LENARTOWICZ
Sacha
ET4 Informatique

Internship report
Engineering Assistant

Engineering program at Polytech Paris Sud – 4th year

From 2017-05-15 to 2017-08-04

Internship coordinator : LAVERGNE Thomas
E-mail : thomas.lavergne@limsi.fr

"Design, develop and realize crypto primitives for nonstandard applications"

Novosibirsk State University
Address: 1, Pirogova str., Novosibirsk, 630090, Russia
Telephone number: +7 (383) 363-42-92
E-mail : interstudy@lab.nsu.ru
From 2017-05-15 to 2017-08-04
Supervisor: Sergey F. KRENDELEV

Summary

1 Work environment 3
1.1 Novosibirsk history 3
1.2 Novosibirsk nowadays 4
1.3 Novosibirsk State University 4
1.4 Sergey F. Krendelev 5
2 Mission and goals 7
2.1 The key exchange problem 7
2.2 Tasks 8
3 Common work 9
3.1 Original Diffie-Hellman key exchange 9
3.1.1 Protocol description 9
3.1.2 Issues 10
3.2 Pre-requisite 11
3.2.1 Choosing commutative matrices 11
3.2.2 Grassmann algebra 11
3.3 Our protocol 12
3.4 Security analysis 14
3.4.1 Qualitative security Analysis 14
3.4.2 Quantitative security analysis 14
3.5 Block cypher over Grassmann Algebra 15
4 My Interface 17
4.1 Presentation of the interface 17
4.2 New classes 18
4.3 Possible improvements 20
Conclusion 21
Bibliography 22
Annex I: Scientific article 23

Thanks

I want to thank my supervisor Mr. Sergey F. Krendelev for allowing me to make this internship, even though I explained my knowledge of cryptography and mathematics was limited. I also thank him for giving me independence and for leaving me the choice to work in a team or individually. I discovered what scientific research means and he helped me when it was required.

I also thank Michèle Debrenne, who gave me information about Novosibirsk State University and helped me to make contact with the University and Mr. Krendelev. My thanks also go to Ms. Kseniya Shmugurova, Ms. Yulia Chernenko and Ms. Natalia Yazykova from the International Office, who helped me for administrative formalities and Mr. Thomas Lavergne who took time to come to the University to visit us. Lastly, I thank my new friend Mujtaba Alnashi for showing us more about Russia.

1 Work environment

1.1 Novosibirsk history

Novosibirsk is a pretty recent city, considered a town since 1903. It was built near the railway bridge which cross the river of Ob and support the Trans-Siberian Railway. It was first named Novonikolayevsk. As you can see in the *Figure 1*, it is located midway between Europe and Pacific Ocean, along the Ob River, coming from mountains of Altai.

Figure 1 - Novosibirsk's geographic position (source: BBC's website)

Thanks to its geographic position, the Trans-Siberian Railway and the Turkestan-Siberian Railway (created in 1929 and connecting the city to Central Asia), Novonikolayevsk developed well until the Russian Civil War, started in 1917.

During the war, the Ob River Bridge was destroyed and the number of inhabitants declined. In 1921, with the beginning of Lenin's New Economic Policy introducing a relative liberalization and state capitalism, the city started rebuilding. It changed name for Novosibirsk in 1926.

During Stalin's industrialization period, Novosibirsk became one of the biggest industrial towns of Russia, in fields like heavy mining or metallurgy. It also got a new power station. Because of this fast growth, Novosibirsk was nicknamed the "Chicago of Siberia". During World War 2, around 50 plants relocated from western Russia to Novosibirsk to protect them from the war. This fast increase of factories generated a lack of energy. In reply to this, a new hydroelectric power station was created in 1957, with a big water reservoir named the Ob Sea. This is also during this period that the Soviet Government decided to design a center for scientific research. Akademgorodok was born and it hosts around 35 research institutes and universities, including Novosibirsk State University where I did my internship.

1.2 Novosibirsk nowadays

Nowadays, Novosibirsk is still an important city in Russia. It has the third number of inhabitants in Russia, with more than 1.5 million of people living here. Contrary to other cities located farther north in Siberia, Novosibirsk benefits from favorable temperatures, with still an average of -19° in January.

The city is a hub of activities in Asian part of Russia, in terms of trade and business for example, or also in scientific and cultural fields. It is also a center for public policy with some governmental offices. We can find eight administrative districts.

It is interesting to see that more than 80 thousand foreigners come to Novosibirsk for business and tourism. It develops links with other countries. In this way, you can find here Consulates General of Germany, Uzbekistan and Ukraine for example. It is also linked to China, Japan, and South Korea in particular, about economic and cultural points. Several organizations are present in the city and the University too.

Novosibirsk benefits from a good development despite absence of resource-extraction companies, contrary to most of large towns in Siberia. Over half of people here work in medium and large companies, the rest works in the numerous small enterprises.

1.3 Novosibirsk State University

Novosibirsk represents the most important academic and science applied center in the Asian part of Russia. You can find around 1.5 thousand doctors of science and 3.5 thousand PhD's, and the largest science center in the whole of Siberia.

As explained before, Akademgorodok is home of the educational and scientific center of Siberia. It is located 20 km south of the city center. Around this part of the Sovetsky District, there is a birch and pine forest. It looks like the campus of Orsay, but with larger buildings, roads and forest. There are institutes, dormitories for students, medical academy, flats and houses, stores, hotels, hospitals, stadium, restaurants etc. It is more a town than a campus.

The Novosibirsk State University (NSU) counts between 6 000 and 7 000 students, and between 2 000 and 2 500 academic staffs. It is a public and non-profit organization. We can divide research in this university in 10 main divisions, including the Information Technologies (IT) division. Three branches compose it: the Intel High-performance Computing Laboratory, the Laboratory of Informatics Problems and the Modern Computer Technologies. I worked in the last of them, with at the head Sergey Fedorovich Krendelev, who is my supervisor for this internship.

Figure 2 - View of the new building (source: Prologue Educational Consultants)

The university gives opportunities to foreigners to work on subjects they can find on the website of NSU. Thanks to Michèle Debrenne, who organized an information meeting in Polytech Paris Sud, I found my internship in cryptography by this way.

I had the chance to work in a new huge building (*Figure 2*). First, I was alone in a room with my friend Alexandre TISSIER from Polytech Paris Sud, who came with me in Russia. Then other French students came in this room for their internships.

1.4 Sergey F. Krendelev

My supervisor Sergey Krendelev graduated from the NSU in 1973, getting a Ph.D. about Complex Variables. He worked at the Sobolev Institute of Mathematics from 1982 to 1996, and he became professor at NSU in 1998. He gives lectures on mathematics and works on cryptography, and use to work in the USA and in Poland.

He published scientific papers about securing data. He obtained 5 years ago a Mega Grant from the Russian Ministry of Education on Secure Cloud. It allows running a research project in Russia over several years. It can concern all scientific fields. From 2010 to 2016, only 200 projects were funded, among which the project of my supervisor.

Figure 3 - Mr. Sergey Krendelev, my supervisor (source: JetBrains website)

As already explained, my supervisor is at the head of the Modern Computer Technologies. The aim of this laboratory is to provide internships opportunities for students from NSU. The proposals are mainly about research-oriented projects. The laboratory is for example equipped with database servers. It is not limited to system and network programming, and it is possible to find proposals in developing, for instance, business support applications. Students can also take part in challenges. In 2013, some of them had to design a programming platform supplying cloud services to companies.

He is also at the head of the Cryptographic Lab. The members try to find new systems to secure data. They work on new key exchange methods taking into account quantum computers, on protection of databases, symmetric encryption and more. I worked for this laboratory for my internship.

2 Mission and goals

2.1 The key exchange problem

Key exchange is any method in cryptography by which cryptographic keys are exchanged between users, allowing use of a cryptographic system. The information they require to do so depends on the encryption technique they might use. If they use a code, both will require a copy of the same codebook. If they use a symmetric key cipher (*Figure 4*), both will need a copy of the same key.

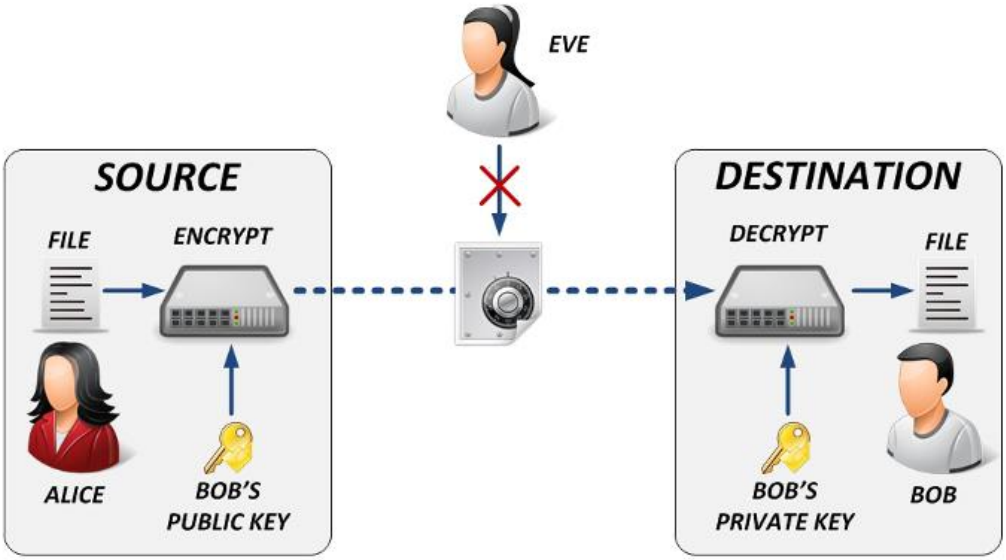

Figure 4 - Secured key exchange prevents from attacks (source: richardgoyette.com)

This is not a simple problem, especially when the two users involved have never met and know nothing about each other. Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical channel, such as paper key lists transported by a trusted courier, diplomatic bags …

The Diffie–Hellman key exchange (see below) method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key even over an unsecured channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

But the original Diffie-Hellman implementation or the one based on elliptic curves present the disadvantage to be possibly broken, even if it requires a huge amount of computational resources. In fact, when using large numbers, this action is likely very difficult for modern supercomputers to do in a reasonable amount of time. However, constantly

increasing length of the keys make the cypher greedier in resources and, more important, the emergence of quantum computers threatens the security of such crypto-systems.

2.2 Tasks

During this internship, I had to work with another student from Polytech (Polytech Marseille) named Lucas TAVEAU. He studied in NSU cryptography for one year, and he had to finish this year by making an internship. Our main goal was to try providing a more secured method for key exchange, based on the Diffie-Hellman method.

First, I had to design a C++ class allowing some matrices operations. During a practical work, we already started to use a similar class, in which values of the matrix are stored in an array with one dimension. It was supposed to increase performance. I overloaded the usual operators for matrices and I added methods. Then, I was supposed to use this class to apply a simple version of the final protocol, based on Diffie-Hellman key exchange (see below).

Then, thanks to Lucas's work, we were able to introduce Grassmann algebra (part 3.2 of this report) and we had to apply the whole protocol given by our supervisor, without interface. Then I had to implement this interface in C++ and Lucas had to design it for smartphones.

Lastly, with Lucas and the help of Russian students, we had to write a scientific article presenting our protocol (part 3.3 of this report). You can find this report in Annexes. This article is not finished yet and it has some common parts with this report, but I thought it was interesting to see how we organized it.

3 Common work

3.1 Original Diffie-Hellman key exchange

3.1.1 Protocol description

During this internship, I had to work on a method based on the Diffie-Hellman protocol, proposed by the two searchers who gave their names to the method. Let's suppose that Alice and Bob want to establish a conversation: they have to use a key K, which will allow them to, subsequently, decipher an encoded message. There is how the protocol works:

Steps	Alice	Bob
1	Both of the users choose a prime number p and also an integer a, which statisfies $1 \leq a \leq p-1$. These parameters are not private.	
2	Alice chooses a secret integer x_1.	Bob chooses a secret integer x_2.
3	Alice calculates $y_1 = a^{x1}(mod\ p)$ and gives it to Bob.	Bob calculates $y_2 = a^{x2}(mod\ p)$ and gives it to Alice.
4	Alice calculates $y_2^{x_1} = (a^{x2})^{x_1}\ (mod\ p)$	Bob calculates $y_1^{x_2} = (a^{x1})^{x_2}\ (mod\ p)$
5	Alice and Bob get the same value of the private key $K = a^{x_1 x_2} = a^{x_2 x_1}$.	

We can also represent this method with the diagram *Figure 5* (see below).

At first glance, it seems that it could be easy to find the key K just from the values of y_1, y_2, a and p, because Alice and Bob just apply very simple operations to find this key. However, it is necessary to solve the discrete logarithm problem. It is about finding an integer x, between 0 and $n-1$ (with n the order of a), which satisfies:

$$y = a^x\ (mod\ p)$$

p being a prime number, y and a, two known integers. Nowadays, this problem is still not solved. That means nobody found an algorithm able to find x in a reasonable amount of time, for high values of y and a. In fact, we need an algorithm with a complexity of $O(\log\ n)$, but the best algorithm we currently have compute in $O\left(n^{\frac{1}{2}}\right)$ opérations.

This kind of protocol, based on the use of an operation which is hard to reverse, was a revolution in the cryptography's world, because in our example, Bob and Alice never had to exchange directly the private keys.

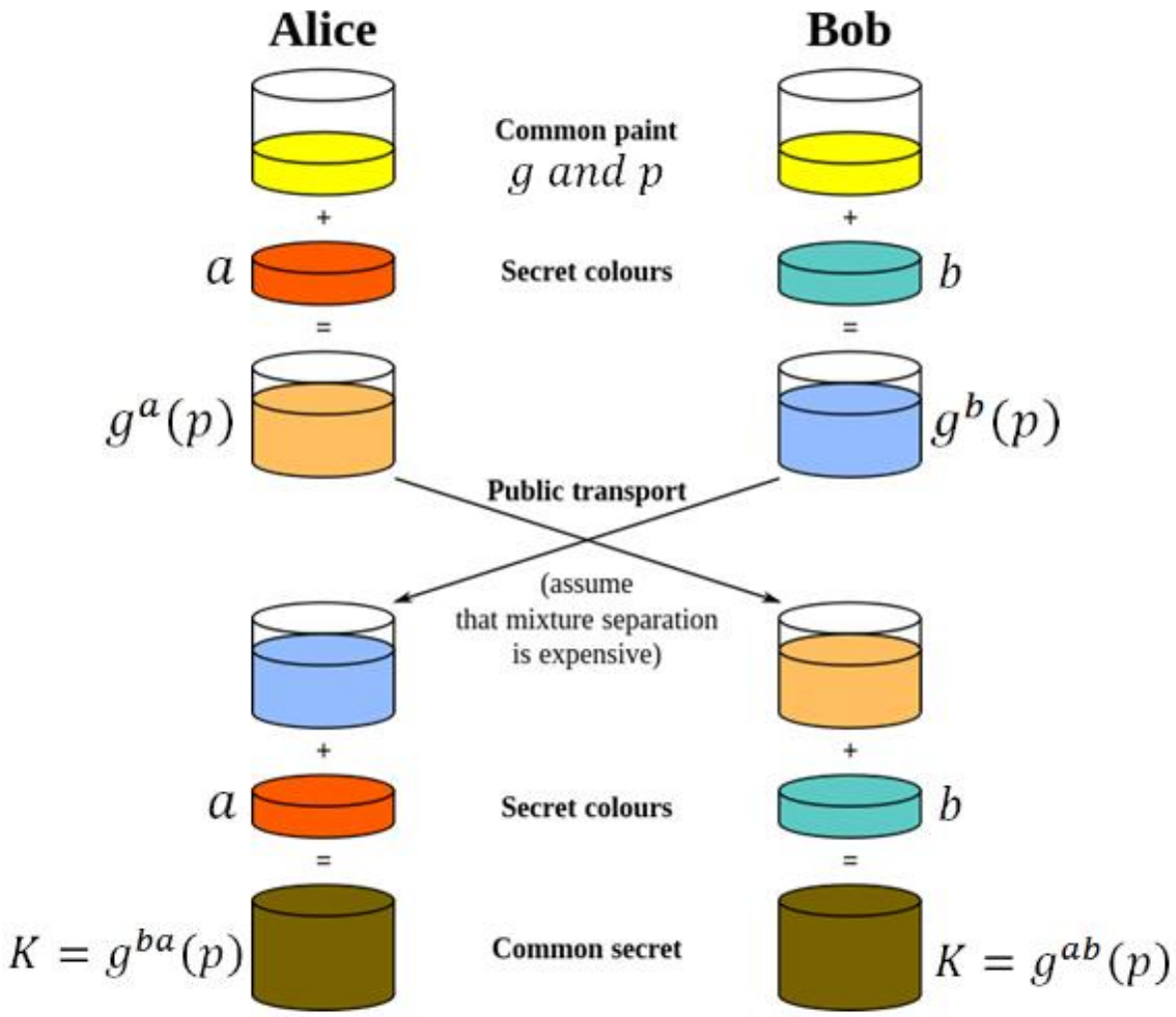

Figure 5 - Illustration of the original Diffie-Hellman key exchange (source: wikipedia)

3.1.2 Issues

However, this protocol has a major defect, like the other systems based on this way of thinking. It is impossible to use it to sign digitally a message, in opposition to RSA. As we can't identify the sender, this protocol is highly exposed to Man-In-The-Middle attack.

Let's take the previous example with Alice and Bob. They agree on common a and p. Alice chooses secretly an integer x_1 and Bob do the same with x_2. We will add to this system another malicious user called Oscar, who wants to intercept messages between Alice and Bob.

Alice calculates and sends y_1, but Oscar will intercept it. He chooses an integer x' and sends $y' = a^{x'} \ (mod\ p)$ to Bob (because he knows a and p since they are public). He executes the same step for the message from Bob to Alice.

From this, Alice calculates her key $K_1 = y'^{(x_1)} \ (mod\ p)$ and Bob $K_2 = y'^{(x_2)} \ (mod\ p)$. These two keys differ, but it is necessary to understand that Alice will not communicate with Bob but with Oscar. Indeed, Oscar has the following values: y_1, y_2, a, p, y' and x'. He can easily find K_1:

$$K_1 = y'^{(x_1)}(mod\ p) = \left(a^{x'}\right)^{x_1}(mod\ p) = (a^{x_1})^{x'}\ (mod\ p) = y_1^{x'}(mod\ p)$$

He also knows $K_2 = y_2^{x'}(mod\ p)$, that allows him to to transmit and recover Bob's data. Thus, Oscar is not only able to catch what the users send, but also to modify these messages.

Moreover, both Alice and Bob must simultaneously execute the steps indicated. Indeed, if Alice wants to establish communication with Bob, she needs the result of the first Bob's calculation to define K (and Bob needs the Alice one too). That's why this protocol was abandoned in favor of protocols using public keys in a lot of fields. Nevertheless, it is still used by Bluetooth technology to establish a connection.

3.2 Pre-requisite

3.2.1 Choosing commutative matrices

Our version of the Diffie-Hellman protocol requires the exchange of two hidden and commutative matrices, one matrix for each user. As they are commutative, they will be able to recover the same key.

The question we are asking now is: how to create efficiently two commutative matrices? The first solution we can think about is to choose diagonal matrices. We can for example select these two 2×2 matrices:

$A = \begin{pmatrix} a_1 & 0 \\ 0 & a_2 \end{pmatrix}$ and $B = \begin{pmatrix} b_1 & 0 \\ 0 & b_2 \end{pmatrix}$

It's easy to see that $AB = \begin{pmatrix} a_1 b_1 & 0 \\ 0 & a_2 b_2 \end{pmatrix} = BA$.

You will get this feature with diagonal matrices of any size, but we use only a small part of these. Here is how to proceed: we start by choosing a common matrix C, which has the same size as the two matrices we are looking for. Then we have to choose two different polynomials. For example, the first user selects the polynomial $f(C) = 3C^2 + 2C + 2I$ and the second one $g(C) = C^2 + 2C + 4I$. These polynomials commute and you can find the proof in the article in Annexes.

3.2.2 Grassmann algebra

The protocol I will describe later is not only based on commutative matrices but also on Exterior algebra or Grassmann algebra. This algebra allows changing basis of matrices in another way. This operation is hard to reverse, and it replaces the discrete logarithm problem from the original Diffie-Hellman protocol. You can find more information about how to change basis in the scientific article in Annexes. Security will be discussed later in this report.

Here is an example of a change of basis for a 3×3 matrix S:

Let $S = \begin{pmatrix} 5 & -2 & 0 \\ 1 & 3 & 2 \\ 4 & 1 & 7 \end{pmatrix}$

The change of basis gives S':

$$S' = \begin{pmatrix} 17 & 10 & -4 \\ 13 & 35 & -14 \\ -11 & -1 & 19 \end{pmatrix}$$

3.3 Our protocol

For our Diffie-Hellman-based protocol, we need to define these public elements:

- 2 integers $n, m \geq 3$.
- Q : $n \times n$ matrix.
- R : $m \times m$ matrix.
- 2 integers k_1 and k_2, $2 \leq k_1 < n$ and $2 \leq k_2 < m$.
- W: $p \times q$ matrix, $p = \binom{n}{k_1}, q = \binom{m}{k_2}$.

Degree of polynomials used by both the users in this protocol is $(\text{size of matrix}) - 1$, so the matrices conserve the change of basis. Alice generates two polynomials $f_A(X)$ and $g_A(X)$, and then calculates matrices $f_A(Q)$ and $g_A(R)$.

She changes the basis of these matrices and gets new polynomials $\overline{f_A(Q)}$ and $\overline{g_A(R)}$. We saw that, by doing a change of basis in Grassmann Algebra, the two matrices computed by Alice can have a different size from matrices $f_A(Q)$ and $g_A(R)$.

Then, Alice chooses two other polynomials $u_A(X)$ and $v_A(X)$ and calculates $u_A\left(\overline{f_A(Q)}\right) = U_A$ and $v_A\left(\overline{g_A(R)}\right) = V_A$. Finally, she computes $W_A = U_A W V_A$ and sends it to Bob. As you can see, W has to be a $p \times q$ matrix (p and q being sizes of matrices calculated) to allow this multiplication.

Bob does the same and gets $f_B(Q)$ and $g_B(R)$, $\overline{f_B(Q)}$ and $\overline{g_B(R)}$, then U_B and V_B, and he finally sends computes $W_B = U_B W V_B$ to Alice.

After these steps, Alice calculates K_A:

$$K_A = U_A W_B V_A = u_A\left(\overline{f_A(Q)}\right) \times u_B\left(\overline{f_B(Q)}\right) \times W \times v_B\left(\overline{g_B(R)}\right) \times v_A\left(\overline{g_A(R)}\right)$$

Then, Bob computes K_B:

$$K_B = U_B W_A V_B = u_B\left(\overline{f_B(Q)}\right) \times u_A\left(\overline{f_A(Q)}\right) \times W \times v_A\left(\overline{g_A(R)}\right) \times v_B\left(\overline{g_B(R)}\right)$$

As shown in the scientific article, u_A and u_B commute if the two matrices as variables commute. Here, $f_A(Q)$ and $f_B(Q)$ commute, so $\overline{f_A(Q)}$ and $\overline{f_B(Q)}$ too, because change of basis keeps the commutativity. We also can apply this to polynomials v_A and v_B. Finally, we see that Alice and Bob get the same private key $K = K_A = K_B$.

The whole protocol is illustrated by the following diagram *Figure 6*.

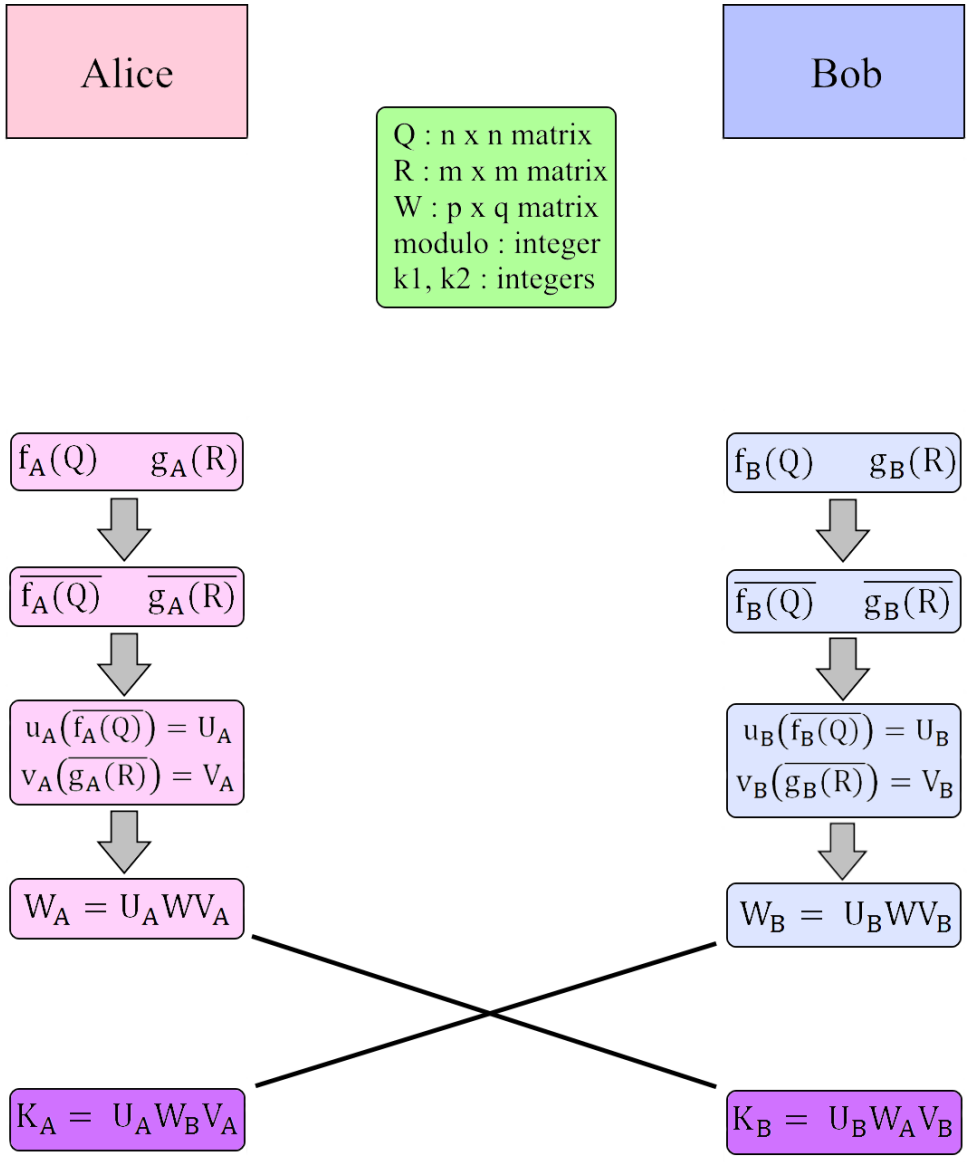

Figure 6: Illustration of our protocol

3.4 Security analysis

3.4.1 Qualitative security Analysis

There are two ways of breaking such cryptographic systems.

This first one is the brute-force attack which corresponds to an exhaustive key search, that is to say trying to find all possible solutions and for each, the verify their validity. The complexity of brute-force attacks on cryptographic keys in n bits is, for an ideal realization, equal to 2^n, that is to say an exponential complexity. However, in general rules, the random number generator lacks of entropy (we currently don't know how to generate a really random set of number), and consequently the key-space is generally much smaller than n bits-space (2^n). As a consequence, some implementations of modern cryptographic algorithms have been cracked by brute-force because the actual key-space was smaller than the one in the algorithm. So a particular attention has to be paid for choosing a relevant source of entropy for random number generation functions (some systems ask the user to type random keys on their keyboard with closed eyes to generate entropy), but I won't deal with this particular issue in this report.

The second one is to resolve the system of equations derived from the internal structure of the cryptographic scheme. Then the attacker could use the Gröbner basis conversion to find satisficing solutions. However, such sets of simultaneous equations generated from the mathematical structure of a scheme are quite big and solving non-linear systems is a problem that has been proven to be NP-hard. As a consequence, even if the complexity is less than the one of a complete exhaustive search, the complexity is still exponential. Because of that, this attack is not likely to be produced with success in the real word, and in this sense, not considered only as a practical flaw that could be exploited, but as a "certificational weakness".

Moreover, we can easily increase exponentially the maximal degree of the equations by changing the basis in Grassmann Algebra recursively. And doing that way, the generation time will be augmented only linearly.

3.4.2 Quantitative security analysis

Using two polynomials, on the left and on the right of W, allows us to choose W as an invertible matrix. With only one matrix, it would be easy to find the polynomial selected by multiplying by W^{-1}, inverse of W.

Using Grassmann algebra prevents from a possible attack coming from a malicious person. In fact, thanks to the change of basis, the system of equations obtained is non-linear. Let see what happen if we try to solve this system. Just as before, polynomials $\mathrm{f_A(Q)}$, $\mathrm{g_A(R)}$, $\mathrm{U_A}$ and $\mathrm{V_A}$ introduce n, m, p and q variables respectively. If we choose $n = m = 6$ and $k_1 = k_2 = 4$ for the change of basis part, sizes of $\overline{\mathrm{f_A(Q)}}$ and $\overline{g_A(R)}$ will be $\binom{6}{4} = 15 = p =$

q. The size of the system to solve will be the size of W, i.e. $q \times p = 15^2 = 225$ equations, and the number of unknowns is 42. This kind of system is a very difficult problem, and we can choose to use matrices much bigger, introducing more equations and unknowns. If the attacker can't figure out this problem, he is no longer able to search the two matrices U_A and V_A multiplying the public matrix W, since these matrices are not direct power of public matrices Q and R. He can't find U_B and V_B computed by Bob too, so he can't calculate the private key.

Finally, the method is vulnerable to Man-In-The-Middle attack (MITM), like the original Diffie-Hellman protocol. In this case, the attacker needs to intercept the matrix sent by Alice and the one sent by Bob. For both the users, he must select polynomials and build two matrices, as Alice and Bob. It allows the attacker to understand what the users sent, and also to possibly modify the messages. We can see it as two protocols working, the first one between Alice and the attacker and the other one between the attacker and Bob. To protect the system from this kind of threat, we can use certificates or, if it is possible, ask the users to apply a first time the key exchange, check by another way if the part of the key sent by Alice is the same matrix than the one receipted by Bob and exchange a true private key a second time.

3.5 Block cypher over Grassmann Algebra

We saw previously in this paper how to transmit keys in an unsecured channel, but we also can use this Diffie-Hellman method to directly exchange message, without the need to exchange keys.

In this case, Alice, who wants to send a message to Bob, must choose two random polynomials and apply this to the two public matrices Q and R. Then she applies Grassmann change of basis and selects two other polynomials to apply on these results, exactly in the same way than already explained. She gets the two matrices U_A and V_A. Then she selects the matrix M, containing message, and sends $U_A \times M \times V_A$.

Bob, who also knows public matrices Q and R, can produce in the same manner two matrices U_B and V_B. He sends to Alice $U_B \times U_A \times M \times V_A \times V_B$. Then, she inverts her two matrices, and computes:

$$\begin{aligned} & U_A^{-1} \times U_B \times U_A \times M \times V_A \times V_B \times V_A^{-1} \\ = & U_B \times U_A^{-1} \times U_A \times M \times V_A \times V_A^{-1} \times V_B \\ = & U_B \times M \times V_B \end{aligned}$$

Alice sends this result. Now, Bob only needs to compute inverse matrices U_B^{-1} and V_B^{-1}. He finds the message by calculating $U_B^{-1} \times U_B \times M \times V_B \times V_B^{-1} = M$. If somebody is sniffing the network, he only knows matrices Q and R, and he won't be able to find the message.

We still have a problem for this kind of protocol: we need invertible matrices. Most of random matrices are invertible, but we need to be sure to create invertible ones. This problem is hard to solve and we need to study this problem more. It could be a main subject for a next internship with my supervisor.

4 My Interface

As I mentioned at the beginning, I had to implement an interface to allow using easily the protocol. In this way, I used Qt. I describe how this framework was helpful in the personal report.

Before starting implementation, I had all the necessary tools: the *Matrix* class and the files *grassmann.cpp* and *grassmann.hpp*. It was the first time I used the framework, so I first practiced myself thanks to tutorials available on OpenClassroom.

4.1 Presentation of the interface

Figure 7 - screenshot of the interface for the key exchange protocol

As you can see in the *Figure 7*, we decided to create the same interface for the two users. It is then possible to produce a key with one window. But in the normal case, the first user runs the program on his computer, like the second user who has to choose the second part named *Bob*. The first user, who chose the *Alice* part, has to select parameters of the exchange, namely a security level and the size for the key. Let's call the two users Alice and Bob as before.

Alice can choose between 5 levels of security, and the number of this level corresponds to the number of times we will apply the change of basis during the whole protocol. She also has the opportunity to select a size for the key, from 64 bits to 512 bits. Then she clicks on “Start” button to begin computing. After that, the program creates a new file, containing matrices and information for the second user Bob. Alice has to send this file to Bob. The aim of our protocol is to send information in a unsecured network, so she can for example send the file by e-mail.

After selecting the second part of the window, Bob must select the file Alice just sent. He clicks on the “Start” button of the second part of the window and wait. The key will be displayed on his text area, and Bob can copy this key to use it in another application. Alice still has not the key, so the program has also created a file containing Bob's information. He sends this to Alice and she just need to select the file and click again on “Start” to get the same private key. Then, the two users can quit the program by clicking on “Quit” or File➜ Exit.

As you can see, Alice gets the key after two actions. Before the end of the first one, the lower part of her box (the Alice part of the window) is disabled, and after, the upper part is disabled. It shows to the user that the program is waiting something.

The session is identified by a new Universal Unique Identifier (UUID). The files created by Alice and Bob are named like this: “UUID.alice” and “UUID.bob”. During the last step, the program makes sure that the UUID sent by Bob is the good one. If the UUID is different, a warning window appears and the file is not selected.

4.2 New classes

We decided with Lucas to create two classes *SessionA* and *SessionB*. In this model, Alice is represented by an instance of the *SessionA* class. This class initiates the protocol and chooses parameters. Bob is represented by an instance of the *SessionB* class. Separating the protocol into two “sessions” allows always running the code directly on the client computer, and never on a server, which could be vulnerable to attacks.

Both *SessionA* and *SessionB* has two main methods *importFile()* and *exportFile()*. When Alice clicks on “Start” for the first time, it calls her method *exportFile()*, which creates random polynomials, changes basis, multiplies matrices and finally creates the new file. Bob

calls *importFile()* and then *exportFile()* by clicking on “Start”. The first method collects data stored in the Alice's file. The second do the same thing as *exportFile()* of *SessionA* and calculates the private key. Lastly, *exportFile()* from *SessionA* is called: it takes information on the Bob's file and calculates the private key. This is illustrated in the *Figure 8*.

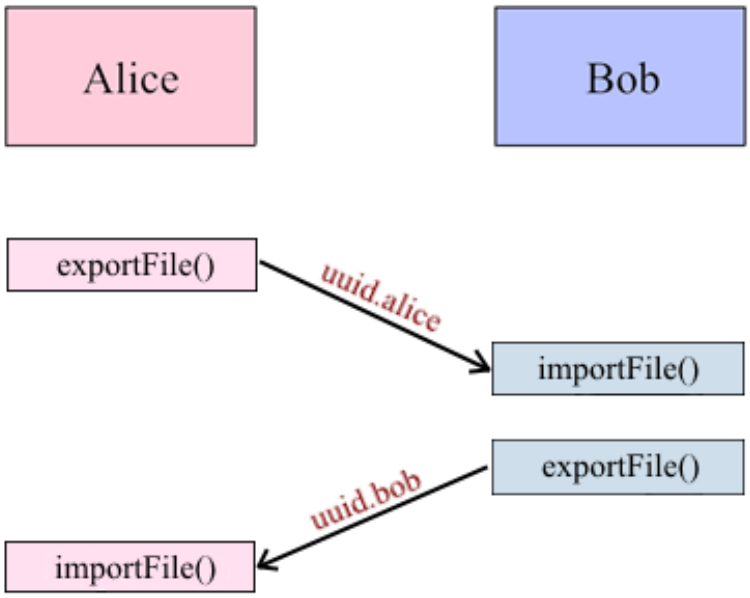

Figure 8 - How a session is organized

I also needed a class to implements the “View” part of the program. I'm using the class *MainWindow* to do this. It implements the *QmainWindow* class, which provides a main application window. In this class, you can find all the layouts and widgets the window contains, and also two instances of *SessionA* and *SessionB*, representing Alice and Bob. It was possible to add a menu bar to my class, since it implements *QmainWindow*. So I decided to also give the opportunity to quit the program and display help.

After testing the code with big matrices and large modulo, the program displays two different public keys. It was due to integer overflow. I had to create a new method to multiply matrices using modulo, and Lucas also had to modify the Grassmann file. After this, I had another problem: for big matrices, it can take more than ten seconds to compute change of basis, so the interface wasn't responsive. I had to use another class named *ComputationThread* which implements the class *QThread* to solve it. As you can see in the code, *MainWindow* doesn't directly call methods imp*portFile()* and *exportFile()* from *SessionA* and *SessionB*, but it calls class method of the attribute *cThread*.

4.3 Possible improvements

Although I lastly succeeded in solving the last problem, the use of thread is not perfect. Indeed, when one of the users clicks on “Stop” button, the thread won't stop computing. If the user tries to begin another key exchange, the program could write and read simultaneously false data. Moreover, the changes of basis can take a long time, so it could be a good idea to indicate how long it will take to finish computation with a progress bar. It could provide more feedback to the user.

The aim was also to use web services to improve user-friendliness. I am aware that it could be not convenient to send the file, but I preferred to supply an effective interface and to write correctly the scientific paper.

Conclusion

Like so many students in computer science, security and cryptography is a field that interests me a lot. Cryptography requires good listening of mathematics, and I wasn't sure to fit with what my supervisor asked. Thanks to Mr. Krendelev and my teammate Lucas, I know more about key exchange and I was able to help them progress. We discovered that it is possible to adapt our key exchange method to encryption and decryption. This introduces a new issue: how to produce perfectly an invertible matrix? This problem could be a subject for a next internship.

It was the perfect opportunity for me to discover research world. I worked in autonomy and, in most situations I had to find out a solution by myself. Furthermore we had to write a scientific article, and it provided us a real goal at the end of our internship. It did improve my research skills and show me what constitutes research work.

In addition, it allowed me to consolidate my learning of C++, a language I discovered this last semester. I applied new technologies about this language. First, I learned how to use the framework Qt to design graphic interfaces. As we had to compute an algorithm for a long time, I saw that I also had to use threads to keep the interface responsive. Although the interface I developed is simple, I learned how to combine the different widgets, and it showed me some problems we can encounter.

This internship was also the opportunity for me to discover a new country and culture. It gave me the chance to improve my English and to apply such knowledge in practice.

Bibliography

EFTEKHARI Mohammad, *A Diffie-Hellman key exchange protocol using matrices over non commutative rings*, Amiens, (2012), p 1-6, consulted June 30 at the web address: https://hal.inria.fr/file/index/docid/735422/filename/keyexchange.pdf

W. SHOR Peter, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Murray Hill, (1994), p 124-134, consulted June 29 at the web address: https://pdfs.semanticscholar.org/2273/d9829cdf7fc9d3be3cbecb961c7a6e4a34ea.pdf

LEWAND Robert, *Cryptological Mathematica*, The Mathematical Association of America, Baltimore, (2000), p 124-140.

PALMGREN Keith, ISSA Journal, *Diffie-Hellman Key Exchange: A Non-mathematician's explanation*, October 2006, (2006), p 30-33, consulted June 30 at the web address: http://academic.regis.edu/cias/ia/palmgren_-_diffie-hellman_key_exchange.pdf

KAK Avi, *Lecture 13: Certificates, Digital Signatures, and the Diffie-Hellman Key Exchange Algorithm*, Purdue, (2017), p 3-61, consulted July 18 at the web address: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture13.pdf

ISRAEL Tristan, *Maîtriser Qt 5, Guide de développement d'applications professionnelles*, Editions ENI, Paris, (2014).

SIVANAGASWATHI Kallam, *Diffie-Hellman: Key Exchange and Public Key Cryptosystems*, Terre Haute, (2015), p 17-18, consulted July 26, 2017 at the web address: http://cs.indstate.edu/~skallam/doc.pdf

CHEFRANOV A. and MAHMOUD A., *Lecture Notes in Electrical Engineering*, Springer, Germany, (2008), p 317-324.

PURBHOO K., *Notes on Tensor Products and the Exterior Algebra*, Waterloo, (2012), p 12-17, consulted July 18, 2017 at the web address: https://www.math.uwaterloo.ca/~kpurbhoo/spring2012-math245/tensor.pdf

ANNEX I: Scientific article

Please note that this paper is a work in progress. A different final version will be written.

Cryptography over Grassmann Algebra
Defining a new secure key exchange scheme

Abstract

In this article we review an algorithm for a key exchange system based on Grassmann algebra. Particularly, it will deal with how to change the exterior power basis in which a matrix is represented. For this last point, we will rely on the document "Modules and Key exchange", findable in the bibliography. Then, we will present a demonstration desktop and mobile application, implementing the whole key-exchange mechanism between two devices. Finally, we will quickly present how a cypher could be designed using the same primitive as the one of our key exchange.

ANNEX I: Scientific article

Summary

1 Key exchange scheme over Grassmann Algebra 3
1.1 The key exchange problem 3
1.2 The security of existing key exchange 3
1.3 Which algorithms for post-quantum cryptography? 5
2 Our protocol 5
2.1 Polynomials and commutativity 5
2.2 Introduction to Grassmann algebra and basis change 6
2.3 Structure of the protocol 8
3 Security analysis 9
3.1 Qualitative security Analysis 9
3.2 Quantitative security analysis 10
4 Our implementation 11
5 Block cypher over Grassmann Algebra 13
6 Bibliography 14

ANNEX I: Scientific article

1 Key exchange scheme over Grassmann Algebra

1.1 The key exchange problem

Key exchange is any method in cryptography by which cryptographic keys are exchanged between users, allowing use of a cryptographic system. The information they require to do so depends on the encryption technique they might use. If they use a code, both will require a copy of the same codebook. If they use a symmetric key cipher, both will need a copy of the same key.

This is not a simple problem, especially when the two users involved have never met and know nothing about each other. Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical channel, such as paper key lists transported by a trusted courier, diplomatic bags …

The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key even over an insecure channel. This key can then be used to encrypt subsequent messsages using a symmetric key cipher, and then permitting the establishing of a secure communication channel.

But the original Diffie-Hellman implementation or the one based on elliptic curves present the disadvantage to be possibly broken, even if it requires a huge amount of computational resources. In fact, when using large numbers, this action is likely very difficult for modern supercomputers to do in a reasonable amount of time. However, constantly increasing length of the keys make the cypher greedier in resources and, more important, the emergence of quantum computers threatens the security of such crypto-systems.

1.2 The security of existing key exchange

First, let's focus on the deficiencies of the original and the elliptic curves implementations. Let's call Alice and Bob the two parties. They agree on an arbitrary starting color that does not need to be kept secret. Each of them selects a secret color -red and aqua respectively- that they keep secret.

The next part of the process is that Alice and Bob now mix their secret color together with their mutually shared color. Finally, each of the two mix together the color they received from the partner with their own private color. The result is a final color mixture, which is the same for the both parties.

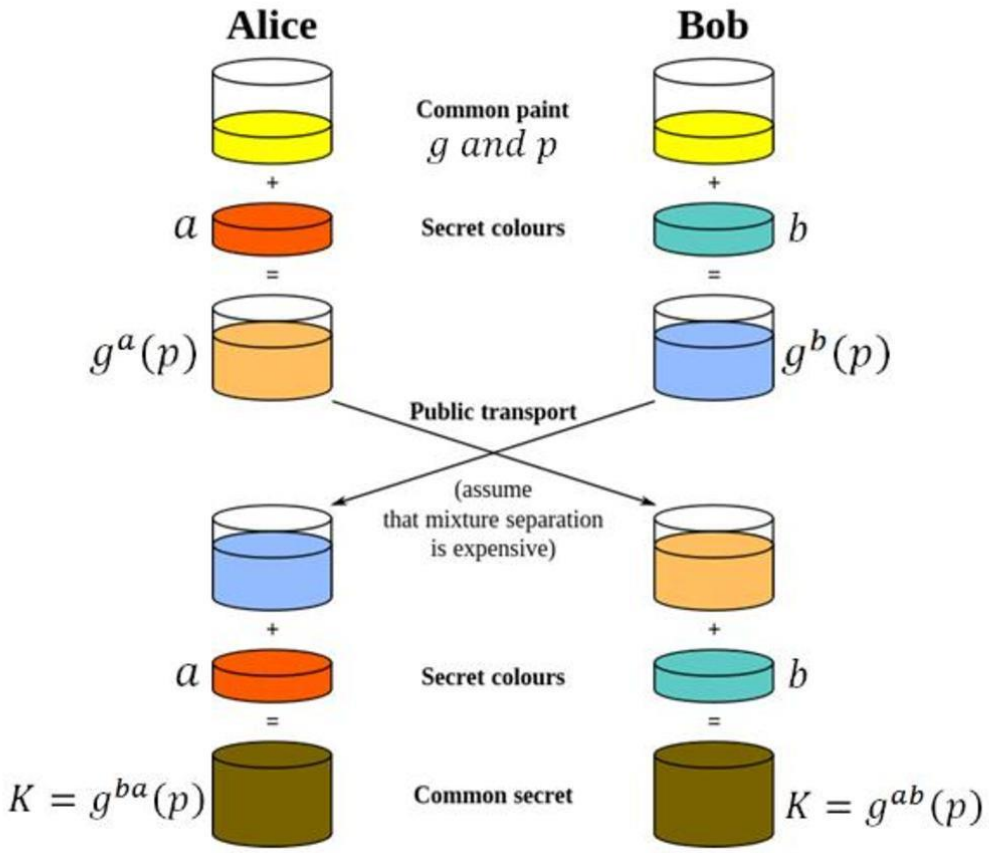

Figure 1 - Illustration of first Diffie-Hellman protocol

So the security of the original implementation relies on the problem of the discrete logarithm. The latter implies that no efficient general method for computing discrete logarithms (i.e. reverse an exponentiation) on conventional computers is known. The implementation using elliptic curves rely also exactly on this problem of the discrete logarithm.

So, the weak point of the Diffie-Hellman implementations used nowadays is that the operation represented by the *mixing color mechanism* in the above example has a corresponding reverse operation which is still nowadays considered as hard to reverse. But the problem of discrete logarithm could be solved by using Shor algorithm. This process, formulated by Peter Shor in 1994, demonstrated that quantum computers can overcome this issue, as well as integer factorization problem.

We can now consider using another operation but way more difficult to reverse, with the same Diffie-Hellman global scheme. That's basically the goal of the crypto-system we will describe in this article.

1.3 Which algorithms for post-quantum cryptography?

As we said, nowadays, popular public-key cryptographic algorithms- as well as their related key exchange schemes- mostly rely on the impossibility for any computation systems knows at this day to break two hard mathematical problems: the integer factorization problem and the discrete logarithm problem. But as big enough quantum computers could easily break such problems, and as the technology is beginning to emerge (even with low capabilities), cryptographs try to find new solutions, known as "quantum safe", that would stay reliable and secure even in a future with quantum computers.

One of the families of post-quantum cryptographic primitives is the multivariate cryptography. Secure encryption algorithms based on this approach, because of its very structure, are hard to design. However, it's very well suited for signature and key exchange schemes.

More precisely, such systems are based on multivariate polynomials over a finite field F. If the maximum degree of the polynomials is 2, we talk about MQ (Multivariate Quadratic) cryptography. The system that we describe in this present paper will have polynomials with degrees superiors to 2 and superior to 8 in the implementation.

2 Our protocol

2.1 Polynomials and commutativity

2 polynomials commute

Let A be a square n × n matrix. We choose $f(A) = a_0 I_n + a_1 A + a_2 A^2 + \cdots + a_{n-1} A^{n-1}$ and $g(A) = b_0 I_n + b_1 A + b_2 A^2 + \cdots + b_{n-1} A^{n-1}$.

$$\begin{aligned} f(A)g(A) &= (a_0 I_n + a_1 A + a_2 A^2 + \cdots + a_{n-1} A^{n-1})(b_0 I_n + b_1 A + \cdots + b_{n-1} A^{n-1}) \\ &= a_0 I_n \times g(A) + a_1 A \times g(A) + \cdots + a_{n-1} A^{n-1} \times g(A) \end{aligned}$$

$$\begin{aligned} g(A)f(A) &= (b_0 I_n + b_1 A + b_2 A^2 + \cdots + b_{n-1} A^{n-1})(a_0 I_n + a_1 A + \cdots + a_{n-1} A^{n-1}) \\ &= b_0 I_n \times (a_0 I_n + a_1 A + a_2 A^2 + \cdots + a_{n-1} A^{n-1}) + b_1 A \times (a_0 I_n + a_1 A + a_2 A^2 + \cdots + \\ &a_{n-1} A^{n-1}) + \cdots + b_{n-1} A^{n-1} \times (a_0 I_n + a_1 A + a_2 A^2 + \cdots + a_{n-1} A^{n-1}) \\ &= a_0 I_n \times (b_0 I_n + b_1 A + b_2 A^2 + \cdots + b_{n-1} A^{n-1}) + a_1 A \\ &\quad \times (b_0 I_n + b_1 A + b_2 A^2 + \cdots + b_{n-1} A^{n-1}) + \cdots + a_{n-1} A^{n-1} \\ &\quad \times (b_0 I_n + b_1 A + b_2 A^2 + \cdots + b_{n-1} A^{n-1}) \end{aligned}$$

(A)

because A commute with I (and obviously A commute with A) and are square matrices with the same size n.

$$g(A)f(A) = a_0 I_n \times g(A) + a_1 A \times g(A) + \cdots + a_{n-1} A^{n-1} \times g(A) = f(A)g(A)$$

We can see that two matrices calculated as polynomials of matrix A commute. In this demonstration, we can see in step (A) that we only need to have a matrix which commutes with itself. If we choose another matrix B for the second polynomial, we need B to be square and to commute with A. So, we can more generally state this:

Let two square matrices A *and* B *with the same size. Let two polynomials* f(X) *and* g(X). A and B commute $\Rightarrow$ f(A)and g(B)commute.

2.2 Introduction to Grassmann algebra and basis change

Exterior algebra or Grassmann algebra of a module over a commutative ring, algebra, is an associative algebra that contains the module and such that the square of any element of the module is zero.

Let $\mathbf{Z}^n$ be a module over ring **Z**. Define a standard basis in $\mathbf{Z}^n$: $e_1, e_2, \dots, e_n$. We can define Grassmann algebra with Grassmann multiplication ∧. It's graded algebra every component denoted by $\Lambda^k(\mathbf{Z})$, basis of this component is

$$\{e_{i_1} \wedge e_{i_2} \wedge \dots \wedge e_{i_k} \mid 1 \le i_1 < i_2 < \cdots < i_k \le n\}$$

The dimension of this module is $dim\, \Lambda^k(\mathbf{Z}) = \binom{n}{k}$.

Suppose that there are relations:

$e_i \wedge e_i = 0$ and $e_i \wedge e_j = -e_j \wedge e_i \ (i \neq j), i, j \in [1, n]$ (B)

Let S be a matrix of linear mapping from $\mathbf{Z}^n$ to $\mathbf{Z}^n$. Then this matrix can be lifted to the mapping from $\Lambda^k(\mathbf{Z})$ to $\Lambda^k(\mathbf{Z})$ for every suitable k, i.e. $2 \le k \le n-1$.

Let $n = 3, k = 2$. Basis will be $e_1 \wedge e_2, e_1 \wedge e_3, e_2 \wedge e_3$

and dimension $\Lambda^2(\mathbf{Z}) = \binom{3}{2} = 3$. Every element $\omega \in \Lambda^2(\mathbf{Z})$ can be performed as $\omega = \lambda_1 e_1 \wedge e_2 + \lambda_2 e_1 \wedge e_3 + \lambda_3 e_2 \wedge e_3, \lambda_1, \lambda_2, \lambda_3 \in \mathbf{Z}$.

Example with $k = n - 1$:

Let $S = \begin{pmatrix} 5 & -2 & 0 \\ 1 & 3 & 2 \\ 4 & 1 & 7 \end{pmatrix}$

$$Se_1 \wedge Se_2 = \begin{pmatrix}5\\1\\4\end{pmatrix} \wedge \begin{pmatrix}-2\\3\\1\end{pmatrix} = \left[5\begin{pmatrix}1\\0\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix} + 4\begin{pmatrix}0\\0\\1\end{pmatrix}\right] \wedge \left[-2\begin{pmatrix}1\\0\\0\end{pmatrix} + 3\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\0\\1\end{pmatrix}\right]$$
$$= (5e_1 + e_2 + 4e_3) \wedge (-2e_1 + 3e_2 + e_3)$$
$$= -10e_1 \wedge e_1 + 15e_1 \wedge e_2 + 5e_1 \wedge e_3 - 2e_2 \wedge e_1 + 3e_2 \wedge e_2 + e_2 \wedge e_3 - 8e_3 \wedge e_1 + 12e_3 \wedge e_2 + 4e_3 \wedge e_3$$

By using identities (B) we get:

$$Se_1 \wedge Se_2 = 17e_1 \wedge e_2 + 13e_1 \wedge e_3 - 11e_2 \wedge e_3$$

$Se_1 \wedge Se_3$ and $Se_2 \wedge \mathrm{Se}_3$ can be performed in the same manner. As result, we get the following matrix:

$$S' = \begin{pmatrix}17 & 10 & -4\\13 & 35 & -14\\-11 & -1 & 19\end{pmatrix}$$

Example with $n = 4$ and $k = 2$:

Let $T = \begin{pmatrix}2 & 1 & 2 & 1\\1 & 4 & 0 & 1\\3 & -1 & 2 & 1\\2 & 2 & 2 & 3\end{pmatrix}$

We should get T' a 6×6 matrix, since $\binom{4}{2} = 6$.

$$Te_1 \wedge Te_2 = \begin{pmatrix}2\\1\\3\\2\end{pmatrix} \wedge \begin{pmatrix}1\\4\\-1\\2\end{pmatrix} = (2e_1 + e_2 + 3e_3 + 2e_4) \wedge (1e_1 + 4e_2 - e_3 + 2e_4)$$
$$= 2e_1 \wedge e_1 + 8e_1 \wedge e_2 - 2e_1 \wedge e_3 + 4e_1 \wedge e_4 + e_2 \wedge e_1 + 4e_2 \wedge e_2 - e_2 \wedge e_3 + 2e_2 \wedge e_4 + 3e_3 \wedge e_1 + 12e_3 \wedge e_2 - 3e_3 \wedge e_3 + 6e_3 \wedge e_4 + 2e_4 \wedge e_1 + 8e_4 \wedge e_2 - 2e_4 \wedge e_3 + 4e_4 \wedge e_4$$

By using identities (B) we get:

$$Te_1 \wedge Te_2 = 7e_1 \wedge e_2 - 5e_1 \wedge e_3 + 2e_1 \wedge e_4 - 13e_2 \wedge e_3 - 6e_2 \wedge e_4 + 8e_3 \wedge e_4$$

We can compute $Te_1 \wedge Te_3, Te_1 \wedge e_4, Te_2 \wedge e_3, Te_2 \wedge e_4, Te_3 \wedge e_4$in the same manner. We finally obtain a 6×6 matrix:

$$T' = \begin{pmatrix}7 & -2 & 1 & -8 & -3 & 2\\-5 & -2 & -1 & 4 & 2 & 0\\2 & 0 & 4 & -2 & 1 & 4\\-13 & 2 & -2 & 8 & 5 & -2\\-6 & 2 & 1 & 8 & 10 & -2\\8 & 2 & 7 & -6 & -5 & 4\end{pmatrix}$$

2.3 Structure of the protocol

For our key-exchange protocol, we need to define these public elements:

- 2 integers $n, m \geq 3$.
- Q : $n \times n$ matrix.
- R : $m \times m$ matrix.
- 2 integers k_1 and k_2, $2 \leq k_1 < n$ and $2 \leq k_2 < m$.
- W: $p \times q$ matrix, $p = \binom{n}{k_1}, q = \binom{m}{k_2}$.

Degree of polynomials used by both the users in this protocol is $(\text{size of the matrix}) - 1$. Alice generates two polynomials $f_A(X)$ and $g_A(X)$, and then calculates matrices $f_A(Q)$ and $g_A(R)$.

She changes the basis of these matrices and gets new polynomials $\overline{f_A(Q)}$ and $\overline{g_A(R)}$. We saw that, by doing a change of basis in Grassmann Algebra, the two matrices computed by Alice can have a different size from matrices $f_A(Q)$ and $g_A(R)$.

Then, Alice chooses two other polynomials $u_A(X)$ and $v_A(X)$ and calculates $u_A\left(\overline{f_A(Q)}\right) = U_A$ *and* $v_A\left(\overline{g_A(R)}\right) = V_A$. Finally, she computes $W_A = U_A W V_A$ and sends it to Bob. As you can see, W has to be a $p \times q$ matrix (p and q being sizes of matrices calculated) to allow this multiplication.

Bob does the same and gets $f_B(Q)$ and $g_B(R)$, $\overline{f_B(Q)}$ and $\overline{g_B(R)}$, then U_B and V_B, and he finally sends computes $W_B = U_B W V_B$ to Alice.

After these steps, Alice calculates K_A:

$$K_A = U_A W_B V_A = u_A\left(\overline{f_A(Q)}\right) \times u_B\left(\overline{f_B(Q)}\right) \times W \times v_B\left(\overline{g_B(R)}\right) \times v_A\left(\overline{g_A(R)}\right)$$

Then, Bob computes K_B:

$$K_B = U_B W_A V_B = u_B\left(\overline{f_B(Q)}\right) \times u_A\left(\overline{f_A(Q)}\right) \times W \times v_A\left(\overline{g_A(R)}\right) \times v_B\left(\overline{g_B(R)}\right)$$

As already shown, u_A and u_B commute if the two matrices as variables commute. Here, $f_A(Q)$ and $f_B(Q)$ commute, so $\overline{f_A(Q)}$ and $\overline{f_B(Q)}$ too, because change of basis keeps the commutativity. We also can apply this to v_A and v_B. Finally, we see that Alice and Bob get the same private key $K = K_A = K_B$.

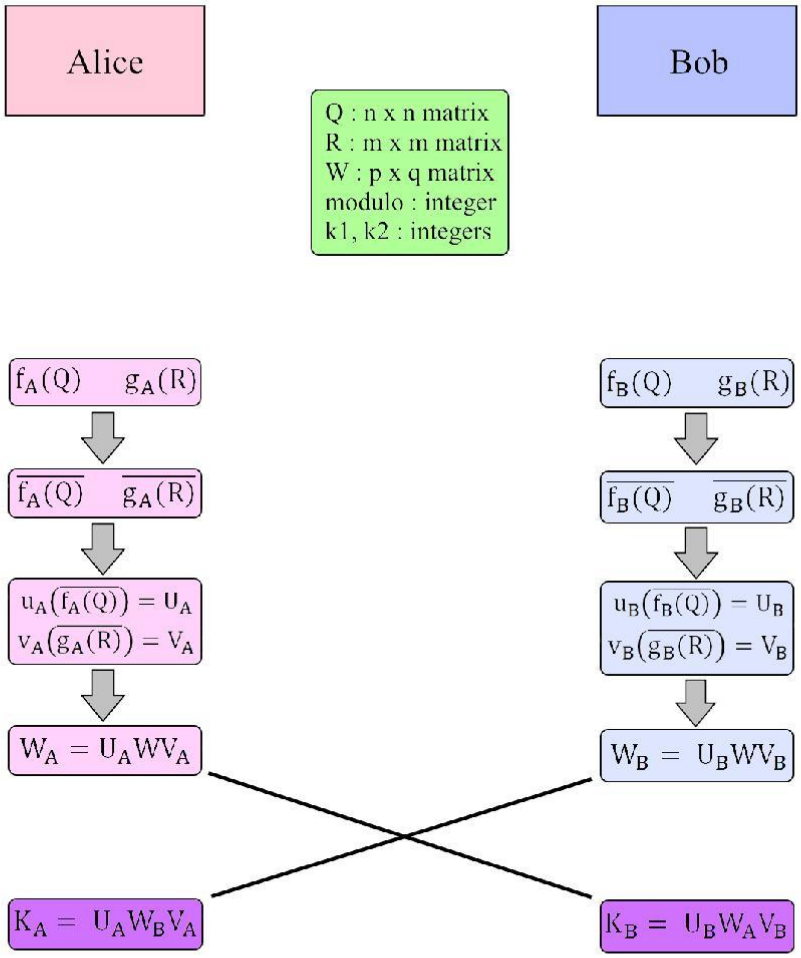

Figure 2 - Illustration of our protocol

3 Security analysis

3.1 Qualitative security Analysis

There are two ways of breaking such multivariate cryptographic systems.

This first one is the brute-force attack which corresponds to an exhaustive key search, that is to say trying to find all possible solutions and for each, the verify their validity. The complexity of brute-force attacks on cryptographic keys in n bits is, for an ideal realization,

equal to 2^n, that is to say an exponential complexity. However, in general rules, the random number generator lacks of entropy (we currently don't know how to generate a really random set of number), and consequently the key-space is generally much smaller than n bits-space (2^n). As a consequence, some implementations of modern cryptographic algorithms have been cracked by brute-force because the actual key-space was smaller than the one in the algorithm. So a particular attention has to be paid for choosing a relevant source of entropy for random number generation functions (some systems ask the user to type random keys on their keyboard with closed eyes to generate entropy), but we won't deal with this particular issue in this paper.

The second one is to resolve the system of equations derived from the internal structure of the cryptographic scheme. Then the attacker could use the Gröbner basis conversion to find satisficing solutions. However, such sets of simultaneous equations generated from the mathematical structure of a scheme are quite big and solving non-linear systems is a problem that has been proven to be NP-hard. As a consequence, even if the complexity is less than the one of a complete exhaustive search, the complexity is still exponential. As a consequence, this attack is not likely to be produced with success in the real word, and in this sense, not considered only as a practical flaw that could be exploited, but as a "certificational weakness".

Moreover, we can easily increase exponentially the maximal degree of the equations by changing the basis recursively. And doing that way, the generation time will be augmented only linearly.

3.2 Quantitative security analysis

Using two polynomials, on the left and on the right of W, allows us to choose W as an invertible matrix. With only one matrix, it would be easy to find the polynomial selected by multiplying by W^{-1}, inverse of W.

Using Grassmann algebra prevents from a possible attack coming from a malicious person. In fact, thanks to the change of basis, the system of equations obtained is non-linear. Let see what happen if we try to solve this system. Just as before, polynomials $\mathrm{f_A(Q)}$, $\mathrm{g_A(R)}$, $\mathrm{U_A}$ and $\mathrm{V_A}$ introduce n, m, p and q variables respectively. If we choose $n = m = 6$ and $k_1 = k_2 = 4$ for the change of basis part, sizes of $\overline{\mathrm{f_A(Q)}}$ and $\overline{g_A(R)}$ will be $\binom{6}{4} = 15 = p = q$. The size of the system to solve will be the size of W, i.e. $q \times p = 15^2 = 225$ equations, and the number of unknowns is 42. This kind of system is a very difficult problem, and we can choose to use matrices much bigger, introducing more equations and unknowns. If the attacker can't figure out this problem, he is no longer able to search the two matrices U_A and V_A multiplying the public matrix W, since these matrices are not direct power of public matrices Q and R. He can't find U_B and V_B computed by Bob too, so he can't calculate the private key.

Finally, the method is vulnerable to Man-In-The-Middle attack (MITM), like the original Diffie-Hellman protocol. In this case, the attacker needs to intercept the matrix sent by Alice and the one sent by Bob. For both the users, he must select polynomials and build two

matrices, as Alice and Bob. It allows the attacker to understand what the users sent, and also to possibly modify the messages. We can see it as two protocols working, the first one between Alice and the attacker and the other one between the attacker and Bob. To protect the system from this kind of threat, we can use certificates or, if it is possible, ask the users to apply a first time the key exchange, check by another way if the part of the key sent by Alice is the same matrix than the one receipted by Bob and exchange a true private key a second time.

4 Our implementation

After implementing the Grassmann change of basis in C++, we decided to use k_1 and k_2 as larger as possible. It allows increasing significantly the complexity of equations to solve. We saw it was possible to use 10×10 matrices, $k_1 = k_2 = 9$ and modulo 257 (each cell of a matrix will then be coded on one byte) in a reasonable time.

We implemented a new interface in C++ using the framework Qt. We decided to give the possibility to select a level of security and a size for the key. For the level of security, we allow to choose between five levels: for the first one, we apply only one change of basis, and for the last one, we apply five change of basis. Then, the user A has to choose the key size he wants, between for possibilities from 64 bits to 512 bits.

After that, the user A must start computation. The program creates a new file containing information for user B. The session is identified by a new Universal Unique Identifier (UUID), stored in the file name. Then user A sends the file created to user B, who has to start loading this file. The program calculates the common key and opens a new file, indicating the matrices user A needs and storing in the same manner the same UUID. User A finally loads this file. During this step, the program makes sure that the UUID is the good one.

After a few tries, it was clear that the longest part of computation is the change of basis, as expected. The complexity of a change of basis of a matrix n to the base Λ^k with k=n-1 is k! (because of the number of possible permutations of a set of k elements). This explains why with a usual personal computer, we couldn't apply Grassmann change of basis to more thanΛ^{10}, the complexity becoming too important after the threshold of 10 (see *Figure 3*).

Thanks to the C/C++ function *clock()*, we were able to determine the time it takes to do this part. We want to specify we tested this part with a laptop HP ProBook 4540s, equipped with a processor Intel i5-3230M 2.60 GHz and with a 4 GB RAM. We also used an Ubuntu 14.04 partition and GCC 4.9 to run the program.

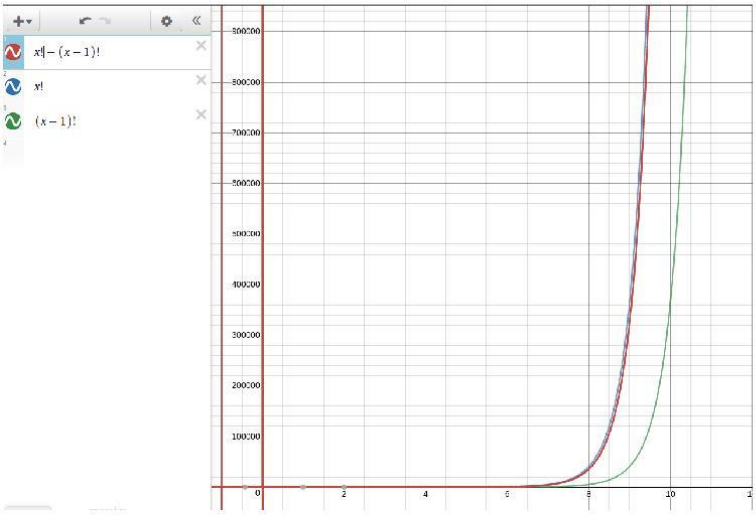

Figure 3 - Representation of the time complexity of the Grassmann change of basis to Λ^x (in blue)

In this configuration, the average of ten tests was about 3.734 seconds. Of course, if the user wants a security level of two or more, he has to multiply this time by the security level wanted. In the same way, if he chooses a key size greater than 800 bits (equal to 100 bytes, so that can be generated by the 10×10 matrix used in the implementation), he will have to multiply by the number of cycles needed to generate the chosen-length key. In comparison, it takes around one hundredth of a second to choose all the polynomials, to multiply matrices U_A, W and V_A and to put matrices into the file. It also takes less than one thousandth of a second to Alice to import the file created by Bob and calculate the private key.

For example, if Alice wants to send her part to Bob with a security level of 5, she has to calculate 5 times the change of basis for the matrix $F_A(Q)$ and $G_A(R)$, that means she has to wait about 37 seconds to send her file.

We also took the time needed to send a file into account. This time depends on the file sizes, so it also depends on key size. With 10×10 matrices, file sizes are equal to 1.5 kB. With a low connection speed of 100 Kbps, sending the file lasts 0.12 seconds. One change of basis lasts around 4 seconds, so we can ignore this duration.

The mobile application has been developed to permit the interoperability with the desktop one. It uses the same data structure, naming conventions and file types. Moreover, it has been thought to generate a key within a quite correct time, allowing the use of the mobile application in real word conditions. For example, with a Blackview BV6000 running Android 6, it takes 23 seconds to generate matrices, executing the basis change and exporting to file. This time, even if longer that on the desktop application, is relatively short for an algorithm requiring so much intensive calculus computing. This has been possible by running natively C++ code on the mobile, rather than a Java code that would have been executed on the Android virtual machine. Thus, the interface use Java-android, while the part responsible for the change of basis uses the Android NDK (Native Development Kit).

5 Block cypher over Grassmann Algebra

We saw previously in this paper how to transmit keys in an unsecured channel, but we also can use this Diffie-Hellman method to directly exchange message.

In this case, Alice, who wants to send a message to Bob, must choose two random polynomials and apply this to the two public matrices Q and R. Then she applies Grassmann change of basis and selects two other polynomials to apply on these results, exactly in the same way than already explained. She gets the two matrices U_A and V_A. Then she selects the matrix M, containing message, and sends $U_A \times M \times V_A$.

Bob, who also knows public matrices Q and R, can produce in the same manner two matrices U_B and V_B. He sends to Alice $U_B \times U_A \times M \times V_A \times V_B$. Then, she inverts her two matrices, and computes:

$$U_A^{-1} \times U_B \times U_A \times M \times V_A \times V_B \times V_A^{-1}$$
$$= U_B \times U_A^{-1} \times U_A \times M \times V_A \times V_A^{-1} \times V_B$$
$$= U_B \times M \times V_B$$

Alice sends this result. Now, Bob only needs to compute inverse matrices U_B^{-1} and V_B^{-1}. He finds the message by calculating $U_B^{-1} \times U_B \times M \times V_B \times V_B^{-1} = M$. If somebody is sniffing the network, he only knows matrices Q and R, and he won't be able to find the message.

We still have a problem for this kind of protocol: we need invertible matrices. Most of random matrices are invertible, but we need to be sure to create invertible ones. This problem is hard to solve and we need to study this problem more.