Jones polynomial. *Topology*, 26(3):395–407, 1987.

[21] Mikhail Khovanov. A categorification of the Jones polynomial. 2000.

[22] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jae Woo Han, Ju-sung Kang, and Choonsik Park. New public-key cryptosystem using braid groups. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*, pages 166–183. Springer, 2000.

[23] Daan Krammer. Braid groups are linear. *Annals of Mathematics*, pages 131–156, 2002.

[24] Deniz Kutluay. Randomeister. https://github.com/denizkutluay/Randomeister.

[25] Marc Lackenby. A polynomial upper bound on Reidemeister moves. *Annals of Mathematics*, pages 491–564, 2015.

[26] Ruth J Lawrence. Homological representations of the Hecke algebra. *Communications in mathematical physics*, 135:141–191, 1990.

[27] William BR Lickorish and Kenneth C Millett. The new polynomial invariants of knots and links. *Mathematics Magazine*, 61(1):3–23, 1988.

[28] Annalisa Marzuoli and Giandomenico Palumbo. Post quantum cryptography from mutant prime knots. *International Journal of Geometric Methods in Modern Physics*, 8(07):1571–1581, 2011.

[29] Gérard Maze. *Algebraic Methods for constructing One-Way Trapdoor Functions.* University of Notre Dame, 2003.

[30] Gérard Maze, Chris Monico, and Joachim Rosenthal. Public key cryptography based on semigroup actions. *Adv. in Math. of Communications 1.4*, pages 489–507, 2007.

[31] William Menasco. Closed incompressible surfaces in alternating knot and link complements. *Topology*, 23(1):37–44, 1984.

[32] Christopher Monico. *Semirings and semigroup actions in public-key cryptography.* University of Notre Dame, 2002.

[33] Kunio Murasugi. Jones polynomials and classical conjectures in knot theory. *Topology*, 26(2):187–194, 1987.

[34] Peter Ozsváth and Zoltán Szabó. Knot Floer homology and the four-ball genus. *Geometry & Topology*, 7(2):615–639, 2003.

[35] Peter Ozsváth and Zoltán Szabó. An overview of knot Floer homology. *arXiv preprint arXiv:1706.07729*, 2017.

[36] Peter S Ozsváth, András I Stipsicz, and Zoltán Szabó. Concordance homomorphisms from knot Floer homology. *Advances in Mathematics*, 315:366–426, 2017.

[37] John M Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

[38] Jacob Rasmussen. Khovanov homology and the slice genus. *Inventiones mathematicae*, 182(2):419–447, 2010.

[39] Justin Roberts. Knots knotes. *Lectures from Edinburgh Course Maths*, 415, 1999.

[40] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc., 1971*, volume 20, pages 415–440, 1971.

[41] Edward Witten. Quantum field theory and the Jones polynomial. *Communications in Mathematical Physics*, 121(3):351–399, 1989.

[42] Marc Zucker. *Studies in cryptological combinatorics.* City University of New York, 2005.