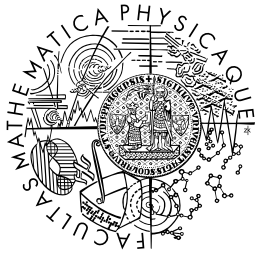

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Martin Mach

Cryptography based on semirings

Department of Algebra

Supervisor of the master thesis:	RNDr. Miroslav Korbelář, Ph.D.
Study programme:	Mathematics
Study branch:	Mathematics for Information Technologies

Prague 2019

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

At this place I would like to say thank you to my supervisor RNDr. Miroslav Korbelář, Ph.D. for the countless consultations. Further, I would like to thank to Mgr. Robert El Bashir, Ph.D. for some ideas about the topic. Finally, I am grateful to Bc. Markéta Calábková for the language and typographic correction.

Title: Cryptography based on semirings

Author: Bc. Martin Mach

Department: Department of Algebra

Supervisor: RNDr. Miroslav Korbelář, Ph.D., Department of Mathematics, FEE CTU in Prague

Abstract: Cryptography based on semirings can be one of the possible approaches for the post-quantum cryptography in the public-key schemes. In our work, we are interested in only one concrete semiring – tropical algebra. We are examining one concrete scheme for the key-agreement protocol – tropical Stickel's protocol. Although there was introduced an attack on it, we have implemented this attack and more importantly, stated its complexity. Further, we propose other variants of Stickel's protocol and we are investigating their potential for practical usage. During the process, we came across the theory of tropical matrix powers, thus we want to make an overview of it due to the use in cryptography based on matrices over the tropical algebra semiring.

Keywords: public-key cryptography, semiring, tropical algebra, semigroup action

Contents

Introduction

Cryptography is a rapidly developing discipline and there are plenty of cryptographic schemes based on different principles. The domain of most schemes (at least of most of the commonly known) are finite fields.

The cryptography is divided into two main parts – public-key cryptography and private-key cryptography. We are using both in ordinary life, usually working together to achieve some goal.

A lot of today's public-key cryptographic schemes are based on the discrete logarithm problem or the integer factorization. Unfortunately, both these problems are proposed to be broken by quantum computers in the near future. That is the reason why many researchers are interested in so-called post-quantum cryptography. And it seems that some of the schemes based on semirings instead on the finite fields are suitable candidates for post-quantum cryptography because they can stand against quantum computers (Gnilke [2014]).

In this thesis, we would like to explore some of the schemes of public-key cryptography based on semirings. We have chosen the specific semiring – *Tropical algebra*. This is a part of algebra which is popular in the last decades but it is still not fully understood. The semiring of tropical algebra consists of the set of real numbers with the special element ε and two tropical operations. The first operation is tropical addition which is done by taking the minimum of two numbers. The second operation is tropical multiplication which works as standard addition.

Tropical algebra is a relatively new and modern branch of algebra. Today, tropical algebra is mostly used in tropical geometry (Maclagan and Sturmfels [2015]) for studying algebraic varieties.

An important feature of tropical operations for the cryptography is the fact that there is no inverse of tropical addition.

The main motivation of our thesis is to pick one or two concrete proposals of the algorithms from some basic public-key cryptography areas, which are adapted for tropical algebra, and fully describe them. Both from the mathematical point of view (why are they working, their complexity) and from the practical point of view (security and possible attacks). The main aim of this thesis is to decide whether these protocols are good candidates for the public-key cryptography schemes. Alternatively, we can find out if there exists some other promising variation on these algorithms.

We hope that our work brings inspiration for further research in the direction of cryptography based on tropical algebra.

In the second chapter of our thesis, we focus mainly on an article written by Grigoriev and Shpilrain [2014] where there are two cryptographic schemes described. We will examine the one for the key agreement. An attack on this protocol is suggested in the article of Kotov and Ushakov [2018]. We would like to further explore both – the protocol and the possible attack. This protocol was only proposed without any discussion about the size of parameters and both protocol and the attack were presented without any statement of its computation complexity. We would like to replicate the attack and verify the results given by Kotov and Ushakov [2018]. But more importantly, we will focus on the estimation

of the complexity of both – the original key-agreement protocol and the attack on it.

It turned out that the proposed key-agreement protocol does not seem to be suitable for practical use even when we consider another value of the parameters than the ones suggested in Grigoriev and Shpilrain [2014]. That is why we came up with another variant of the key agreement protocol, we are calling it Fast Stickel's protocol. This protocol is less complex thus the complexity of the attacks proposed in Kotov and Ushakov [2018] is supposed to be exponential with respect to the running time of Fast Stickel's protocol.

The fourth chapter describes attacks on Fast Stickel's protocol. It uses properties of tropical matrices which are summarized in the third chapter. We found out an attack on the key agreement protocol which has exponential complexity in the size of matrices. Since too large matrices are not practical, we consider this attack as an effective one.

As we will show, even when we change the Fast Stickel's protocol to work with more complex matrices the complexity of an attack stays the same.

The last chapter is devoted to computational problems such as the discrete logarithm problem and Diffie-Hellman problems.

1. Definitions

In this chapter, we will list some of the definitions used in the rest of the thesis and we will recall the basics of cryptography needed for understanding the cryptographic parts.

1.1 Algebraic basics

Definition 1 (Ring)**.** *A ring is a set R with two binary operations $+$ (addition) and $\cdot$ (multiplication) that meet the following conditions:*

- $(R, +)$ *is an abelian group:*
  - $(a + b) + c = a + (b + c)$ *holds for all* $a, b, c \in R$
  - $a + b = b + a$ *for all* $a, b \in R$
  - $\exists 0 \in R$ *such that* $a + 0 = a$ *for all* $a \in R$
  - $\forall a \in R$ *there exists an additive inverse* $-a \in R$ *such that* $a + (-a) = 0$
- $(R, \cdot)$ *is a monoid:*
  - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ *holds for all* $a, b, c \in R$
  - $\exists 1 \in R$ *such that* $a \cdot 1 = 1 \cdot a = a$ *for every* $a \in R$
- *Multiplication is distributive:*
  - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ *holds for all* $a, b, c \in R$
  - $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$ *for all* $a, b, c \in R$

Definition 2 (Semiring)**.** *We say that $(R, +, \cdot)$ is a semiring when $(R, +, \cdot)$ differs from a ring in the fact that $(R, +)$ needs not to be an abelian group but only a commutative monoid. That means that we do not have additive inverses in general. Because of that, we need to add one more property to the semiring definition:*

- *Multiplication by 0 annihilates R:*

$$0 \cdot a = a \cdot 0 = 0$$

1.2 Tropical algebra

The concrete semiring we will work with is called the *tropical semiring* (alternatively *min-plus algebra*) due to the non-standard definition of the binary operations in it.

Definition 3 (Tropical operations)**.** *We call the operation $\oplus$ a tropical addition and the operation $\odot$ a tropical multiplication, when:*

$$x \oplus y = \min(x, y)$$
$$x \odot y = x + y$$

for all $x, y \in S \cup \{\varepsilon\}$ where $(S, +, \leq)$ is a linearly ordered commutative group and $\varepsilon \geq z$ for all $z \in S$.

The neutral element for $\oplus$ is denoted by ε. We will call it the tropical zero.

The identity element for $\odot$ is called the tropical one and is denoted by $\mathbb{1}$.

Lemma 1 (Properties of tropical operations)**.** *Both operations $\oplus$ and $\odot$ are associative and commutative, the operation $\odot$ is distributive over $\oplus$. Furthermore, the value of the neutral element for the operation $\oplus$ is ∞ and the value of $\mathbb{1}$ is 0.*

Proof. Associativity, commutativity and distribution are striaghtforward:

- $\oplus$ is associative:

  $$(a \oplus b) \oplus c = \min(\min(a, b), c) = \min(a, b, c) = \min(a, \min(b, c)) = a \oplus (b \oplus c)$$

- $\oplus$ is commutative:

  $$a \oplus b = \min(a, b) = \min(b, a) = b \oplus a$$

- $\odot$ is associative:

  $$(a \odot b) \odot c = (a + b) + c = a + (b + c) = a \odot (b \odot c)$$

- $\odot$ is commutative:

  $$a \odot b = a + b = b + a = b \odot a$$

- $\odot$ is distributive:

  $$a \odot (b \oplus c) = a + \min(b, c) = \min(a + b, a + c) = (a \odot b) \oplus (a \odot c)$$
  $$(a \oplus b) \odot c = \min(a, b) + c = \min(a + c, b + c) = (a \odot c) \oplus (b \odot c)$$

The value of the tropical one is 0, because:

$$a \odot 0 = 0 \odot a = a$$

The value of the tropical zero element is ∞:

$$a \oplus \varepsilon = \min(a, \infty) = a$$
$$a \odot \varepsilon = a + \infty = \infty + a = \varepsilon$$

□

Definition 4 (Tropical semiring)**.** *We call the structure $(S \cup \{\varepsilon\}, \oplus, \odot)$ a tropical semiring for the linearly ordered commutative group $(S, +, \leq)$.*

In most cases, we will work with the tropical algebra $(\mathbb{Z} \cup \{\infty\}, \oplus, \odot)$.

We do not have an inversion of addition in tropical algebra, but we can still invert multiplication, which we denote as $\oslash$. We can also define a tropical exponentiation in an intuitive way:

$$a \oslash b = a - b$$
$$a^{\odot n} = \underbrace{a \odot \ldots \odot a}_{n\text{-times}}$$

We can also work with tropical matrices and tropical polynomials in the similarly intuitive way.

Example 1. Let us have two tropical matrices A and B of the same size 2×2:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$
$$B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$$

Computing the product of these two matrices is done as:

$$A \odot B = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \odot \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} =$$
$$= \begin{pmatrix} a_{1,1} \odot b_{1,1} \oplus a_{1,2} \odot b_{2,1} & a_{1,1} \odot b_{1,2} \oplus a_{1,2} \odot b_{2,2} \\ a_{2,1} \odot b_{1,1} \oplus a_{2,2} \odot b_{2,1} & a_{2,1} \odot b_{1,2} \oplus a_{2,2} \odot b_{2,2} \end{pmatrix}$$

Example 2. By the term *tropical polynomial*, we mean classical polynomial with tropical evaluating. Example of such polynomial p can be:

$$p(x) = x^{\odot 2} \oplus 3 \odot x \oplus (-2)$$

If we want to evaluate p in $x = 1$, we get:

$$p(1) = 1^{\odot 2} \oplus 3 \odot 1 \oplus (-2) = \min(1 \cdot 2, 3 + 1, -2) = -2$$

Remark. *In the whole thesis, we work with the tropical operators evaluating priority just like in the classical case. Thus $\odot$ has a higher priority than $\oplus$.*

1.3 Graphs and matrices

In Section 2 we will work with graphs of tropical matrices, so here is a good place to recall some terminology from the graph theory.

Definition 5 (Precedence graph)**.** *For the matrix $A = (a_{i,j})$, $a_{i,j} \in \mathbb{Z} \cup \{\varepsilon\}$ of the size $n \times n$ we define a directed weighted graph $\Gamma_A = (V, E)$ where $V = \{1, \ldots, n\}$ and $(i, j) \in E$ whenever $a_{i,j} \neq \varepsilon$. The weight of the edge (i, j) is a value of $a_{i,j}$. We call Γ_A the precedence graph of A.*

Definition 6 (Directed walk)**.** *In a directed graph $G = (V, E)$, we call the sequence $v_1, e_1, v_2, e_2, \ldots, v_{n-1}, e_{n-1}, v_n$, for $e_i \in E$, $v_i \in V$ a directed walk if it holds that every e_i is an edge from v_i to v_{i+1}.*

Definition 7 (Directed path)**.** *The walk is called a path whenever all vertices and edges in it are different up to the first and last vertex – the path can be closed.*

Definition 8 (Strictly connected graph)**.** *We say that the graph is strictly connected if there exists a path from i to j of finite weight for every $i, j \in V$.*

Definition 9 (Weight of a walk)**.** *The weight of a walk is a sum of weights of all edges in the given walk.*

Definition 10 (Cheapest k-step walk)**.** *The walk which contains k edges and leads from v_i to v_j in a weighted graph G is called the cheapest one (or the shortest one) if its weight is the minimum of weights of all walks from i to j that contain precisely k edges.*

Definition 11 (Critical cycle)**.** *A cycle in Γ_A is called critical if it has the smallest average weight per step of all the cycles in Γ_A. The average weight of the cycle is computed as a sum of weights of all edges in the cycle divided by the number of edges in the cycle.*

Definition 12 (Critical graph)**.** *For the matrix A we define the critical graph of A as the union of critical cycles from Γ_A.*

1.4 Cryptography

Because the original motivation for our work was to explore some cryptography schemes, we would like to recall cryptography basics for those who are not familiar with them.

The main goal of cryptography is to construct a secure communication between two parties via an insecure communication channel.

In classical cryptography, this is achieved by using some common key known only to these two parties between whom is the transported data encrypted. An example of this symmetric cryptography is an `AES` cipher. The main disadvantage of symmetric cryptography is the fact that both parties have to share the secret key.

A totally different part of cryptography is asymmetric (public-key) cryptography. The goal of asymmetric cryptography is to allow communication between parties which do not know each other and do not have the possibility to exchange a secret key via a secure channel (such as a personal meeting). Asymmetric cryptography is usually used for the initialization of the communication, namely for verifying both parties to each other and for the agreement on the key which will be used later for some symmetric cipher. That is because encrypting data is much faster by using some of the symmetric algorithms.

Public-key cryptography consists of numerous protocols and algorithms. For example the message encryption, key-agreement protocols, digital signatures, party authentication, etc. In our work, we will use some protocols of the first two types – for encrypting and decrypting messages and key-agreement protocols.

One of the keystones of the public-key cryptography are one-way functions. The one-way function is defined as a map $f : S \to S$ such that everyone can compute $f(x)$ relatively fast but it is hard to recalculate x from $f(x)$. An example of such a function can the be multiplication – we can compute a product of two primes, but the inversion (factorization) is hard.

1.4.1 Public-key encryption scheme

The idea of a public-key encryption scheme is that party `A` publishes its public key. Then everyone is able to encrypt the message in such a way, that only party `A` is able to decode it by using a private key which cannot be derived from the public one in a simple way.

Public-key encryption scheme consists of three algorithms: $\mathcal{G}, \mathcal{E}, \mathcal{D}$ and lives on some set $\mathcal{M}$ called the message space. We will denote the set of all possible keys as $\mathcal{K}$.

- *Key generation algorithm* $\mathcal{G}$ – the output of the algorithm is a pair of keys $(K_p, K_s) \in \mathcal{K}$. K_s is called the secret (private) key and is known only to the key owner. On the other hand, the public key K_p is known to everyone.

- *Encryption algorithm* $\mathcal{E}$ – the encryption algorithm outputs a ciphertext message for the plaintext input and public part of the key. The encryption is a function $\mathcal{E} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$.

- *Decryption algorithm* $\mathcal{D}$ – decrypts the ciphertext by using the private part of the key and an encrypted message. It is a function $\mathcal{D} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$.

We demand the correctness – decryption of encrypted message yields the original message:

$$\mathcal{D}(\mathcal{E}(m, K_p), K_s) = m$$

for every key $(K_p, K_s) \in \mathcal{K}$ and every message $m \in \mathcal{M}$.

One of the most known examples of the public encryption scheme is the `RSA` scheme.

1.4.2 Key-agreement protocol

The key-agreement protocol is used for establishing the common key for later usage in symmetric cryptography. It is used in the case when parties have no opportunity to exchange data via a secure channel. There are many different approaches on how to achieve this. One of the most used ones is Diffie-Hellman key exchange protocol.

Diffie-Hellman protocol is based on the discrete logarithm problem.

Definition 13 (Discrete logarithm problem)**.** *Let us have a cyclic group* $G = \langle g \rangle$*. For an element* $h \in G$ *compute such* $m \in \mathbb{N}$ *that* $h = g^m$*.*

One can easily compute g^n for a given $n \in \mathbb{N}$. But it is believed that the discrete logarithm problem is hard to compute, at least in a correctly chosen group, for example the multiplicative groups of $\mathbb{Z}_{/p}$ or cyclic subgroups of elliptic curves over finite fields.

Algorithm 2 (Diffie-Hellman key exchange protocol)**.** *The cyclic group* $G = \langle g \rangle$ *is publicly known.*

1. *Party* `A` *picks random* $a \in \mathbb{N}$*, party* `B` *picks random* $b \in \mathbb{N}$
2. `A` *computes* g^a *and sends it to* `B`*.*
3. `B` *sends* g^b *to* `A`*.*
4. *Both parties can now compute the same secret key* $(g^b)^a = (g^a)^b = g^{ab}$

At the end of the algorithm, both parties share the same key. This key can not be computed easily without the knowledge of a or b. Thanks to the discrete logarithm problem, the attacker is unable to compute these numbers (in a reasonable time) only from g, g^a, g^b.

1.5 Difference logic

In the next chapter, we will need to reduce a certain problem to the problem called "difference logic". Because it is not a commonly known problem we will introduce it here. We took inspiration from the second chapter of the paper Bjørner et al. [2008].

Definition 14 (Integer difference logic)**.** *The objective of the difference logic problem is to verify the existence of a solution of a system of inequalities of the form*

$$x_i - x_j \leq c_k \qquad k \in \{1, \ldots, m\} \qquad i, j \in \{1, \ldots, n\} \qquad i \neq j$$

where $x_1, \ldots, x_n \in \mathbb{Z}$ are variables and c_i are constant integers. If the solution exists, find such integer values of all x_i, y_i to satisfy all inequalities in the system, otherwise, output with failure.

This problem can be reduced to the problem of finding the shortest path between every pair of vertices of a directed graph. This is a commonly known problem which can be solved in polynomial time. The reduction is done as follows:

Claim 3 (Reduction of the difference logic problem)**.** *We will construct a directed weighted graph $G = (V, E)$ relevant to the given difference logic problem:*

- *$V = \{s, 1, \ldots, n\}$*
- *For each inequality $x_i - x_j \leq c_k$ in the difference logic system add an edge (i, j) of weight c_k into E.*
- *For each $i \in \{1, \ldots, n\}$ add (i, s) of the weight 0 into E.*

If G contains a cycle with negative weight, then there is no solution to the relevant difference logic problem. If G does not contain any negative cycle, then we can find a value of x_i as the minimal weight of the path from i to s.

Proof of this claim can be found in the article Bjørner et al. [2008] as Proposition 1.

Finding minimal weights for paths between every two vertices in a graph with n vertices is a polynomial problem solvable in an asymptotic time $\mathcal{O}(n^3)$ by Floyd-Warshall algorithm (Floyd [1962]).

2. Tropical variation of Stickel's key exchange protocol

Stickel's algorithm (Stickel [2005]) is one of many approaches to establish the key exchange. The original algorithm works in a nonabelian group but it is vulnerable to the linear algebra attack as was shown in (Shpilrain [2008]). In the same paper, the authors suggested the usage of semigroups with a lot of non-invertible elements. The concrete implementation of such protocol was presented in Grigoriev and Shpilrain [2014]. Their algorithm works with a semigroup of matrices over the tropical algebra. We will be interested in this proposal.

2.1 Algorithm description

Let us have two participants `A` and `B` who want to agree on a secret key via an unsecured channel for encrypted communication. Both sides have their public keys which are known to everybody. All the parameters used in the following algorithm are also publicly known.

Algorithm 4 (Tropical variation of Stickel's key exchange protocol)**.** *Let R be the semigroup of $n \times n$ matrices over the tropical algebra, and let $A, B \in R$ be public matrices such that $A \odot B \neq B \odot A$*

1. `A` *picks two random tropical polynomials $p_1(x)$, $p_2(x)$ of degrees taken uniformly randomly from $\{1, \ldots, d\}$ and sends $U = p_1(A) \odot p_2(B)$ to* `B`
2. `B` *picks two random tropical integer polynomials $q_1(x)$, $q_2(x)$ of degrees taken uniformly randomly from $\{1, \ldots, d\}$ and sends $V = q_1(A) \odot q_2(B)$ to* `A`
3. `A` *computes $K_A = p_1(A) \odot V \odot p_2(B)$*
4. `B` *computes $K_B = q_1(A) \odot U \odot q_2(B)$*

After Algorithm 4 successfully ends, both sides `A`, `B` share a computed common key $K = K_A = K_B$

Suppose that tropical operations ($\odot$ and $\oplus$) require a constant time to compute when working with integers. Let us compute the time complexity of the given algorithm.

Theorem 5 (The time complexity of Stickel's algorithm)**.** *The time complexity of Algorithm 4 is $\mathcal{O}(n^3 d)$*

Proof. First of all, we have to calculate the complexity of tropical matrix operations:

- Matrix addition: to add two matrices in R we have to calculate minimums of n^2 pairs of integers. It is in $\mathcal{O}(n^2)$.
- Matrix multiplication: to multiply two matrices in R we have to calculate a minimum of n results of integer addition for every matrix cell it is in $\mathcal{O}(n^3)$.

- Scalar multiplication of a matrix from R is in $\mathcal{O}(n^2)$.

In Algorithm 4 we have to evaluate tropical polynomials. The result of the polynomial evaluation is also a matrix in R. To compute it in the worst scenario, we have to compute all first d powers of the given matrix, multiply them by a scalar integer and sum it all together. The complexity of computing all d powers is in $\mathcal{O}(n^3 d)$. Indeed, powers can be precomputed and saved into a memory to avoid computing the powers for every monomial separately. So it remains for $i \in \{1, \ldots, d\}$ to take the i-th power of the matrix and multiply it by a scalar. And finally, sum it all together. It takes d scalar multiplications and d matrix additions. Altogether, the polynomial evaluation is in $\mathcal{O}(n^3 d)$.

The most expensive operation in Algorithm 4 is the polynomial evaluation which is carried out only constant times. The total complexity of Algorithm 4 is $\mathcal{O}(n^3 d)$. □

The goal of a passive attacker to Algorithm 4 is to compute the common key with knowledge of the public parameters of the protocol and messages sent during the key exchange – matrices A, B, U, V. It suffices to find matrices X, Y satisfying the following three equations:

$$\begin{aligned} X \odot A &= A \odot X \\ Y \odot B &= B \odot Y \\ X \odot Y &= U \end{aligned} \tag{2.1}$$

Now, we can easily compute K as:

$$\begin{aligned} X \odot V \odot Y = X \odot q_1(A) \odot q_2(B) \odot Y = q_1(A) \odot X \odot Y \odot q_2(B) = \\ = q_1(A) \odot U \odot q_2(B) = K_B = K \end{aligned}$$

Finding matrices X, Y as a solution of a system of linear equations would be easy if we worked in a ring of matrices over a field. But since we compute in a tropical algebra where addition is not invertible, this approach should not work.

2.2 Simple heuristic attack

When computing powers for matrices in R, values of matrix entries are likely to drop, especially if many of entries are negative. This behavior follows from Theorem 14 which is described in the chapter about powers of tropical matrices. A matrix with many negative entries tends to have also a negative value of the parameter c from Theorem 14.

Such behavior leads to the fact that a value of a polynomial of high enough degree is given only by a monomial of the highest degree. This is because every lower monomial usually has each matrix entry value higher than the leading monomial. Thus for the polynomial p of degree d with a leading coefficient c_d and the matrix A we get with high probability:

$$p(A) = c_d \odot A^{\odot d}$$

This can be used in the following attack designed by Kotov and Ushakov (see Kotov and Ushakov [2018]). In the attack, we use the fact that the tropical multiplication of scalars is easily invertible – it is the same as the usual subtraction.

Algorithm 6 (Simple heuristic attack on Stickel's protocol)**.** *The input of the algorithm is an integer d (maximal allowed degree of polynomials) and matrices A, B, U.*

1. *For every $i \in \{0, \ldots, d\}$ and for every $j \in \{0, \ldots, d\}$ compute the matrix $T^{i,j}$ as*
$$T^{i,j} = U - A^{\odot i} \odot B^{\odot j}$$
2. *If $T^{i,j}$ is constant for some $c \in \mathbb{Z}$, $T^{i,j} = c \odot (\mathbb{1})_{n \times n}$, then set*
$$X = c \odot A^{\odot i}$$
$$Y = B^{\odot j}$$
and output with SUCCESS.
3. *Else return FAILED.*

Algorithm implementation in Python:

```
def simpleAttack(A, B, U, d):
    Ai = A.getPowers(d)              #Ai = [A^0, A^1, ... , A^d]
    Bj = B.getPowers(d)              #Bj = [B^0, B^1, ... , B^d]
    for i in range(0,d+1):
        for j in range(0,d+1):
            T = U - (Ai[i] * Bj[j])
            if(T.isConstant()):
                X = T.M[0][0] * Ai[i]            #T.M[x,y] = c
                Y = Bj[j]
                if(checkSolution(A,B,U,X,Y)):
                    return True
    return False
```

This attack relies on the fact that with high enough probability the following inequalities hold (in every matrix entry):

$$\begin{aligned} c_{d_1} \odot A^{\odot d_1} &\leq c_i \odot A^{\odot i} \qquad i \in \{0, \ldots, d_1 - 1\} \\ c_{d_2} \odot B^{\odot d_2} &\leq c_i \odot B^{\odot i} \qquad i \in \{0, \ldots, d_2 - 1\} \end{aligned} \tag{2.2}$$

where d_1 is the degree of p_1 and d_2 is the degree of p_2.

Theorem 7 (Conditions of the success of Algorithm 6)**.** *Algorithm 6 yields the solution (outputs with SUCCESS) if the inequalities 2.2 hold for both matrices A and B.*

Proof. We will denote the degree of p_1 as d_1 and the degree of p_2 as d_2. When both inequalities 2.2 hold then the matrix U is computed as:

$$U = p_1(A) \odot p_2(B) = c_{d_1} \odot A^{\odot d_1} \odot c_{d_2} \odot B^{\odot d_2} = k \odot A^{\odot d_1} \odot B^{\odot d_2}$$

for $k = c_{d_1} + c_{d_2}$.

Because $d_1, d_1 \in \{0, \ldots, d\}$ we must definitely find such i, j in the first step of Algorithm 6 that the matrix T is:

$$T = U - A^{\odot i} \odot B^{\odot j} = c \odot (\mathbb{1})_{n \times n}$$

for some $c \in \mathbb{Z}$. Therefore T has to be a constant matrix.

Now we need to verify that the conditions 2.1 are met for the pair (i, j).

1. $X \odot A = A \odot X$. This condition holds because of $X = c \odot A^{\odot i}$ and therefore $X \odot A = c \odot A^{\odot i} \odot A = c \odot A^{\odot i+1} = c \odot A \odot A^{\odot i} = A \odot c \odot A^{\odot i} = A \odot X$.

2. $Y \odot B = B \odot Y$. This condition is clear because Y is a power of B.

3. $X \odot Y = U$. This is true from the calculation of U and due to the fact that $i = d_1$ and $j = d_2$.

□

Theorem 8 (The time complexity of Simple heuristic attack)**.** *Time complexity of Algorithm 6 is in* $\mathcal{O}(n^3 d^2)$.

Proof. The algorithm contains two nested loops, each one runs $d + 1$ times in the worst case. The main work is done in the inner loop – matrix multiplication, which has a complexity $\mathcal{O}(n^3)$. While this is done in two nested loops, the complexity of loops is in $\mathcal{O}(n^3 d^2)$. We avoided computing the i-th matrix power every time in the inner loop by precomputation all needed powers before loops, which corresponds to d times matrix multiplication, which is in $\mathcal{O}(n^3 d)$. But this complexity is lower than the total complexity of loops.

Altogether, the total complexity of Algorithm 6 is in $\mathcal{O}(n^3 d^2)$. □

2.2.1 Experiment

Although authors of the heuristic attack come with results of measured attack time and attack accuracy, we would like to explore how the attack is affected by a change of parameters n, d and size of the coefficients of polynomials. In Kotov and Ushakov [2018] there are considered only $n = 10$, $d = 10$ and coefficient sizes equal to 10^3 and 10^{10}.

We have run the attack implemented in Python on the desktop computer with Intel iCore 3.4 GHz processor and 16.0 GB RAM. We have run the algorithm for various sizes of n between 5 and 30 and for d between 5 and 100. We have chosen the maximal sizes of coefficients of polynomials as 10^{10}.

As expected from asymptotic estimation, the running time of Algorithm 4 is not very different from the running time of Algorithm 6 and with higher n it increases very fast. For the maximum size of parameters, the successful attack takes about 4 times longer than the key agreement. This makes this variant of the key-exchange protocol unsuitable for practical use.

A little bit more interesting is the success rate of Algorithm 6. It can be seen from results (Figure 2.1), that the success rate of the attack is very quickly growing towards 100% with increasing n and d. Because the attack relies on the occurrence of inequalities 2.2, it fails thanks to:

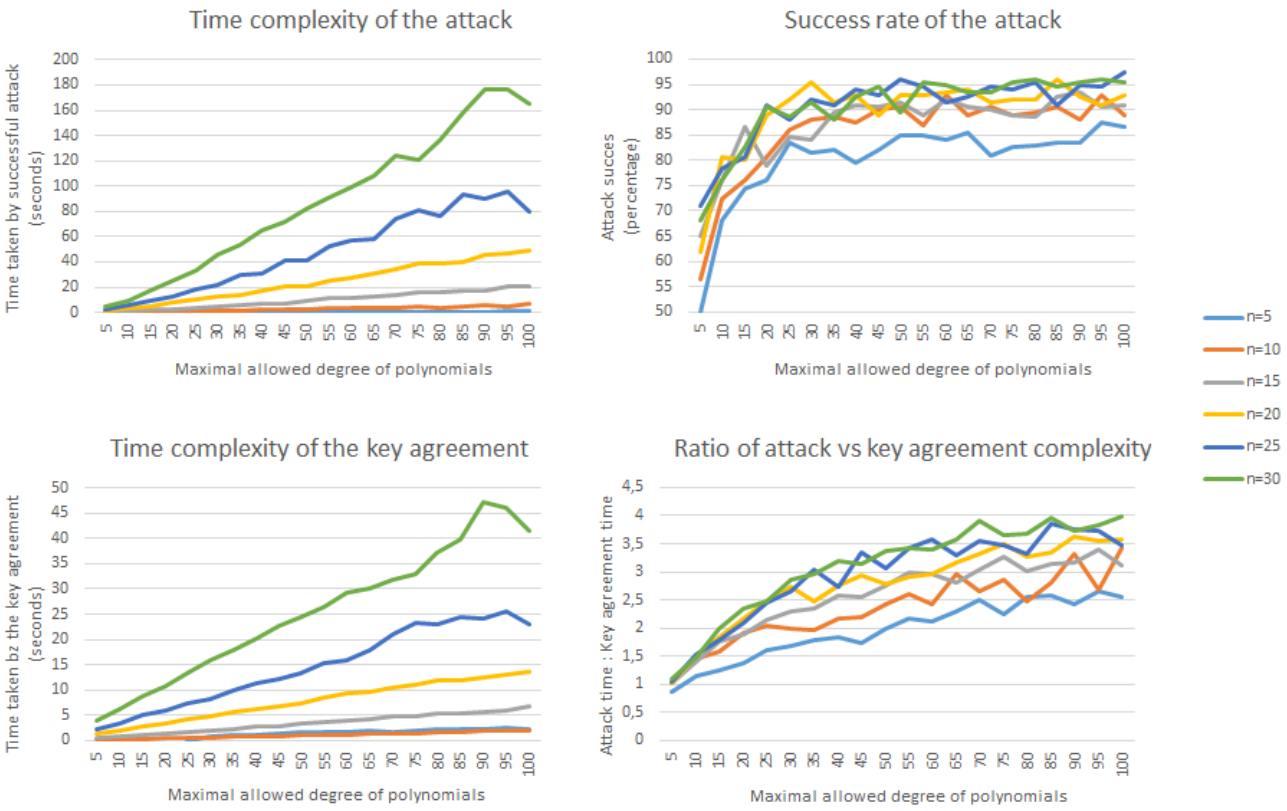

Figure 2.1: Results of Algorithm 6

- Small d: with small d there is not enough time for the matrix to make every entry negative. Further, coefficients of polynomials also play an important role. For example, if c_{d-1} is negative and c_d is positive then it is possible that for some i, j occurs:

$$c_d \odot \left(A^d\right)_{i,j} > c_{d-1} \odot \left(A^{d-1}\right)_{i,j}$$

- Small n: with small n there is a bigger probability that most of the matrix entries will be positive. In that case, it is not guaranteed that the values of entries will decrease with the increasing power of the matrix.

2.3 Variant of the protocol with non-negative numbers

What happens if we slightly change the Stickel's algorithm to work only with non-negative numbers? In the algorithm, we will generate only non-negative matrix entries and polynomials coefficients. Rest of the algorithm stays the same. With this change, we will not change the functionality nor the complexity of the algorithm.

But now, while computing powers of the matrix, there are no negative numbers and so the behavior of the matrix is less predictable. This has a big impact on Algorithm 6 because there is only a small probability that there exists one monomial which gives us the total value of the polynomial evaluation. When we

run the simple attack on the non-negative variant of Stickel's protocol we get a very low success rate of the algorithm.

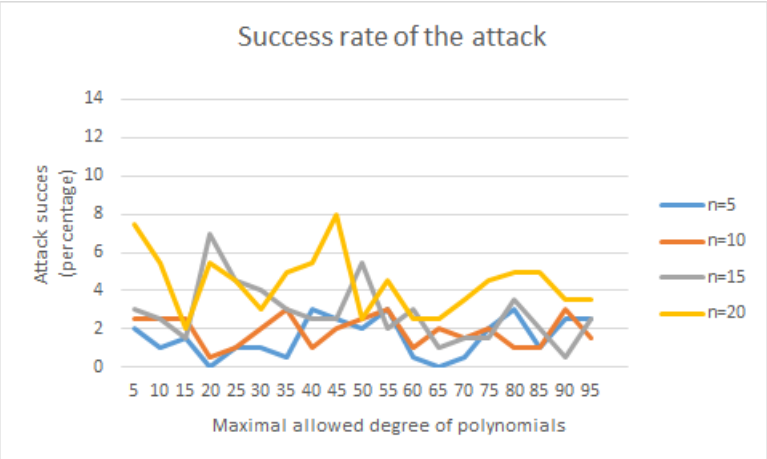

Figure 2.2: Results of Algorithm 6 for non-negative numbers

There is an attack suggested in Kotov and Ushakov [2018] which deals also with non-negative protocol and which has a 100% success rate. Their algorithm is looking for x_i, y_j, $i, j \in \{0, \ldots, d\}$ which satisfy the following equality:

$$\bigoplus_{i,j}^{d} x_i \odot y_j \odot A^{\odot i} \odot B^{\odot j} = U$$

To achieve this, we have to solve the following system of equations. For the matrices $T^{i,j} = A^{\odot i} \odot B^{\odot j} - U$ and every $k, l \in \{1, \ldots, n\}$ it has to hold:

$$\min_{i,j} \left\{ x_i + y_j + T^{i,j}_{k,l} \right\} = 0$$

We will denote minimum of the entries in the matrix $T^{i,j}$ as $m_{i,j}$:

$$m_{i,j} = \min_{k,l} T^{i,j}_{k,l}$$

Further, we compute $(d+1)^2$ sets of minimum entries of $T^{i,j}$:

$$P_{i,j} = \left\{ (k,l); T^{i,j}_{k,l} = m_{i,j} \right\}$$

Now, from these sets, we generate covers of the set $\{1, \ldots, n\} \times \{1, \ldots, n\}$ and for every cover C we get the following system of equalities and inequalities:

$$\begin{aligned} x_i + y_j &= -m_{i,j}, \ P_{i,j} \in C \\ x_i + y_j &\geq -m_{i,j}, \ P_{i,j} \notin C \end{aligned} \tag{2.3}$$

For a given C, system 2.3 does not always have a solution. A smaller amount of equalities gives us a bigger probability that this system is consistent and can be solved by linear algebra tools. The optimum is a minimal cover. But finding the minimal cover is a NP-complete problem (Chew).

In the article written by Kotov and Ushakov [2018], they solve the system 2.3 by finding all covers of $\{1,\dots,n\}\times\{1,\dots,n\}$, ordering them by the number of sets used in the cover and other criteria. After that, they are trying the covers one after another until one of them yields the solution of 2.3. This approach – to generate all covers – works fine for parameter $d = 10$ because the size of most of the $P_{i,j}$ is small and so there are about 700 covers on average (according to their measurement). But because there may be exponentially many covers (in d) the attack time will probably grow exponentially with d. When we run their implementation of the attack, we waited only one second for the solution when d was 10, nearly 20 seconds for the solution for $d = 20$ and more than 200 seconds for $d = 30$. The size of the matrix was always the same: $n = 10$.

2.4 General attack

It seems that Stickel's protocol could be resistant against attacks proposed in Kotov and Ushakov [2018] as long as entries of matrices A and B are non-negative. But if we do not require 100% success rate of the attack we can construct Algorithm 9 which is only a slight modification of the attack proposed by Kotov and Ushakov [2018]. The complexity of our attack is not exponential with respect to the running time of tropical Stickel's protocol.

In our attack, we will not be searching through all covers. We will just pick one, which should be close to the minimum one, and test if it yields a solution. With this approach, we get a relatively high success rate and whenever we get the solution, we get it relatively fast.

Algorithm 9 (Attack on Stickel's protocol with non-negative numbers)**.** *Input of the algorithm is the maximal allowed degree of polynomials d and matrices A, B, U.*

1. *For every $i \in \{0,\dots,d\}$, for every $j \in \{0,\dots,d\}$ compute matrix $T^{i,j}$ as*

$$T^{i,j} = A^{\odot i} \odot B^{\odot j} - U$$

2. *For all $T^{i,j}$ compute the set $P_{i,j}$ as:*

$$P_{i,j} = \left\{(k,l); T^{i,j}_{k,l} = \min_{k,l} T^{i,j}_{k,l}\right\}$$

3. *Find such pairs (i,j) that $P_{i,j}$ has to be in any cover of the set $\{1,\dots,n\}\times\{1,\dots,n\}$. By this we mean that if for $(k,l)\in\{1,\dots,n\}\times\{1,\dots,n\}$ there exists just one $P_{i,j}$ such that $(k,l)\in P_{i,j}$, then add this $P_{i,j}$ into the cover C.*

4. *Iterate the following until C is a cover of $\{1,\dots,n\}\times\{1,\dots,n\}$:*

   (a) *Pick (a,b) from $\{1,\dots,n\}\times\{1,\dots,n\}$ such that $(a,b)\notin\{\bigcup P; P\in C\}$*

   (b) *Find the set $C_{a,b} = \{P_{i,j}; (a,b)\in P_{i,j}\}$*

(c) *Find such $P_{i,j} \in C_{a,b}$ that $P_{i,j}$ has the maximal size among sets in $C_{a,b}$ and append this $P_{i,j}$ into C*

5. *Try to find a solution of the system 2.3:*

$$x_i + y_j = -m_{i,j},\ P_{i,j} \in C$$
$$x_i + y_j \geq -m_{i,j},\ P_{i,j} \notin C$$

6. *Return SUCCESS iff the solution exists*

In our implementation, we left the solution of the 5-th step of the algorithm for the SAT solver[1].

As we will show in Theorem 10, the time needed by the attack depends mainly on computing all matrices $T^{i,j}$ the rest of the algorithm is really fast. In experiment measurements, we checked that the success rate of the attack is around 70% (see figure 2.3) which is high enough. The success rate could be further improved by devoting more efforts to search for the minimal cover or trying more covers candidates for the one which yields the solution.

Theorem 10 (Time complexity of the attack on Stickel's protocol)**.** *The time complexity of Algorithm 9 is $\mathcal{O}(n^3d^3)$*

Proof. The first step of the algorithm is to compute all $T^{i,j}$ for all $i \in [0, d]$, for all $j \in [0, d]$. For this it is needed:

- Precompute all the powers of matrices A and B. For every matrix it is needed to perform d tropical multiplications, altogether the complexity of precomputing the powers is $\mathcal{O}(n^3d)$.
- For computing one of the $T^{i,j}$ we need to perform one tropical matrix multiplication and one classical matrix subtraction. So for one $T^{i,j}$ we have complexity $\mathcal{O}(n^3)$, for every $T^{i,j}$ we obtain the complexity $\mathcal{O}(n^3d^2)$.

From this, we have the complexity of the first step: $\mathcal{O}(n^3d^2)$.

The second step of the algorithm is easy – we just have to find minimum values in every matrix $T^{i,j}$. For that, it is needed to look at every value of the matrix. And for finding all the pairs (k, l) where the minimum is achieved we have to look at every value once more. Altogether, the time complexity of step 2 is $\mathcal{O}(n^2d^2)$ because we have to go through d^2 matrices with n^2 entries.

Because we need to check if a concrete pair (k, l) is in $P_{i,j}$ we can implement $P_{i,j}$ as a field of size $n \times n$. If $(k, l) \in P_{i,j}$ then $P_{i,j}[k, l] = 1$, else we set the value of $P_{i,j}[k, l] = 0$.

We have implemented step 3 as follows:

- Fill array a of the size $n \times n$ with zeros.
- For every $P_{i,j}$ and for every $(k, l) \in P_{i,j}$ increment $a[k, l]$ by 1.
- If $a[k, l] = 1$ then find the only one $P_{i,j}$ that $(k, l) \in P_{i,j}$.

[1] Z3 Solver library for python: https://pypi.org/project/z3-solver/

So we need to iterate over every member of every $P_{i,j}$ which includes $n^2(d+1)^2$ members. And then, in the last part of this sub-algorithm, we iterate through every $P_{i,j}$ and we are looking for the one where $P_{i,j}[k,l] = 1$. This can be done also in $\mathcal{O}(n^2d^2)$.

The maximum number of iterations of substeps of step 4 is limited by n^2. The maximal size of the cover C can be n^2. We note which tiles of the field $\{1,\ldots,n\} \times \{1,\ldots,n\}$ are covered into some field c of size $n \times n$. In the substep (b) we iterate through $(d+1)^2$ sets $P_{i,j}$ and check whether the value of $P_{i,j}[a,b]$ is equal to 1. In the substep (c), we list through the sets $P_{i,j}$ of $C_{a,b}$, there are no more than $(d+1)^2$ of them. We need to find out the number of 1 in $P_{i,j}$. This could be done during the calculation of $P_{i,j}$ and saved into some auxiliary field of size $(d+1) \times (d+1)$ without affecting the complexity. The total complexity of step 4 is in $\mathcal{O}(n^2d^2)$.

The 5-th step of the algorithm is tricky. We left the solving of the system of linear inequations and equations on the SAT solver because the number of equalities and inequalities is low enough for solving by it. But we can compute the complexity of this step.

We have a system of linear integer equalities and inequalities. There are all together $(d+1)^2$ of them with a total number of $2(d+1)$ variables. We will reduce this into integer difference logic problem (see Definition 14) similarly as in Miné [2006]. We will define $2(d+1)$ variables: $\tilde{x}_0,\ldots,\tilde{x}_d,\tilde{y}_0,\ldots,\tilde{y}_d$. The variable $\tilde{x}_i$ corresponds to the negative form of x_i. The variable $\tilde{y}_i$ corresponds to the variable y_j. From every equality of the shape $x_i + y_j = -m_{i,j}$ we can create two inequalities: $x_i + y_j \leq -m_{i,j}$ and $x_i + y_j \geq -m_{i,j}$. Then we will process through all inequalities and transform them into the form suitable for the difference logic:

- $x_i + y_j \geq -m_{i,j} \quad \rightarrow \quad \tilde{x}_i - \tilde{y}_j \leq m_{i,j}$
- $x_i + y_j \leq -m_{i,j} \quad \rightarrow \quad \tilde{y}_j - \tilde{x}_i \leq -m_{i,j}$

Now we have built a set of inequalities which meets entry conditions of a difference logic problem. That problem is solvable in polynomial time depending on the number of variables. We have $2d$ variables, so using, for example, Floyd–Warshall algorithm we will get the complexity of this step in $\mathcal{O}(d^3)$.

The total complexity of Algorithm 9 is $\mathcal{O}(n^3d^3)$. □

From these results, we came to the conclusion that the suggested tropical scheme for key agreement is not suitable for real usage. We have an attack which works with high success rate and its complexity is only polynomially worse than the key agreement protocol (Algorithm 4).

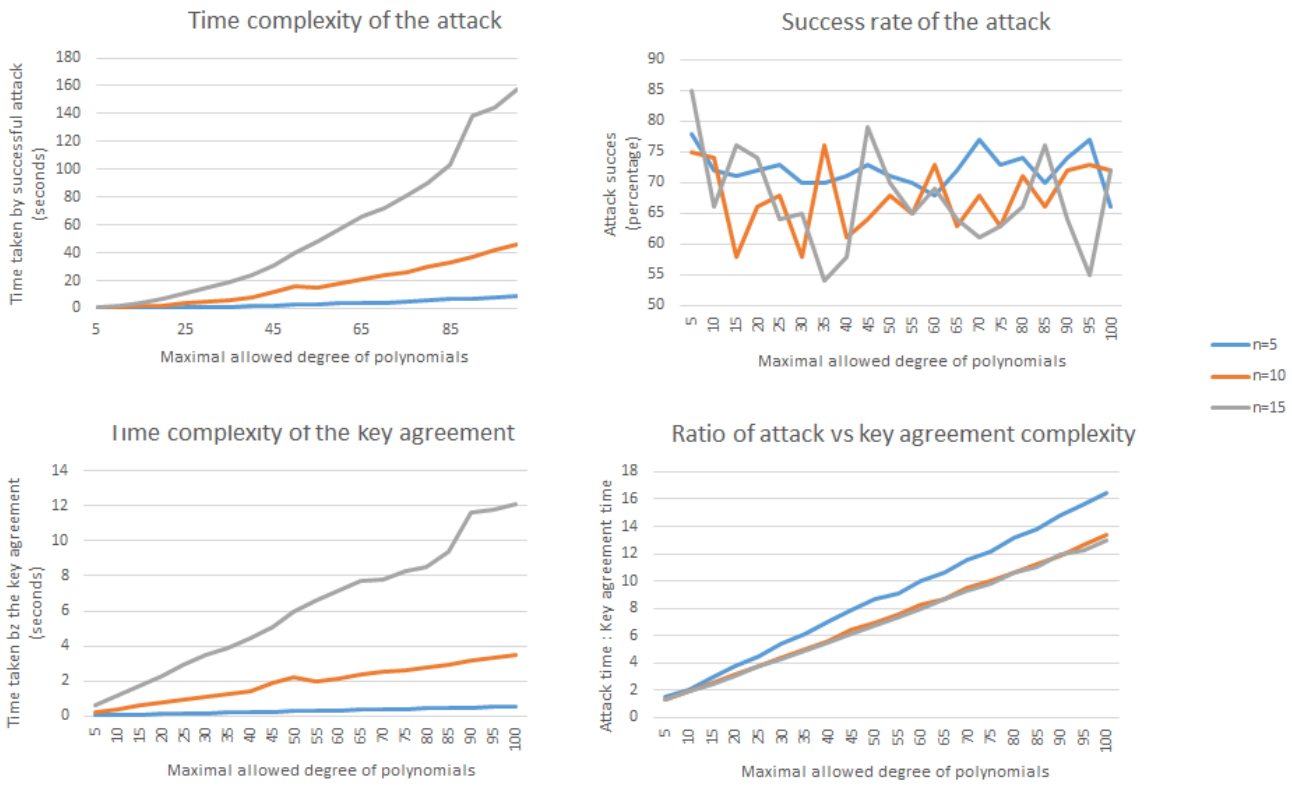

Figure 2.3: Results of Algorithm 9

2.5 Other variants of the tropical key exchange protocol

At this point, we would like to briefly explore another variant of the Stickel's protocol. The main disadvantage of the original tropical protocol is the need for the evaluation of matrices by polynomials. When it would be enough to compute only one matrix power (for each matrix) we would get logarithmic time for the key generation because there exists a fast algorithm for computing powers – binary exponentiation. In the attacks that we were discussing, it is needed to precompute all the matrix powers which had to be done one after another. So we would get a fast algorithm against slow attacks and one could think about the practical usage of the algorithm as an implementation of a secure key-agreement protocol.

Algorithm 11 (Fast Stickel's key exchange protocol)**.** *Let R be the semiring of $n \times n$ matrices over the tropical integers, and let $A, B \in R$ be public matrices such that $A \odot B \neq B \odot A$.*

1. `A` *picks two random positive integers a_1, a_2 and sends $U = A^{\odot a_1} \odot B^{\odot a_2}$ to* `B`.
2. `B` *picks two random positive integers b_1, b_2 and sends $V = A^{\odot b_1} \odot B^{\odot b_2}$ to* `A`.
3. `A` *computes $K_A = A^{\odot a_1} \odot V \odot B^{\odot a_2}$.*
4. `B` *computes $K_B = A^{\odot b_1} \odot U \odot B^{\odot b_2}$.*

In the whole algorithm, we generate random integers (a_1, a_2, b_1, b_2) of the maximal value d.

Since the algorithm works mainly as the original tropical Stickel's algorithm, it is not hard to compute its complexity.

Claim 12 (Complexity of the Fast Stickel's algorithm)**.** *The fast tropical Stickel's protocol has time complexity in $\mathcal{O}(n^3 \log(d))$.*

Proof. Using the binary exponentiation we recompute the complexity of computing U and V. Because there is needed to carry on only $\mathcal{O}(\log(d))$ matrix multiplications when calculating $A^{\odot d}$, the time complexity of computing $A^{\odot d}$ is $\mathcal{O}(n^3 \log(d))$. Still, we need to compute only a constant number of matrix powers and do some matrix multiplications in the rest of the algorithm, the total complexity of Algorithm 11 is $\mathcal{O}(n^3 \log(d))$. □

Both attacks which we presented are based on computing all matrix powers from 0 to some unknown d. The essence of the simple attack (Algorithm 6) was to find out the exact maximum degrees of the polynomials generated during the Stickel's algorithm. Because now the calculation of U (and V respectively) is only a product of two matrices and it is not complicated by a polynomial evaluation, we would get a deterministic attack with 100% success rate when using Algorithm 6. But its complexity would stay in $\mathcal{O}(n^3 d^2)$. So the complexity of the attack is exponential against the complexity of the key exchange protocol (Algorithm 4) in d. And it makes sense to fix the matrix size n and to think of the attack complexity only in the parameter d.

We tried to run the attack (Algorithm 6) on Protocol 11 with a low absolute value of matrix entries. We found out that, despite the fact that the chosen exponents a_i, b_i were really large, the attack came out with a positive result very fast. When we let it list the found exponents, the results were really surprising. The attack found the matrix T constant for really low powers (only about 5) of matrices A and B. That inspired us for the further exploration of the matrix powers behavior in the next chapter.

3. Tropical matrix powers

In this chapter, we will focus on computing matrix powers in tropical algebra. In the whole chapter, we will stay focused only on square matrices with integer entries and our tropical algebra stays over the set $\mathbb{Z} \cup \{\infty\}$. Because of the zero element of the tropical algebra, matrices may contain the tropical zero (∞) value.

Let us recall the matrix multiplication in a tropical semiring.

$$\begin{aligned} A \odot B &= \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} \odot \begin{pmatrix} b_{1,1} & \dots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \dots & b_{n,n} \end{pmatrix} \\ &= \Big(a_{i,1} \odot b_{1,j} \oplus a_{i,2} \odot b_{2,j} \oplus \dots \oplus a_{i,n} \odot b_{n,j} \Big)_{i,j=1,\dots,n} \\ &= \Big(\min(a_{i,1} + b_{1,j}, a_{i,2} + b_{2,j}, \dots, a_{i,n} + b_{n,j}) \Big)_{i,j=1,\dots,n} \\ &= \Big(\min_k (a_{i,k} + b_{k,j}) \Big)_{i,j=1,\dots,n} \end{aligned}$$

3.1 Congruence of tropical matrices

We will start with an easy example.

Example 3. Let us look at what happens during the computation of powers of the matrix of size 2×2 when the values of its entries are ordered: $a_{1,1} \leq a_{1,2} \leq a_{2,1} \leq a_{2,2}$

$$\begin{aligned} A &= \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \\ A^{\odot 2} &= \begin{pmatrix} \min(2a_{1,1}, a_{1,2} + a_{2,1}) & \min(a_{1,1} + a_{1,2}, a_{1,2} + a_{2,2}) \\ \min(a_{2,1} + a_{1,1}, a_{2,2} + a_{2,1}) & \min(a_{2,1} + a_{1,2}, 2a_{2,2}) \end{pmatrix} \\ &= \begin{pmatrix} 2a_{1,1} & a_{1,1} + a_{1,2} \\ a_{1,1} + a_{2,1} & a_{1,2} + a_{2,1} \end{pmatrix} \\ A^{\odot 3} &= \begin{pmatrix} \min(3a_{1,1}, a_{1,1} + a_{1,2} + a_{2,1}) & \min(2a_{1,1} + a_{1,2}, a_{1,1} + a_{1,2} + a_{2,2}) \\ \min(2a_{1,1} + a_{2,1}, a_{1,2} + 2a_{2,1}) & \min(a_{1,1} + a_{1,2} + a_{2,1}, a_{1,2} + a_{2,1} + a_{2,2}) \end{pmatrix} \\ &= \begin{pmatrix} 3a_{1,1} & 2a_{1,1} + a_{1,2} \\ 2a_{1,1} + a_{2,1} & a_{1,1} + a_{1,2} + a_{2,1} \end{pmatrix} \end{aligned}$$

And if we continued with higher powers, we would have noticed that difference of consecutive powers is always the same. That means that for every $i \geq 3$ holds:

$$A^{\odot i} - A^{\odot (i-1)} = \begin{pmatrix} a_{1,1} & a_{1,1} \\ a_{1,1} & a_{1,1} \end{pmatrix}$$

In other words, we can express the i-th power of the matrix A as a tropical scalar multiple of the $(i-1)$-th power of the matrix A:

$$A^{\odot i} = a_{1,1} \odot A^{\odot (i-1)}$$

Let us define some terms for further usage:

Definition 15 (Constant matrix)**.** *We will call the matrix constant whenever all of its entries have the same value c for some $c \in \mathbb{Z}$. We will denote the fact that the matrix A of size $n \times n$ is constant (for given c) as:*

$$A = c \odot (\mathbb{1})_{n \times n}$$

The surprising fact from the last section of the previous chapter leads us to the definition of constantly different matrices. We could call $A, B \in \mathbb{Z}^{n \times n}$ constantly different iff it holds that $A - B = c \odot (\mathbb{1})_{n \times n}$ for some $c \in \mathbb{Z}$. This definition is good enough when we do not work with the tropical zero element ε. For that we need a better definition of constantly different matrices:

Definition 16 (Constantly different matrices)**.** *If for $A, B \in (\mathbb{Z} \cup \{\infty\})^{n \times n}$ and some $c \in \mathbb{Z}$ holds that $A = c \odot B$, then we call these two matrices constantly different.*

Nonnegative powers of a matrix define a subsemigroup of $((\mathbb{Z} \cup \{\infty\})^{n \times n}, \odot)$. As we have seen in the previous Example 3, it can happen that the semigroup will contain constantly different matrices. We can define congruence on this semigroup:

Definition 17 (Tropical matrix congruence)**.** *On the set of integer square matrices $(\mathbb{Z} \cup \{\infty\})^{n \times n}$ we define the congruence $\equiv$. We say $A \equiv B$ when $A = c \odot B$ for some $c \in \mathbb{Z}$ (A and B are constantly different).*

It is easy to verify that this relation is congruence.

Now we would like to look at the semigroup generated by some matrix under this congruence. If we take the semigroup generated by matrix A from Example 3 then the quotient semigroup $(\langle A \rangle_{\equiv}, \odot_{\equiv})$ is finite and has only 3 elements $(A^{\odot 0}, A^{\odot 1}, A^{\odot 2})$.

Not surprisingly, not every matrix behaves like the matrix given in Example 3.

Example 4. The matrix which has different entries on diagonal and tropical zeros off-diagonal generates an infinite sub-semigroup of $((\mathbb{Z} \cup \{\infty\})^{n \times n}, \odot)$ under the congruence $\equiv$.

Example 5. Let us have a matrix B of size 2×2 with entries:

$$B = \begin{pmatrix} 2 & 7 \\ 7 & 1 \end{pmatrix}$$

Then when we are computing powers of B we can discover that for every $i \geq 13$ it holds that $B^{\odot i} - B^{\odot (i-1)}$ is a constant matrix (for smaller i this does not hold). This means that:

$$|(\langle B \rangle_{\equiv}, \odot_{\equiv})| = 13$$

Example 6. Let us have a matrix C of size 2×2 with entries:

$$C = \begin{pmatrix} 3 & 0 \\ 1 & 1 \end{pmatrix}$$

If we compute powers of C we will not find any $i \in \mathbb{N}$ for which $C^{\odot i}$ and $C^{\odot (i-1)}$ are constantly different. But on the other hand, if we check the result of $C^{\odot i} - C^{\odot (i-2)}$ for $i > 4$ then we will find out that it is a constant matrix.

From the previous examples, we obtain an intuitive question. What are all the possible structures of quotient semigroups (under the congruence $\equiv$) generated by tropical matrices? What are the conditions for the semigroup to be finite?

3.2 Structure of quotient semigroups generated by tropical matrices

In the previous examples, we have presented two different types of quotient semigroups $(\langle A\rangle_{\equiv}, \odot_{\equiv})$. The first is an infinite semigroup, the second is a semigroup composed of a tail and a cycle. Of course, a semigroup generated by the identity matrix $(\mathbb{I}_{n\times n}, \odot)$ has only one element, but when considered as a quotient semigroup with the congruence $\equiv$ it corresponds with the second type having a tail of the length 0 and a cycle of the length 1.

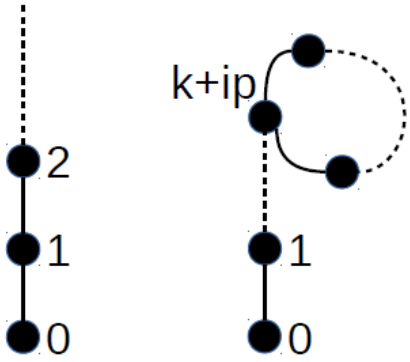

Figure 3.1: Two types of quotient semigroups of integer tropical matrices

For a tropical square matrix, we can define cyclicity and transience which characterize a structure of a finite semigroup generated by this matrix under the congruence $\equiv$ as we will see in Definition 19.

3.2.1 Cyclicity theorem

We would like to characterize the criterion for the type of the quotient semigroup created by the given tropical matrix. This topic is covered in the article *Supertropical matrix algebra III: Powers of matrices and their supertropical eigenvalues* (Izhakian and Rowen [2011]). We will recapitulate the main ideas from it and we will expand them more in the following subsections.

Let us recall that we can represent a matrix by its precedence graph (see Definition 5). This will be useful for an illustration of how the matrix powers are computed. Thanks to the definition of tropical multiplication, we can execute the computation of matrix powers on its precedence graph. We will start with matrix multiplication:

Claim 13. *Let us have three matrices $A, B, C \in (\mathbb{Z}\cup\{\infty\})^{n\times n}$ such that $A\odot B = C$. Let Γ_A, Γ_B be precedence graphs of corresponding matrices. Then $c_{i,j}$ equals to the weight of the cheapest path of the length 2 from i to j, when we take the first step in a graph Γ_A (from i to some vertex x) and the second step in Γ_B (from x to j). When no such path exists, then $c_{i,j} = \infty$.*

This implies that when computing powers of a matrix A, the value of the entry $a^m_{i,j}$ in A^m corresponds to the weight of the cheapest walk of the length m from the vertex i to the vertex j in Γ_A.

Definition 18 (Irreducible matrix)**.** *We call matrix A irreducible whenever Γ_A is strictly connected.*

The behavior of matrix powers for an irreducible matrix is described in Cyclicity theorem, which is referred to as a given fact in many papers on the tropical algebra. But to locate the exact citation with proof was not easy, and finally, we managed to find it in Gaubert [1994] as Theorem 1.2.3. We present it here in the context of our work as follows:

Theorem 14 (Cyclicity theorem)**.** *If the matrix $A \in (\mathbb{Z} \cup \{\infty\})^{n\times n}$ is irreducible, then there exist $c \in \mathbb{Z}$ and $p \in \mathbb{N}$ such that for $k \in \mathbb{N}$ high enough it holds that:*

$$A^{\odot(k+p)} = c \odot A^{\odot k}$$

Definition 19 (Index of cyclicity and transience of a matrix)**.** *The smallest possible $p \in \mathbb{N}$ from Theorem 14 is called cyclicity (or the index of cyclicity) of A and we will denote it as p_A. The smallest possible $k \in \mathbb{N}$ is called transience (or the index of transience) of A and will be denoted as t_A.*

We will see later in this chapter that the value of p_A can be computed from the precedence graph of A (see Lemma 16). Further, we will see in Proposition 18, that $c = \lambda(A)^{\odot p_A}$ where $\lambda(A)$ is an average cost of one step in a critical cycle of Γ_A. There exists an upper bound for the value of t_A, we present it in Claim 17.

This implies that every tropical matrix from $\mathbb{Z}^{n\times n}$ creates a finite semigroup under the congruence $\equiv$. What about matrices which contain a tropical zero element? While the precedence graph stays strictly connected, then such a matrix corresponds to a finite semigroup, again by Theorem 14.

3.2.2 Quotient semigroups with a cycle

For this moment, we will focus only on matrices with strictly connected precedence graphs. We want to estimate the length of a cycle and a tail of the semigroup generated by such a matrix with respect to $\equiv$.

The length of the cycle

Let A be a matrix of size $n \times n$ and, for simplicity, let A be from $\mathbb{Z}^{n\times n}$. From Cyclicity theorem, we know that this matrix is cyclic and p denotes cyclicity of A. What does cyclicity mean for Γ_A?

Critical cycles are important for the cyclicity of matrices. We can interpret entries in the k-th power of a given matrix as the cost of the cheapest k-step walk in its precedence graph. Thanks to that, the walk has to contain some cycle when the power grows because the number of edges in the graph is limited by the size of the matrix. It can be shown that, for high enough power, the cheapest k-step walk will contain one of the critical cycles because they have the smallest average cost per step which outweighs the cost needed for getting to and from the cycle

(for a detailed explanation see Chapter 4 – Critical Bound of Charron-Bost et al. [2017]).

For this moment, let us suppose that Γ_A contains only one critical cycle. In that case the length of the critical cycle equals to cyclicity of the matrix as follows from Lemma 16. Let l_c be the length (in the meaning of the count of edges) of that critical cycle. When we have the cheapest k-step walk in Γ_A from i to j which contains a critical cycle, the cheapest $(k + l_c)$-step walk in Γ_A from i to j is supposed to differ only in one more pass through the cycle. For small k (compared to the number of vertices of Γ_A) this extension of the walk does not have to go through the critical cycle in every case. But it should illustrate the behavior of powers of A.

From that, one would suggest that the maximum cyclicity of A can be n. The cycle with the smallest average weight can go through every vertex only once. But things are getting more complicated when we have a matrix with Γ_A which contains more critical cycles as we illustrate in the following example.

Example 7. Let us have the following matrix:

$$A = \begin{pmatrix} 100 & 1 & 10 & 10 & 10 \\ 1 & 100 & 10 & 10 & 10 \\ 10 & 10 & 100 & 1 & 10 \\ 10 & 10 & 10 & 100 & 1 \\ 10 & 10 & 1 & 10 & 100 \end{pmatrix}$$

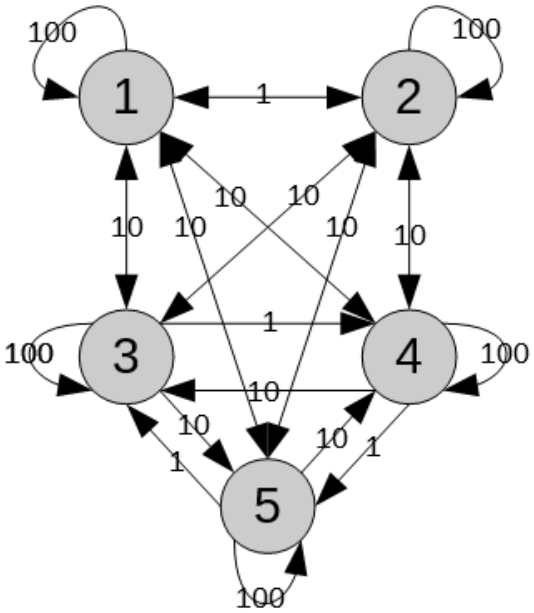

Figure 3.2: The precedence graph of the matrix A from Example 7

It can be seen that Γ_A contains two cycles with the average weight 1. The first one (we will call it c_1) is $1 \to 2 \to 1$, the second one is $3 \to 4 \to 5 \to 3$ (we will call it c_2).

It is clear that when we are powering A to the k for $2 \mid k$ then the cheapest walk $1 \to 1$ of the length k uses only c_1. On the other hand, for $3 \mid k$, the cheapest

walk $3 \to 3$ uses only c_2. When we tried to compute powers of A, we have noticed that the first different powers which are constantly different are 2 and 8 and it holds that $A^{\odot 8} - A^{\odot 2} = 6 \odot (\mathbb{1})_{5\times 5}$. And, as we will describe in Lemma 16, thanks to the fact that both cycles in the critical graph of A are disjoint the cyclicity of A is indeed equal to 6.

From what we have seen so far (and is further described in De Schutter [1999]), we can formulate the following statement:

Claim 15 (Cyclicity estimation)**.** *For a tropical irreducible matrix of size $n \times n$ holds that its cyclicity p is a divisor of the least common multiple of lengths of critical cycles in Γ_A.*

So we have some upper bound for matrix cyclicity. We look at Γ_A, identify critical cycles and compute upper bound as the least common divisor of their lengths. That it is only an upper bound can be seen from the following example of a matrix with intersecting critical cycles.

Example 8. Let us have the following matrix:

$$A = \begin{pmatrix} 100 & 1 & 1 \\ 1 & 100 & 10 \\ 10 & 1 & 100 \end{pmatrix}$$

Then the critical graph of A looks like this:

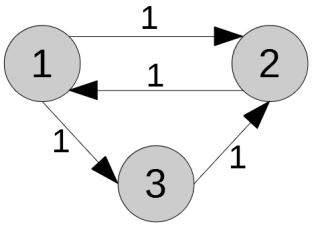

Figure 3.3: The critical graph of the matrix A from Example 8

The matrix A has two critical cycles: $1 \to 2 \to 1$ and $1 \to 2 \to 3 \to 1$. But the cyclicity of A is only 1 because 2 and 3 are co-prime numbers. This follows from Lemma 16 and we can verify it by calculation. We can see that $A^{\odot 6} - A^{\odot 5} = 1 \odot (\mathbb{1})_{3\times 3}$, thus the cyclicity of A is 1.

In De Schutter [1999] in Section 3, there is a guide on how to compute the cyclicity of the given matrix in Section 2. First of all, we need to find a critical graph of Γ_A. Then, for its every connected component, we need to compute its cyclicity as the greatest common divisor of lengths of its elementary cycles. And finally, we have to find the least common multiple of cyclicities of these components.

Lemma 16. *Let A be an irreducible tropical matrix and Γ_A its precedence graph. Let $\mathcal{C}_A$ be the critical graph of Γ_A. Let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be disjoint connected components of $\mathcal{C}_A$. For every component $\mathcal{C}_i$ we can compute its cyclicity p_i as the greatest*

common divisor of lengths of its elementary cycles. The index of cyclicity p_A of the matrix A can be computed as:

$$p_A = lcm(p_1, \ldots, p_m)$$

Further, we can compute $\lambda(A)$, $\lambda(A) \in \mathbb{Q}$ which denotes an average cost per step in a critical cycle of A. Let c_1 be an arbitrary critical cycle in Γ_A. Let l_1 be the number of edges in c_1 and w_1 be a total cost of c_1. The value of $\lambda(A)$ can be computed as:

$$\lambda(A) = \frac{w_1}{l_1}$$

As is described in De Schutter [1999], the cyclicity can be computed with time complexity $\mathcal{O}(n^3)$ where the size of the given matrix is $n \times n$. To compute the cyclicity, it is needed to run three algorithms. One for computing the average weight of a step in a critical cycle. Then we need to construct the critical graph of the matrix. And finally, one can compute the cyclicity from the critical graph by the algorithm constructed in Denardo [1977].

Later, we will use an upper bound for the cyclicity, which is $\exp(\frac{n}{e})$ according to Lemma 3.7 from De Schutter [1999].

Transience of a matrix

To estimate the length of a tail of a quotient semigroup is more complicated. We know that every irreducible matrix is cyclic and therefore for every matrix A there exists k for which every cheapest k-step walk from i to j in Γ_A contains some of the critical cycles. Such k will give us the upper bound for the transience of A.

It can happen that A contains two cycles, one with the smallest average weight and another one with average weight only slightly larger. Eventually, with high enough power of A, every cheapest walk $i \to j$ will contain the critical cycle.

Example 9. Let us slightly change matrix A from Example 7:

$$B = \begin{pmatrix} 100 & 2 & 10 & 10 & 10 \\ 1 & 100 & 10 & 10 & 10 \\ 10 & 10 & 100 & 1 & 10 \\ 10 & 10 & 10 & 100 & 1 \\ 10 & 10 & 1 & 10 & 100 \end{pmatrix}$$

Now, the cycle with the smallest average weight is $3 \to 4 \to 5 \to 3$ and we will call it c_1. The average cost per step of c_1 is 1. The second cheapest cycle is $1 \to 2 \to 1$ (c_2) with an average cost per step equal to 1.5.

Let k be a multiple of 6. We are constructing the k-step cheapest walk from 1 to 1. First, we can try the walk which contains only edges between vertices 1 and 2 – it uses the cycle c_2. We can see that for small k it is really the cheapest k-step walk from 1 to 1. The cost of this walk is $\frac{3k}{2}$.

On the other hand, we can try another walk – which uses the cycle c_1. It starts at 1, goes to 3, cycle in c_1 for $\frac{k-3}{3}$ times and finally goes from 3 to 2 and back to 1. The total cost of this walk is $18 + k$.

When we compare these two walks, we can compute that the second one is cheapest for $k \geq 36$. And by calculation, we can verify that 36 is transience of A

because $A^{\odot 39} - A^{\odot 36} = 3 \odot (\mathbb{1})_{5\times 5}$ and there are no lower powers of A which are constantly different.

From the example, it can be seen that the transience of a matrix depends on the average cost per step of a critical cycle and average cost per step of the cycle with the second lowest cost per step. The next theorem (see original Theorem 4.3 in Nowak and Charron-Bost [2014]) gives us an upper bound for the transience.

Claim 17 (An upper bound for matrix transience)**.** *Let A be an irreducible tropical square matrix of size n. Then the transience of A is at most:*

$$2n^2 + \frac{2n^2\|A\|}{|\lambda(A) - \lambda_2(A)|}$$

where $\|A\|$ is the difference between the maximum and minimum finite entry in the matrix A.

Remark. *In the previous claim, the symbol $\lambda(A)$ means an average weight per step in a critical cycle (as in Lemma 16). And $\lambda_2(A)$ is an average weight per step in a non-critical cycle with the least average weight per step.*

The following proposition provides us the value of c from Cyclicity theorem.

Proposition 18. *Let A be an irreducible matrix with the index of transience t_A and cyclicity p_A. For $k, l \in \mathbb{Z}$ such that $k, l \geq t_A$ and $k \equiv l \pmod{p_A}$ it holds that:*

$$A^{\odot k} = \lambda(A)^{\odot(k-l)} \odot A^{\odot l}$$

Proof. From Theorem 2.1 and Theorem 2.3 of De Schutter [1999] follows that:

$$A^{\odot t_A + p_A} = \lambda(A)^{\odot p_A} \odot A^{\odot t_A} \tag{3.1}$$

We can express k and l in the following manner:

$$k = t_A + u_k p_A + v_k$$
$$l = t_A + u_l p_A + v_l$$

where $u_k, u_l \in \mathbb{N}_0$ and $v_k, v_l \in \{0, \ldots, p_A - 1\}$. This decomposition is unambiguous. Because $k \equiv l \pmod{p_A}$ we know that $v_k = v_l$.

The rest is clear from equality 3.1. □

3.2.3 Quotient semigroups without a cycle

Now we will focus on matrices without a strictly connected precedence graph. Every reducible tropical matrix can be transformed into the following form by renumbering indexes of rows and columns (see Proposition 3.10 in Izhakian and Rowen [2011]). If the matrix is supposed to be reducible, there has to be a tropical zero block in the matrix.

$$\begin{pmatrix} B_1 & C_{1,1} & \ldots & C_{1,(n-1)} \\ \varepsilon & B_2 & \ldots & C_{2,(n-2)} \\ \vdots & \ddots & \ddots & \vdots \\ \varepsilon & \ldots & \varepsilon & B_n \end{pmatrix}$$

Every B_i is a square irreducible matrix and $C_{i,j}$ are rectangle matrices of a suitable size.

A special example of a reducible matrix is the matrix with non-zero blocks only on the diagonal, such as this one:

$$\begin{pmatrix} B_1 & \varepsilon & \dots & \varepsilon \\ \varepsilon & B_2 & \dots & \varepsilon \\ \vdots & \ddots & \ddots & \vdots \\ \varepsilon & \dots & \varepsilon & B_n \end{pmatrix}$$

All B_i's are square irreducible matrices.

Such a matrix creates an infinitely large semigroup in general, but its blocks are independent of each other and are cyclic.

A matrix with non-zero blocks above the diagonal is more complicated. When computing powers of this matrix, blocks on the diagonal stay independent of everything else. Values in blocks above the diagonal are dependent on other entries of the matrix. But as we will show, they behave similarly like cyclic matrices. Let us illustrate this with an example:

Example 10. Let us have a matrix X of size 5×5 in a triangular form with blocks A, B, C which do not contain element ε. A and B are square matrices of sizes 2 and 3. We will denote matrix entries in the following way:

$$X = \begin{pmatrix} A & C \\ \varepsilon & B \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ a_{2,1} & a_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ \varepsilon & \varepsilon & b_{3,3} & b_{3,4} & b_{3,5} \\ \varepsilon & \varepsilon & b_{4,3} & b_{4,4} & b_{4,5} \\ \varepsilon & \varepsilon & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix}$$

We can construct a precedence graph of matrix X. It consists of 5 vertices and is composed of two complete subgraphs corresponding to blocks A and B. Edges from block A to block B are given by entries of the block C.

From Γ_X it can be clearly seen that blocks A and B are dependent only on themselves. When computing the k-th power of X, all walks of length k which both start and end in the block A contain only vertices 1 and 2 due to the absence of edges from B to A. Similarly for the block B.

What happens with block C? The entries in the k-th power of X corresponding to block C are given as a weight of the cheapest k-step walk from one of the vertices of A to one of the vertices of B. Again, thanks to the absence of edges from down to up, we can see that such a walk consists of i steps A, one step on a C edge and $n - i - 1$ steps in B.

Then for high enough k, every k-step cheapest walk from a (vertex in block A) to b (vertex in block B) will contain some of the critical cycles. This means that when comparing blocks of X^k and X^{k+p} (for $p = \mathrm{lcm}(p_A, p_B)$ where p_A is cyclicity of A and p_B is cyclicity of B), the blocks are constantly different. But the constant differs for each block.

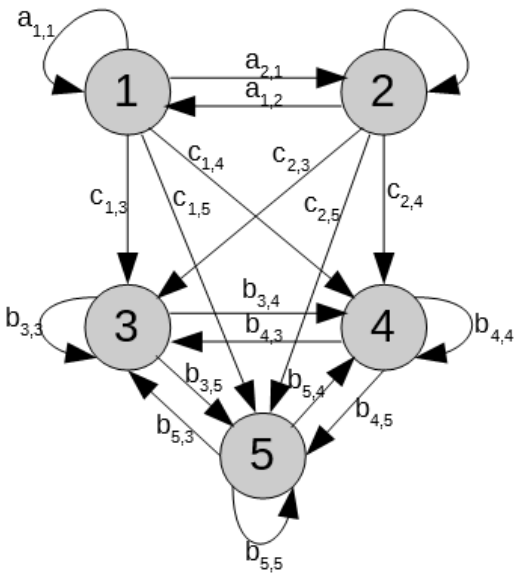

Figure 3.4: The precedence graph of matrix X from Example 10

Definition 20 (Partially constant matrix)**.** *We call the matrix partially constant when it is in an upper-diagonal form, it has square blocks on diagonal (other blocks are given by the sizes of diagonal blocks) and its every block is constant.*

We will try to show that there is a relation for reducible matrices similar to the congruence $\equiv$ for the irreducible matrices. For simplicity, we will focus only on matrices in the upper triangular form with two blocks on the diagonal. We believe that it can be proven that matrices with more blocks on the diagonal behave similarly.

Hypothesis 1. *Let us have a reducible tropical matrix X in the upper triangular form with two blocks on diagonal.*

$$X = \begin{pmatrix} A & C \\ \varepsilon & B \end{pmatrix}$$

There exist $n_0 \in \mathbb{N}$, $\alpha, \beta, \gamma \in \mathbb{Z}$ and $p \in \mathbb{N}$ such that for every $n \geq n_0$ it holds that:

$$X^{\odot(n+p)} = \begin{pmatrix} \alpha^{\odot p} \odot \widetilde{A} & \gamma^{\odot p} \odot \widetilde{C} \\ \varepsilon & \beta^{\odot p} \odot \widetilde{B} \end{pmatrix}$$

where

$$X^{\odot(n)} = \begin{pmatrix} \widetilde{A} & \widetilde{C} \\ \varepsilon & \widetilde{B} \end{pmatrix}$$

Moreover, it holds that $\gamma = \alpha + \beta$.

We will give some arguments about why this hypothesis should hold.

The claim is clear for the blocks on diagonal thanks to the fact that their values are independent on other entries of X. Blocks A and B are cyclic matrices and we can take n_0 as the maximum of transiences of A and B. For a matrix with

non-zero blocks only in the diagonal the value of p is the least common multiple of cyclicities of these blocks (as follows from Izhakian and Rowen [2011]). We suppose that this stays true also for the block above the diagonal.

It is likely that the following should be true for the value of γ. It seems that $\gamma = \alpha \oplus \beta$ (γ is minimum of α and β). We know that α is an average weight of one step in a critical cycle of the matrix A. And accordingly, β is an average weight of one step in a critical cycle of matrix B.

How do entries of $\widetilde{C}$ look like? From Figure 3.4 it is clear that they correspond to the weight of the cheapest n-step walk from some point in the subgraph of A to some point in the subgraph of B. First a steps are in A, one step is an edge from A to B and then there are b steps in B (and it holds that $a + 1 + b = n$). Because n is big enough, the walk should use critical cycles in A or B. If $\alpha > \beta$, then it most likely uses critical cycles in B and vice versa. It seems to be true, that when computing a bigger power of X the walk uses more steps in the better critical cycle. From that, we think that $\gamma = \alpha \oplus \beta$.

We can prove the following claim. When Hypothesis 1 is true then there exists a relation for reducible matrices similar to the congruence $\equiv$.

Claim 19. *Let us have some reducible tropical matrix X in the upper irreducible form with two blocks on the diagonal.*

$$X = \begin{pmatrix} A & C \\ \varepsilon & B \end{pmatrix}$$

Suppose that there exist $n, p \in \mathbb{N}$, $\alpha, \beta \in \mathbb{Z}$ such that:

$$X^{\odot(n+p)} = \begin{pmatrix} \alpha \odot \widetilde{A} & (\alpha \oplus \beta) \odot \widetilde{C} \\ \varepsilon & \beta \odot \widetilde{B} \end{pmatrix}$$

where

$$X^{\odot n} = \begin{pmatrix} \widetilde{A} & \widetilde{C} \\ \varepsilon & \widetilde{B} \end{pmatrix}$$

Then the following holds:

$$X^{\odot(n+p+1)} = \begin{pmatrix} \alpha \odot \widetilde{A'} & (\alpha \oplus \beta) \odot \widetilde{C'} \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix}$$

for

$$X^{\odot(n+1)} = \begin{pmatrix} \widetilde{A'} & \widetilde{C'} \\ \varepsilon & \widetilde{B'} \end{pmatrix}$$

Proof. Our interest is only in the block over the diagonal because the values in the blocks on the diagonal are given only by matrix A (respectively B). And the behavior is given by the cyclicity of A (respectively B).

First of all, from

$$X^{\odot n} \odot X = X \odot X^{\odot n}$$

it holds that

$$A \odot \widetilde{C} \oplus C \odot \widetilde{B} = \widetilde{A} \odot C \oplus \widetilde{C} \odot B = \widetilde{C'}$$

Let us suppose that $\beta \leq \alpha$. Then $\alpha \oplus \beta = \beta$. Now we can compute $X^{\odot(n+p+1)}$ as:

$$X^{\odot(n+p+1)} = X \odot X^{\odot(n+p)} = \begin{pmatrix} A & C \\ \varepsilon & B \end{pmatrix} \odot \begin{pmatrix} \alpha \odot \widetilde{A} & (\alpha \oplus \beta) \odot \widetilde{C} \\ \varepsilon & \beta \odot \widetilde{B} \end{pmatrix} =$$
$$= \begin{pmatrix} \alpha \odot \widetilde{A'} & (\alpha \oplus \beta) \odot A \odot \widetilde{C} \oplus \beta \odot C \odot \widetilde{B} \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix} =$$
$$= \begin{pmatrix} \alpha \odot \widetilde{A'} & \beta \odot (A \odot \widetilde{C} \oplus C \odot \widetilde{B}) \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix} = \begin{pmatrix} \alpha \odot \widetilde{A'} & \beta \odot \widetilde{C'} \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix}$$

When $\alpha < \beta$ we can get the following equality in a similar way as the previous one:

$$X^{\odot(n+p+1)} = X \odot X^{\odot(n+p)} = \begin{pmatrix} \alpha \odot \widetilde{A'} & \alpha \odot \widetilde{C'} \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix}$$

The theorem is proven and it holds that

$$X^{\odot(n+p+1)} = \begin{pmatrix} \alpha \odot \widetilde{A'} & (\alpha \oplus \beta) \odot \widetilde{C'} \\ \varepsilon & \beta \odot \widetilde{B'} \end{pmatrix}$$

□

Definition 21. *Let us have two square reducible tropical matrices X, Y of the same size, both composed of 4 blocks. We denote blocks on the diagonal of X as X_1 and X_2, the block above the diagonal is marked as X_3.*

We will denote $X \equiv_{\alpha,\beta} Y$ if there exist $\alpha, \beta \in \mathbb{Z}$ such that:

$$Y = \begin{pmatrix} \alpha \odot X_1 & (\alpha \oplus \beta) \odot X_3 \\ \varepsilon & \beta \odot X_2 \end{pmatrix}$$

The defined relation $\equiv_{\alpha,\beta}$ is not congruence but it is only equivalence.

4. Back to Fast Stickel's protocol

In the previous chapter, we got a theoretical explanation for the reason why the simple attack on Protocol 11 was so successful.

Let us recall the essence of Fast Stickel's protocol (Algorithm 11). Tropical square matrices A and B are publicly known. Party `A` choose two random positive integers a_1 and a_2 which are its secret key and computes the matrix $U = A^{\odot a_1} \odot B^{\odot a_2}$ and sends it to `B`. Party `B` choose two random positive integers b_1 and b_2 which are its secret key and computes $V = A^{\odot b_1} \odot B^{\odot b_2}$ and sends it to `A`. The common key can be obtained as $K = A^{\odot a_1} \odot V \odot B^{\odot a_2} = A^{\odot b_1} \odot U \odot B^{\odot b_2}$.

4.1 Irreducible matrices

Matrices tested in Protocol 11 were irreducible with low matrix entries and thanks to that the quotient semigroups had short tails. So the attack always works if chosen exponents a_i, b_i are greater than the length of a tail of the corresponding matrix. When the chosen coefficients are not bigger than lengths of tails, the attack fails.

Algorithm 20 (Attack on Fast Stickel's protocol)**.** *The input of the algorithm are matrices A, B, U, V of size $n \times n$. Both matrices A and B are irreducible. U and V are matrices computed in Protocol 11.*

- *Find the cyclicities p_A, p_B of the matrices A and B*
- *Find the value v – the difference between the maximal and minimal finite entry in A and B.*
- *Put $l_A = l_B = 2n^2(1 + n^2 v)$.*
- *Try to find such integers $i_A \in \{0, \ldots, p_A - 1\}$ and $i_B \in \{0, \ldots, p_B - 1\}$ that $T = U - A^{\odot(l_A + i_A)} \odot B^{\odot(l_B + i_B)}$ is a constant matrix. If no such i_A, i_B exist, return with failure.*
- *Let $T = c \odot (\mathbb{1})_{n \times n}$. Then for $X = c \odot A^{\odot(l_A + i_A)}, Y = B^{\odot(l_B + i_B)}$ we get the secret key K as*

$$K = X \odot V \odot Y$$

Theorem 21 (Algorithm 20 works)**.** *Let t_A be transience of the matrix A and t_B the index of transience of B. In the case that $a_1 \geq t_A$ and $a_2 \geq t_B$, Algorithm 20 finds the secret key computed by Fast Stickel's protocol.*

Proof. First of all we need to check that $l_A \geq t_A$ and $l_B \geq t_B$. Let us recall the upper bound for the index of transience of an irreducible matrix from Claim 17. It is $2n^2 + \frac{2n^2 \|A\|}{|\lambda(A) - \lambda_2(A)|}$.

What is the least possible value of denominator? Both $\lambda(A)$ and $\lambda_2(A)$ can be expressed as $\lambda(A) = \frac{k}{m}$, $\lambda_2(A) = \frac{k_2}{m_2}$ where $m, m_2 \in \mathbb{N}$ are the lengths of some cycles and $k, k_2 \in \mathbb{Z}$ are the costs of the cycles. And it holds that $\lambda(A) < \lambda_2(A)$. Further, it has to be true that $m \leq n$, $m_2 \leq n$. From all this, we know that $|\lambda(A) - \lambda_2(A)| \geq \frac{1}{n^2}$.

And that is the reason why $l_A = l_B = 2n^2(1 + n^2v) \geq \max\{t_A, t_B\}$.

The matrix U was computed as $U = A^{\odot a_1} \odot B^{\odot a_2}$. Because we are searching through the whole interval of cyclicity and we are above transiences, there has to be such $i_A \in \{0, \ldots, p_A - 1\}$ that $A^{\odot a_1} \equiv A^{\odot(l_A + i_A)}$ and such $i_B \in \{0, \ldots, p_B - 1\}$ that $B^{\odot a_2} \equiv B^{\odot(l_B + i_B)}$. Therefore $X \odot Y = U$. □

Theorem 22 (Time complexity of Algorithm 20)**.** *The time complexity of Algorithm 20 is in* $\mathcal{O}(n^3(\exp(\frac{2n}{e}) + \log(n)\log(v)))$ *where* n *is the size of the matrices and* v *is the difference between the maximal and minimal finite matrix entry.*

Proof. In the first step of the algorithm, we need to compute the cyclicities of both matrices. It can be done in $\mathcal{O}(n^3)$ (De Schutter [1999]).

Further, we need to compute $A^{\odot l_A}$ (and $B^{\odot l_B}$, which has the same complexity). Because $l_A = 2n^2(1 + n^2v)$ we need to perform $\log(2n^2(1 + n^2v))$ matrix multiplications which is in $\mathcal{O}(n^3 \log(n) \log(v))$.

In the fourth step, we need to try p_A values of i_A and p_B values of i_B. The upper bound for matrix cyclicity is $\exp(\frac{n}{e})$ according to De Schutter [1999]. Altogether, we need to process at most $\exp(\frac{2n}{e})$ matrix multiplications which has the complexity $\mathcal{O}(\exp(\frac{2n}{e})n^3)$. We have precomputed values of $A^{\odot l_A}$ and $B^{\odot l_B}$, we can also precompute first p_A powers of A (and p_B powers of B) in time $\mathcal{O}(\exp(\frac{n}{e})n^3)$.

Altogether the complexity of Algorithm 20 is in $\mathcal{O}(n^3(\exp(\frac{2n}{e}) + \log(n)\log(v)))$. □

When we compare the complexity of the attack with the complexity of Protocol 11, which is $\mathcal{O}(n^3 \log(d))$, then the attack seems to be inappropriate. But the complexity of the attack mainly depends on the fourth step of Algorithm 20. A randomly generated matrix would not have exponentially large cyclicity because it is not likely that there would be more disjoint critical cycles of large lengths in a random matrix. If we consider that there is only one critical cycle in the randomly generated matrix, then the cyclicity will be at most n and the complexity of the attack will be in $\mathcal{O}(n^5 \log(v))$. That is only quadratically worse in n than the key agreement protocol. And moreover, the complexity of the attack is independent on d (though it depends on the logarithm of the size of the matrix entries).

From the previous consideration, Protocol 11 is not suitable for practical use when the matrices are randomly generated. If we want to use Protocol 11 we need to select special matrices with high cyclicity.

4.2 Reducible matrices

If we would like to avoid the attack on the cyclicity, one could suggest generating reducible matrices. These can not be used in a simple attack, are they? We have come up with a way how to break the scheme with reducible matrices in some cases. We slightly modified Algorithm 20 to be able to work with partially different matrices.

The key agreement algorithm with reducible matrices works similarly as the algorithm with irreducible ones. We generate two matrices A, B in the following form:

$$A = \begin{pmatrix} A_1 & A_3 \\ \varepsilon & A_2 \end{pmatrix}, B = \begin{pmatrix} B_1 & B_3 \\ \varepsilon & B_2 \end{pmatrix}$$

where A_1, B_1 are square matrices of size $n_1 \times n_1$, A_2, B_2 are square matrices of size $n_2 \times n_2$ and A_3, B_3 are matrices of size $n_1 \times n_2$.

Let t_{A_i} be transience of the diagonal block A_i and let t_{B_i} be the index of transience of the diagonal block B_i. In the rest of the chapter, we will denote $t_A = \max\{t_{A_1}, t_{A_2}\}$ and $t_B = \max\{t_{B_1}, t_{B_2}\}$.

Similarly for the cyclicities: let p_{A_i} be cyclicity of the diagonal block A_i and let p_{B_i} be the index of cyclicity of the diagonal block B_i. We will denote $p_A = \mathrm{lcm}\{p_{A_1}, p_{A_2}\}$ and $p_B = \mathrm{lcm}\{p_{B_1}, p_{B_2}\}$. Now p_A is something like an equivalent of the cyclicity of A, similarly for the matrix B.

Further, we will denote by λ_i the average cost of one step in a critical cycle of the block A_i and by μ_i the average cost of one step in a critical cycle of the block B_i.

We were thinking about alternating the attack on the protocol with irreducible matrices to be able to work also with the reducible ones. But then, we have noticed that the scheme with reducible matrices is simply breakable by linear algebra.

4.2.1 Attack on the scheme with reducible matrices

We have constructed the following algorithm for breaking the Fast Stickels's protocol with reducible matrices:

Algorithm 23 (Attack on Fast Stickel's protocol with reducible matrices)**.** *The input of the algorithm are matrices A, B, U, V of size $n \times n$. A and B are reducible with 2 blocks on the diagonal. U and V are matrices computed in Protocol 11.*

1. *Find the cyclicities $p_{A_1}, p_{A_2}, p_{B_1}, p_{B_2}$ of all the corresponding matrices given by blocks A_1, A_2, B_1, B_2. During the process, find $\lambda_1, \lambda_2, \mu_1, \mu_2$ as the average costs of one step in the critical cycles of the corresponding blocks.*

2. *Find the value p_A as the least common multiple of p_{A_1}, p_{A_2}. And accordingly p_B for the matrix B.*

3. *Put $l_A = l_B = 2n^2(1 + n^2 v)$.*

4. *Try to find all pairs of integers (i_A, i_B) where $i_A \in \{l_A, \ldots, l_A + p_A - 1\}$ and $i_B \in \{l_B, \ldots, l_B + p_B - 1\}$ for which the matrix $T = U - A^{\odot(i_A)} \odot B^{\odot(i_B)}$ has constant matrices on the diagonal. If no such i_A, i_B exist, output with failure.*

5. *For every such a pair (i_A, i_B) when T has constant matrices on the diagonal, do:*

   - *Try to find an integer solution of the following system with unknowns a_1, a_2:*

$$\begin{pmatrix} \lambda_1 & \mu_1 \\ \lambda_2 & \mu_2 \end{pmatrix} \begin{pmatrix} a_1 - i_A \\ a_2 - i_B \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$$

where α_1 is the value of constants in the first diagonal block of T and α_2 is the value of constants in the other diagonal block of T.

- *If no integer solution exists, try another pair of i_A, i_B.*
- *Verify whether $A^{\odot a_1} \odot B^{\odot a_2} = U$ holds. If not, try another pair (i_A, i_B). If yes, output with SUCCESS and returns the values of a_1 and a_2.*

6. If the algorithm has not ended yet, output with failure.

This algorithm has no guaranteed success if the powers of matrices A and B are not high enough or if the matrix from the linear system which we are solving in the 5-th step is not regular.

Theorem 24 (Algorithm 23 works)**.** *Algorithm 23 breaks the Fast Stickel's protocol when $a_1 \geq t_A$, $a_2 \geq t_B$ and $\lambda_1\mu_2 \neq \mu_1\lambda_2$.*

Proof. Let us have the matrix $U = A^{\odot a_1} \odot B^{\odot a_2}$ where $a_1 \geq t_A$ and $a_2 \geq t_B$.

Thanks to the fact that $p_A = \operatorname{lcm}\{p_{A_1}, p_{A_2}\}$ and $p_B = \operatorname{lcm}\{p_{B_1}, p_{B_2}\}$, there have to exist $i_A \in \{l_A, \ldots, l_A + p_A - 1\}$ and $i_B \in \{l_B, \ldots, l_B + p_B - 1\}$ such that

$$\begin{aligned} a_1 &\equiv i_A \pmod{p_{A_i}} \qquad i = 1, 2 \\ a_2 &\equiv i_B \pmod{p_{B_2}} \qquad i = 1, 2 \end{aligned} \tag{4.1}$$

when $l_A \geq t_A$ and $l_B \geq t_B$.

Thus we can decompose a_1, a_2, i_A and i_B in the following manner:

$$\begin{aligned} a_1 &= t_{A_i} + k_{A_i} p_{A_i} + c_{A_i} \qquad i = 1, 2 \\ a_2 &= t_{B_i} + k_{B_i} p_{B_i} + c_{B_i} \qquad i = 1, 2 \\ i_A &= t_{A_i} + k'_{A_i} p_{A_i} + c_{A_i} \qquad i = 1, 2 \\ i_B &= t_{B_i} + k'_{B_i} p_{B_i} + c_{B_i} \qquad i = 1, 2 \end{aligned}$$

where all $k_{A_i}, k_{B_i}, k'_{A_i}, k'_{B_i} \in \mathbb{N}_0$, all $c_{A_i} \in \{0, \ldots, p_{A_i} - 1\}$ and similarly all $c_{B_i} \in \{0, \ldots, p_{B_i} - 1\}$.

Now we will proceed simultaneously for both diagonal blocks. By U_i we mean the i-th diagonal block of U. When we look at the differences of U_i and $A_i^{\odot i_A} \odot B_i^{\odot i_B}$, we can see from Proposition 18 that for $i \in \{1, 2\}$ it holds:

$$\begin{aligned} U_i - A_i^{\odot i_A} \odot B_i^{\odot i_B} &= A_i^{\odot a_1} \odot B_i^{\odot a_2} - A_i^{\odot i_A} \odot B_i^{\odot i_B} = \\ &= \lambda_i^{\odot (a_1 - i_A)} \odot \mu_i^{\odot (a_2 - i_B)} \odot A_i^{\odot i_A} \odot B_i^{\odot i_B} - A_i^{\odot i_A} \odot B_i^{\odot i_B} = \\ &= \left(\lambda_i^{\odot (a_1 - i_A)} \odot \mu_i^{\odot (a_2 - i_B)}\right) \odot (\mathbb{1})_{n_1 \times n_1} \end{aligned} \tag{4.2}$$

That means that we can find a matrix T which has constant blocks on the diagonal.

$$T = \begin{pmatrix} \alpha_1 \odot (\mathbb{1})_{n_1 \times n_1} & T_3 \\ \varepsilon & \alpha_2 \odot (\mathbb{1})_{n_2 \times n_2} \end{pmatrix}$$

We can rewrite equations 4.2 into standard algebra and we will get the following:

$$\lambda_i(a_1 - i_A) + \mu_i(a_2 - i_B) = \alpha_i \qquad i = 1, 2$$

These are giving us the system of two equations with unknown a_1 and a_2

$$\begin{pmatrix} \lambda_1 & \mu_1 \\ \lambda_2 & \mu_2 \end{pmatrix} \begin{pmatrix} a_1 - i_A \\ a_2 - i_B \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} \tag{4.3}$$

Values of a_1 and a_2 are uniquely determined by α_1 and α_2 whenever the first matrix has non-zero determinant, therefore we need $\lambda_1\mu_2 \neq \mu_1\lambda_2$. □

The test of equality $U = A^{\odot a_1} \odot B^{\odot a_2}$ in the 5-th step of Algorithm 23 is performed because we do not know whether congruences 4.1 follow from the solvability of system 4.3 in integers.

Theorem 25 (Time complexity of Algorithm 23)**.** *The time complexity of Algorithm 23 is in* $\mathcal{O}(n^3(\exp(\frac{2n}{e}) + \log(v)\log(n)))$*, where* n *is a size of matrices and* v *is the difference between the maximal and minimal finite matrix entry.*

Proof. Many steps of this algorithm are similar to the previous one. So from the proof of Theorem 22, we know that the time complexity of the first two steps is in $\mathcal{O}(n^3)$. Also, we know that there are $\exp(\frac{2n}{e})$ candidates for the matrix T. We can precompute $A^{\odot l_A}$ and $B^{\odot l_B}$ and also first p_A (and p_B) powers of matrices A (and B) in time $\mathcal{O}(n^3(\exp(\frac{2n}{e}) + \log(v)\log(n)))$. But thanks to that, the complexity of the computation of the matrix T consists only from three matrix multiplications.

The complexity of the 5-th step is not important for the total complexity of Algorithm 23 because we just have to solve the linear system of the constant number of equations and variables.

Altogether the complexity of Algorithm 23 is in $\mathcal{O}(n^3(\exp(\frac{2n}{e})+\log(v)\log(n)))$. □

From the comparison of complexities of Algorithm 20 and Algorithm 23, we can see that the reducibility of the input matrices does not change the time complexity of the attack on Fast Stickel's protocol.

5. Computational problems with tropical matrices

In this section, we would like to present some of the computational problems which can be examined in the semigroup of tropical matrices. One of the most famous problems is the discrete logarithm problem (DL), see Definition 13. In the introductory section, we have also introduced readers to the definition of Diffie-Hellman protocol. It is based on the Diffie-Hellman problem:

Definition 22 (Computational Diffie-Hellman problem)**.** *Let G be a cyclic group with a generator g. Then CDH problem is: for the given $h_1, h_2 \in G$, where for $x, y \in \mathbb{N}$ is $h_1 = g^x$, $h_2 = g^y$, compute $h \in G$ such that $h = g^{xy}$.*

This problem is believed to be hard (in some structures such as cyclic groups with multiplication modulo a prime number). But surprisingly, it makes sense to define also the following problem:

Definition 23 (Decisional Diffie-Hellman problem)**.** *Let G be a cyclic group with a generator g. Then DDH problem is: for the given $h_1, h_2, h \in G$, where for $x, y, z \in \mathbb{N}$ is $h_1 = g^x$, $h_2 = g^y$ and $h = g^z$ decide if $g^z = g^{xy}$.*

Together with the discrete logarithm (DL) problem, we have three similar problems and we can be interested in the relations between them. What is known is the following proposition (for more info and propositions with proofs see Bellare and Rogaway [2005]:

Claim 26. *For the fixed group $G = \langle g \rangle$ it holds that if you can solve DL problem effectively then you can solve effectively also CDH problem. And if you can solve CDH problem effectively you can also effectively solve DDH problem.*

Unfortunately, there is not too much known about the opposite direction of relations between these problems in a general group G.

Diffie-Hellman problem can be generalized for the usage in semigroups, see for example Maze et al. [2005]. That leads us to the following problems in the semigroup generated by tropical matrices. These problems are similar to the original Diffie-Hellman problems.

Definition 24 (Semigroup Diffie-Hellman related problems)**.** *Let $\mathcal{S} = (S, \cdot)$ be a semigroup. Let $\mathcal{A}$ and $\mathcal{B}$ be one-generated subsemigroups of $\mathcal{S}$, thus $\mathcal{A} = \langle a \rangle$ and $\mathcal{B} = \langle b \rangle$ where $a, b \in S$. Let $\mathcal{C}$ be a set defined as*

$$\mathcal{C} = \mathcal{A} \cdot \mathcal{B} = \{uv \mid u \in \mathcal{A}, v \in \mathcal{B}\}$$

Then we can state the following problems:

(P1) Given $c \in \mathcal{C}$, find such $i, j \in \mathbb{N}_0$ that $c = a^i b^j$.

(P2) Given $c \in \mathcal{C}$ find such $a_1 \in \mathcal{A}$, $b_1 \in \mathcal{B}$ that $c = a_1 b_1$.

(P3) Given $c \in \mathcal{C}$ and $d \in \mathcal{C}$ find such $e \in \mathcal{C}$ that $e = a_1 a_2 b_2 b_1$ where $c = a_1 b_1$ and $d = a_2 b_2$.

As we can recall from previous chapters, the third problem is the exact foundation stone of Stickel's protocol. We will see that the decomposition in (*P3*) is ambiguous but the result is independent on the exact decomposition of c and d.

In our case, the semigroup $\mathcal{S}$ is the semigroup of square tropical matrices of the size $n \times n$ over tropical algebra semiring. Semigroups $\mathcal{A}$ and $\mathcal{B}$ consist of powers of some tropical matrices.

Example 11. The decomposition of tropical matrices from semigroup $\mathcal{C}$ is ambiguous. Let us have matrices $A, B \in \mathcal{S}$ such that:

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$
$$B = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$$

We will denote $\mathcal{A} = \langle A \rangle$ and $\mathcal{B} = \langle B \rangle$. It is clear that $A \odot B \neq B \odot A$. Let $\mathcal{C}$ be a semigroup $\mathcal{C} = \mathcal{A} \odot \mathcal{B}$. Let us have a matrix $C = A^{\odot 3} \odot B^{\odot 2}$.

$$C = \begin{pmatrix} 6 & 6 \\ 5 & 5 \end{pmatrix}$$

We can see that there is another decomposition of C: $C = A^{\odot 1} \odot B^{\odot 4}$.

Remark. *Let us notice that we can define an operation $\star$ on $\mathcal{C}$. For $C, D \in \mathcal{C}$, $C = A_1 \odot B_1$, $D = A_2 \odot B_2$ where $A_1, A_2 \in \mathcal{A}$, $B_1, B_2 \in \mathcal{B}$ we have:*

$$C \star D = A_1 \odot A_2 \odot B_2 \odot B_1$$

The result of $\star$ is independent on the decomposition of matrices C, D. It can be shown that this operation is commutative and associative.

How these problems are related is obvious from one side. (*P1*) is at least as hard as (*P2*) and (*P2*) is at least as hard as (*P3*) in every semigroup, not only in tropical matrices. But is there any difference in their complexity in tropical algebra semigroup? And which are the best estimations which we have on the complexities of these problems?

We were able to determine the complexity of solving the discrete logarithm problem in the cyclic tropical matrix semigroup. That means we want to find such $i \in \mathbb{N}_0$ that for the given $A_1 \in \mathcal{A}$ it holds that $A^{\odot i} = A_1$. The discrete logarithm problem is known to be hard in some structures (for example a finite field with operations modulo large prime number). But in semigroups, it depends on the concrete semigroup (and its operation – semigroup action, see Maze et al. [2005]). We would like to explore the semigroup of tropical matrices.

For the simplicity, we will stay focused only on irreducible tropical matrices. As we have shown in the previous chapter in the attack on Fast Stickel's protocol, it is very likely that the reducibility of matrices does not affect the complexity of the problem. And the complexity of such an algorithm is the same or similar to the complexity of an algorithm which works only with irreducible matrices.

Because powers of tropical matrices tend to behave linearly (values of all entries are increasing or decreasing linearly) due to the cyclicity, the discrete logarithm problem in the tropical matrix semigroup seems to be easy to solve.

Claim 27 (Discrete logarithm problem for tropical matrices)**.** *For the given irreducible matrix $A_1 \in \mathcal{A}$ we can find $i \in \mathbb{N}_0$ such that $A^{\odot i} = A_1$ in time $\mathcal{O}(n^7 \log(v) + \exp(\frac{n}{e})n^3)$.*

Proof. What is certainly needed is to compute the cyclicity of A, denoted as p_A, and $\lambda(A)$ – cost per step in a critical cycle of A. This can be done in $\mathcal{O}(n^3)$. Now we need to find out whether the power of A_1 is above the transience of A or not. We can test if $A_1 \odot A^{\odot p_A} - A_1$ is a constant matrix. Because the upper bound for matrix cyclicity is $\exp(\frac{n}{e})$, the computation of $A^{\odot p_A}$ takes time in $\mathcal{O}(n^3 \log(\exp(\frac{n}{e}))) = \mathcal{O}(n^4)$. So the check can be done in $\mathcal{O}(n^4)$.

If we are under the index of transience, we just try all possible powers of matrix A until we find the right one. Thanks to the upper bound for transience, which is $\mathcal{O}(n^4 \log(v))$, this can be done in time $\mathcal{O}(n^7 \log(v))$ where v is the difference between the maximal and minimal finite entry of A.

If we are above the transience t_A we can use the cyclicity property. Unfortunately, we do not know anything about the values of matrix entries between $A^{\odot t_A}$ and $A^{\odot t_A + p_A}$. These can increase (or decrease) or they can oscillate. We have to find such ℓ between t_A and $t_A + p_A$ that $A_1 - A^{\odot \ell}$ is a constant matrix. Due to the upper bound for p_A we need to process $\mathcal{O}(\exp(\frac{n}{e}))$ matrix multiplications (and subtractions), so the complexity of this part of the algorithm is in $\mathcal{O}(\exp(\frac{n}{e})n^3)$. We do not need to calculate exact t_A, the upper bound $2n^2(1+n^2v)$ is good enough.

When we have ℓ, we can compute the value of i. Let c be a value of entries in the constant matrix $A_1 - A^{\odot \ell}$. Then $i = \ell + \frac{c}{\lambda(A)}$.

The total complexity of finding the discrete logarithm is in $\mathcal{O}(n^7 \log(v) + \exp(\frac{n}{e})n^3)$. □

We can see that the discrete logarithm problem is hard in the semigroup generated by an irreducible tropical matrix (in the case that there is not a better algorithm than ours). But it relies on the cyclicity of the generator. But as we have discussed in the previous chapters, randomly generated matrices are supposed to have a small index of cyclicity. And it is questionable if it is practical to work with large matrices even if we generate the ones with a high index of cyclicity.

Remark. *The complexity of the discrete logarithm problem is commonly expressed in k where k is a size of a finite cyclic group. Since we are working with infinite large semigroups we have no such parameter. So it is maybe a little surprising that the complexity depends only on the generating matrix A and not on the power of A_1.*

Conclusion

In our work, we were interested in the cryptographic scheme for the key-agreement protocol. The original scheme was introduced by Grigoriev and Shpilrain [2014]. The main objective of our work was to explore this scheme determine its complexity and security. As for the attacks, they are based on the work of Kotov and Ushakov [2018].

What was done

At the beginning of the second chapter, we have recalled tropical Stickel's protocol introduced by Grigoriev and Shpilrain [2014] and have computed its time complexity. Next, we have summarized the Simple heuristic attack (Algorithm 4) introduced by Kotov and Ushakov [2018] and computed its complexity. We have also implemented this attack in Python (see attachments) and run our experiment to see the attack success rate for different sizes of parameters. As was said in Grigoriev and Shpilrain [2014], this attack had not worked when all the values of matrix entries were non-negative.

This problem has a solution in the attack developed also by Kotov and Ushakov [2018]. This attack works fine for the parameters originally designed for the key-agreement protocol. And it is not heuristic, unlike Algorithm 4. We have downloaded the sources of the attack and run it with bigger parameters. As expected, the running time of the algorithm took a big amount of time with a large parameter size. We have confirmed that generally the attack needs to solve the NP-complete problem of finding the minimal cover.

We were able to avoid this in Algorithm 9, but we had to compromise from the 100% success rate. We have obtained an algorithm with high enough success rate which was able to work also for bigger values of parameters. And of course, we have computed the complexity of this attack.

To sum it up, for Stickel's protocol introduced by Grigoriev and Shpilrain [2014] we have heuristic attack which works with high enough success rate. And its complexity is only polynomially worse than the time of the protocol we are attacking.

At the end of the second chapter, we came up with Fast Stickel's algorithm (Algorithm 11). This key-agreement algorithm should have exponentially less time complexity than attacks on Algorithm 4. But during the experiment in Python, we have noticed that the running time of the Simple attack (Algorithm 6) is lower than it should be. And it was due to the matrix cyclicity which we have summarized in the third chapter.

The task of the third chapter was to introduce the reader with the behavior of powers of tropical matrices. This chapter summarized some claims about tropical matrix powers, mainly about matrix cyclicity and transience. We presented some ideas about why should things work the way they appear, but we have not been always able to give exact proofs of our claims.

In the fourth chapter, we have returned back to Fast Stickel's protocol. Using the results from Chapter 3, we have constructed an attack which uses the cyclicity to break Algorithm 11. The time complexity of this attack is exponentially worse than the running time of Algorithm 11, but only for the matrices with a high

index of cyclicity. We have discussed that only specially generated large size matrices would have such a high index of cyclicity. And we would like to use smaller matrices with larger values of other parameters.

Then we tried to change Algorithm 11 to work with reducible matrices which do not generate a cyclic semigroup. However, we have come up with the attack breaking this variant of Fast Stickel's protocol. And this attack is surprisingly trivial. We tried only matrices composed of 2×2 blocks but this attack would also deal with matrices composed of more blocks. The main result of this chapter was the discovery that the (ir)reducibility of the input matrices of Algorithm 11 does not have an impact on the complexity of the attack.

In the last chapter, we have presented some computational problems which can be examined in the semigroup of tropical matrices. The result of this chapter is the estimation of the complexity of the discrete logarithm problem in the semigroup of matrices over tropical algebra semiring.

What has not been done

In the article by Grigoriev and Shpilrain [2014], there is also a suggestion of a scheme for asymmetric encryption. At first we wanted to examine also the protocol for the public-key encryption, but during the work with the Stickel's protocol we have noticed the behavior of the tropical matrix powers and we have come across the theory of the matrix cyclicity. That led us to the other variant of the Stickel's protocol so we have decided to only focus on the scheme for the key-agreement protocol. If we had more time we would have implemented also algorithms and attacks on the Fast Stickel's protocol. For now, we are satisfied by the theoretical description and some calculations for the confirmation that our ideas work.

Inspiration for further research

Except for the asymmetric encryption scheme from Grigoriev and Shpilrain [2014] one could be interested in the questions which we asked in the last chapter. There is needed to find relations between problems (*P1*), (*P2*) and (*P3*) and state their complexity in semirings.

During the process of writing our thesis, authors of Grigoriev and Shpilrain [2014] published a sequel of their work in the article Grigoriev and Shpilrain [2019], where they proposed another key exchange protocols over tropical algebra. Again, their protocols are proposed without any detailed discussion about complexity and justification of the size of parameters.

Bibliography

M. Bellare and P. Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005. URL https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf. cit. 2019-07-18.

N. Bjørner, A. Blass, Y. Gurevich, and M. Musuvathi. Modular difference logic is hard. *arXiv preprint arXiv:0811.0987*, 2008. URL https://web.eecs.umich.edu/~gurevich/Opera/195.pdf. cit. 2019-07-18.

B. Charron-Bost, M. Függer, and T. Nowak. New transience bounds for max-plus linear systems. *Discrete Applied Mathematics*, 219:83–99, 2017. URL http://www.lsv.fr/~mfuegger/papers/CFN17_dam.pdf. cit. 2019-07-18.

P. Chew. Proving NP-completeness. URL http://www.cs.cornell.edu/courses/cs482/2007sp/NPComplete.pdf. cit. 2019-07-18.

B. De Schutter. Upper bounds for the index of cyclicity of a matrix. Technical report, Tech. rep. 98-32, ESATSISTA, KU Leuven, Leuven, Belgium, 1999. URL https://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/98_32.pdf. cit. 2019-07-18.

E. V. Denardo. Periods of connected networks and powers of nonnegative matrices. *Mathematics of Operations Research*, 2(1):20–24, 1977.

R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6): 345, 1962.

S. Gaubert. On rational series in one variable over certain dioids. *Research Report, no. 2162 (1994), INRIA-Rocquencourt, Le Chesnay Cédex*, 1994. URL https://hal.inria.fr/inria-00074510/document. cit. 2019-07-18.

O. W. Gnilke. The semigroup action problem in cryptography. *Thesis submitted to University College Dublin*, 2014. URL https://shannoninstitute.ucd.ie/~semiring/diss_oliver.pdf. cit. 2019-07-18.

D. Grigoriev and V. Shpilrain. Tropical cryptography. *Communications in Algebra*, 42(6):2624–2632, 2014. URL https://eprint.iacr.org/2013/012.pdf. cit. 2019-07-18.

D. Grigoriev and V. Shpilrain. Tropical cryptography II: Extensions by homomorphisms. *Communications in Algebra*, 46:4224–4229, 2019. URL https://eprint.iacr.org/2018/1104.pdf. cit. 2019-07-18.

Z. Izhakian and L. Rowen. Supertropical matrix algebra III: Powers of matrices and their supertropical eigenvalues. *Journal of Algebra*, 341(1):125–149, 2011. URL https://www.sciencedirect.com/science/article/pii/S0021869311003267. cit. 2019-07-18.

M. Kotov and A. Ushakov. Analysis of a key exchange protocol based on tropical matrix algebra. *Journal of Mathematical Cryptology*, 12(3):137–141, 2018. URL https://eprint.iacr.org/2015/852.pdf. cit. 2019-07-18.

D. Maclagan and B. Sturmfels. *Introduction to tropical geometry*, volume 161. American Mathematical Soc., 2015. URL http://www.cs.technion.ac.il/~janos/COURSES/238900-13/Tropical/MaclaganSturmfels.pdf. cit. 2019-07-18.

G. Maze, Ch. Monico, and J. Rosenthal. Public key cryptography based on semigroup actions. *Advances in Mathematics of Communications*, 1(4): 489–507, 2005. URL https://www.math.uzh.ch/fileadmin/user/rosen/publikation/ma07.pdf. cit. 2019-07-18.

A. Miné. The octagon abstract domain. *Higher-order and symbolic computation*, 19(1):31–100, 2006. URL https://www-apr.lip6.fr/~mine/publi/article-mine-HOSC06.pdf. cit. 2019-07-18.

T. Nowak and B. Charron-Bost. An overview of transience bounds in max-plus algebra. *Tropical and Idempotent Mathematics and Applications*, 616:277–289, 2014. URL https://hal.archives-ouvertes.fr/hal-00993630/document. cit. 2019-07-18.

V. Shpilrain. Cryptanalysis of stickel's key exchange scheme. In *International Computer Science Symposium in Russia*, pages 283–288. Springer, 2008.

E. Stickel. A new method for exchanging secret keys. In *Third International Conference on Information Technology and Applications (ICITA'05)*, volume 2, pages 426–430. IEEE, 2005.

A. Attachments

We have implemented a library for basic computations in tropical algebra. Further, we give an implementation of attacks on Stickel's protocol. The code was written for Python 3.7.

A.1 The Python implementation of attacks on Stickel's protocol

Tropical library

We have declared and implemented three classes in the file *tropical_algebra.py*:

- `T_int` is a class for representing the tropical integers. It overloads standard operations and provides their implementation in tropical algebra.
- `Matrix` is a class for representing the tropical matrices. It overloads standard operations and provides their implementation in tropical algebra. The init function can read the matrix from the file or it can generate an identity matrix of a given size.
- `Polynomial` is a class for representing the tropical polynomials over the tropical matrices. It implements functions for reading polynomials from the file or reconstructing it from the coefficient list and the function for evaluating the polynomial in a tropical matrix.

Together with some functions (such as for generating random matrices and polynomials), this library provides the background for Attack 6.

Key agreement library

File *key_exchange.py* contains functions used for the simulation of Stickel's protocol (Algorithm 4) and in an attack to it:

- `generateKeys()` for generating all random matrices and polynomials during the protocol. It takes parameters for maximal and minimal matrix entries and coefficients of polynomials, size of the matrices and maximal allowed degree of the polynomials.
- `computeKeyPart()` for computing matrices U and V.
- `computeCommonKey()` for computing the common key from data known to one of the parties.
- `simpleAttack()` for the run of Algorithm 6, uses also an auxiliary function `checkSolution()` for checking the correctness of the output.

Simple attack

File *simple_attack.py* simulates many runs (the count is given by the constant `RUNS`) of Attack 6 on randomly generated instances of Stickel's protocol. There are 4 parameters:

- The parameter `n`. This parameter determines the size of the matrices. Algorithm runs for sizes in $\{5, 10, \ldots, 5(n-1)\}$.
- The parameter `d`. This parameter determines the maximal allowed degree of polynomials. Algorithm runs for sizes in $\{5, 10, \ldots, 5(d-1)\}$.
- The parameter `c_poly`. This parameter determines the size of the polynomial coefficients. It is in the interval $\{-10^{\text{c\_poly}}, \ldots, 10^{\text{c\_poly}} - 1\}$.
- The parameter `c_matrix`. This parameter determines the size of the matrix entries. It is in the interval $\{-10^{\text{c\_matrix}}, \ldots, 10^{\text{c\_matrix}} - 1\}$.

The algorithm writes into the console the progress (values of tested `n` and `d`) together with a result of the attack run (1 = success, 0 = failed). The results are saved into the file *simpleAttackResults* in the following format:

`R;n;d;c_p;c_m;T_1;T_2`

where `R` is the result of the run, `c_p` is equal to $10^{\text{c\_poly}}$, `c_m` is equal to $10^{\text{c\_matrix}}$, `T_1` is the time taken by the key-agreement protocol and `T_2` is the time taken by the attack run.

Regular attack

File *attack.py* simulates many runs (the count is given by the constant `RUNS`) of Attack 9 on randomly generated instances of Stickel's protocol. The structure is similar to the simple attack. There are 4 parameters but only `c_poly` and `c_matrix` differ from the simple attack:

- The parameter `c_poly`. This parameter determines the size of the polynomial coefficients. It is in the interval $\{0, \ldots, 10^{\text{c\_poly}} - 1\}$.
- The parameter `c_matrix`. This parameter determines the size of the matrix entries. It is in the interval $\{0, \ldots, 10^{\text{c\_matrix}} - 1\}$.

The algorithm writes into the console the progress (values of tested `n` and `d`) together with a result of the attack run (1 = success, 0 = failed). The results are saved into the file *atackResults* in the same format as in *simpleAttackResults*.

To run this attack, it is required to have the following two third-party libraries installed:

- *Numpy* for Python 3.7.
- *z3* – we have used the verison 4.6.0 for 32 bit system. Compiled library for Windows can be downloaded from GitHub [1].

[1] https://github.com/Z3Prover/z3