



UNIVERSITY OF GENEVA

MASTER THESIS

SECURE GROUP MESSAGING PROTOCOL, BASED ON POST-QUANTUM CRYPTOGRAPHY



Quentin RIVOLLAT

Tutor : Mr. Solana

December 23, 2020

Contents

1	Introduction	4
2	Signal Protocol	5
2.1	Introduction	5
2.1.1	Main concepts	5
2.1.2	Phases of the protocol	6
2.2	Encryption and Security	7
2.2.1	End-to-end encryption	7
2.2.2	Forward Secrecy	8
2.2.3	Post-Compromise Secrecy	8
2.3	Authentication X3DH	9
2.3.1	Publishing the keys	10
2.3.2	Sending the initial message	10
2.3.3	Receiving the initial message	11
2.4	Double Ratchet	12
2.4.1	KDF chains	12
2.4.2	Symmetric-key ratchet	13
2.4.3	Diffie-Hellman Ratchet	14
2.4.4	The Double Ratchet	15
3	Quantum Computer	19
3.1	Introduction	19
3.2	Risks	20
3.2.1	Quantum computer consequences	20
3.2.2	Risks with Signal	21
3.3	Solutions	21
3.3.1	Code-based	23
3.3.2	Lattice-based	25
3.3.3	Isogeny-based	26

4	Group messaging and ART	28
4.1	Introduction	28
4.2	Existing solutions	29
4.3	Asynchronous Ratchet Tree protocol	30
4.3.1	Introduction	30
4.3.2	Notations	31
4.3.3	ART Construction	32
4.3.4	ART Update	36
4.3.5	Complementary information	38
4.3.6	Quantum risks	38
5	Supersingular Isogeny Key Exchange	39
5.1	Introduction	39
5.2	Elliptic curve	39
5.2.1	Fundamental theory	41
5.2.2	Montgomery curve	42
5.2.3	Group law	42
5.3	Isogeny graph	47
5.3.1	j-invariant	47
5.3.2	SIDH in a nutshell	49
5.3.3	Isogeny	50
5.3.4	Example	55
5.4	SIKE protocol	55
5.4.1	Protocol's main steps	55
5.4.2	Detailing of each steps	59
5.4.3	Example	59
5.5	Key Exchange	65
6	SIKE and ART	68
6.1	X3DH and SIKE	68
6.2	Nodes' keys computations	70
6.3	Updating the leaf's keys	70

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Chapter 2

Signal Protocol

2.1 Introduction

Revise the introduction in order to better present the topic of the master> security/ encryption&secure messaging, etc.

A work like this deserves at least one full page of introduction

This is partially correct> people have been using encryption protocols like TLS&SSL well before Snowden

For a long time, most people went about their life without worrying about encryption. But with Edward Snowden's revelations in 2013, it became clear that a new step in end-to-end encryption systems was necessary.

The cryptography community spare no effort to create a next-level of secure messaging. The first ones that appeared were Pretty Good Privacy (PGP) and Off-The-Record messaging (OTR). But the most advanced and nowadays most popular protocol is currently the Signal protocol, developed by Open Whisper Systems. It is used by the Signal messaging app, but also by Whatsapp and Facebook.

2.1.1 Main concepts

Before explaining how the Signal protocol works, let's quickly see the main concepts necessary to understand the following sections.

The concept of privacy in the communication world is deeply linked to the domain of encryption.

Encryption : set of methods whose goal is to keep a message unreadable from any person who do not have the necessary information. The initial clear message is called a *plaintext*, and to obtain a message impossible to read, one needs to transform it in an alternate form, known as *ciphertext*. The process of going from a *plaintext* to a *ciphertext* is the *encryption*, while going in the opposite direction is the *decryption*.

Two persons conversing, who wants to keep private their communication from any outsider party, need an End-to-End encryption protocol.

End-to-end encryption : system of communication where only the communicating parties, meaning the sender and the recipient, are able to read the messages sent [Yel16]

There are mainly two methods to establish an End-to-end encryption between two parties : Symmetric Key Encryption, also called Symmetric Key Cryptography, and Asymmetric Encryption, also known as Public-Key Cryptography.

Symmetric Key Encryption : system providing secrecy between two or more parties by using the same key k , a shared secret, for both encryption and decryption [DK07]

Asymmetric / Public Key Encryption : system providing secrecy using pairs of keys, where each party owns a public key, used to encrypt, and a private, used to decrypt or sign [SBBB12]. The public key is accessible to anyone, while the private one is kept secret by its owner.

If Alice wants to send a message to Bob, she uses his public key to encode the *plaintext*, and Bob will decrypt the *ciphertext* by using his private key.

In the case where it is the Symmetric Key Encryption method that is chosen, the parties will first need to agree on a shared key before starting to communicate. This is done with a Key-Exchange Protocol.

Key-Exchange Protocol : mechanisms by which two parties that communicate over an adversarially-controlled network can generate or exchange a common secret key [CK01].

A network is by default always considered to be insecure, meaning an adversary could be able to add, modify or delete transiting messages.

Sometimes, a shared key cannot directly be used in an encryption protocol. A KDF is then necessary.

Key Derivation Function (KDF): algorithm that generates a cryptographic key from a password or master key [Spa16].

It can also be used to stretch keys or derive multiple keys at once.

Authentication : method used to provide the identity of an entity [ATAa14]

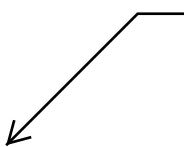
Digital Signature : mathematical scheme for demonstrating the authenticity of digital messages or documents, by computing a unique fingerprint [Pau17]

2.1.2 Phases of the protocol

The Signal protocol is divided in 3 parts :

1. Key generation
2. Key agreement
3. Key renewal

Say explicitly that Key Gen/
Agreement uses Asym Crzpto



The first part, Key generation, happens before the start of the session, i.e. before Alice and Bob have communicated. The protocol used there is the eXtended Triple Diffie-Hellman, or X3DH.

Then, the second part, Key agreement, is done through the first message that Alice send to Bob, where both parties agree on an initial key, necessary for the rest of the Signal protocol. X3DH is also employs in this step.

Finally, the last step step, Key renewal, takes place each time a message is sent, leading to an update of the key. This time, it is the Double Ratchet algorithm that is used.

2.2 Encryption and Security

2.2.1 End-to-end encryption

The first goal of a secure messenger app is to provide absolute secrecy about the transmitting messages, along with authentication from every parties involved in a session.

As mentioned in the first part of this chapter, an end-to-encryption allows only the concerned parties to encrypt and decrypt a message. No-one, not even an intermediate server, should have the possibility to obtain the plaintext corresponding to a ciphertext.

There are basically two ways to encrypt a message : either with symmetric or asymmetric keys. Both methods have their advantages and disadvantages. The symmetric one seems to be the most efficient. Indeed, this way of encryption is more secure than the Public Key Encryption method (PKE) :

- Speed : since the decryption of a message should be half the time of the encryption step is much slower than the latter, but both are slower than the benchmark done to compare [Doe19]
- Security : security level [Lef04] is a measure of the resistance to cryptographic attacks, related to the key length. In order to have comparable security level between symmetric and public keys encryption, the latter will always need a longer key than the first one (see summarizing table at [P+14])

The "faster" argument is clear. The "more secure" one is arguable and the reference provided is not conclusive in my opinion...

Hence, the symmetric key method will be the one used in this whole paper when it comes to encrypt a message. But this method is not perfect. There is one major single point of failure that comes right in mind : what if the shared key used to encrypt and decrypt is revealed or obtained by an adversary ? The repercussion would be immediate : all messages encrypted with this specific key will lose the property of secrecy, meaning the new possessor of the symmetric key will be able to decrypt them all.

This eventuality should be avoided at all cost. If a key is leaked, the consequence should be as small as possible, i.e. ideally almost none ciphertext should be decryptable.

unclear

2.2.2 Forward Secrecy

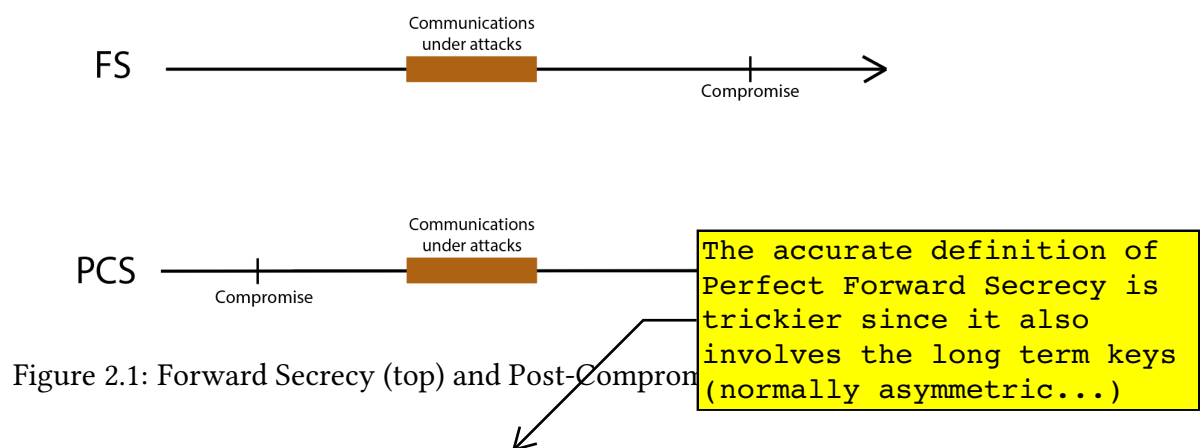
Two things have just been said :

1. A ciphertext is vulnerable if the key used to encrypt the corresponding plaintext has been leaked
2. We want as few vulnerable ciphertexts as possible

So, the logical solution would be that at each encryption stage, one uses a new key. This new key cannot be totally random, since both parties implicated in a conversation must have the same shared key in order to encrypt and decrypt. Instead, we derive the key at time t based on the previous one, by using a Key-Derivation Function :

$$k_t = KDF(k_{t-1})$$

Then, by starting with a common shared secret key k_0 , the parties implicated in the conversation will be able to obtain the same keys. We thus obtain Forward Secrecy (see Figure 2.1).



Forward secrecy : A protocol provides Forward secrecy if the leak of a key at time t does not threaten the security of the messages prior to this key k_t [CGCG16].

2.2.3 Post-Compromise Secrecy

Nevertheless, it is not enough. If an adversary obtains the key at time t , he will still be able recalculate all the following keys simply by applying the KDF onto this leaked key, and thus to read every message encrypted and sent after this moment t . That is why future, or post-compromise, secrecy is necessary as well.

Post-Compromise Secrecy : A protocol provides Post-Compromised secrecy if the leak of a key at time t does not threaten the security of the messages encoded after this key k_t . [CGCG16]

Same applies here...

In order to obtain such a property, we could add as input to the KDF another value, independent to the key k_t , but shared between the parties. We would then obtain what we call one-time keys, i.e. keys used only once, and then deleted. (see Figure 2.2)

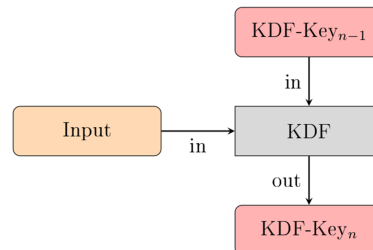


Figure 2.2: Key Derivation Function

An example of an algorithm that has both Forward and Post-compromise secrecy is the Diffie-Hellman Ratchet (see Section 2.4.3)

2.3 Authentication X3DH

Better introduce the scenario for messaging: need for a server, etc.

We now know a way to obtain a good level of security during a communication between two parties, let's say Alice and Bob. Nevertheless, if it is Alice that initiates the exchange of messages, she must be certain that she's speaking with Bob, during the whole session. Thus, we need protection against a man-in-the-middle attack, where an attacker could impersonate Bob. That is why authentication is a key phase.

Extended Triple Diffie-Hellman, or X3DH, is a key agreement protocol, whose goal is to create a shared secret key between two parties but also to authenticate each other with the help of public keys. [PM16b]

One big advantage of X3DH is that it is designed to work asynchronously, meaning that if Alice wants to start the protocol, then Bob doesn't need to be online. He only needs to have previously published a set of keys, that Alice will fetch and use.

The Table 2.1 lists all the public keys necessary in order to apply X3DH (if Alice initiates it). Of course, each public key has a corresponding private key. Moreover, these public keys used here are based on elliptic curve (see Elliptic Curve Diffie-Hellman, or ECDH, [Bro09]).

The X3DH protocol is divided into three phases :

1. Bob publishes to a server his identity key and prekeys, forming a “prekey bundle”
2. Alice fetches the prekey bundle from the same server, uses it to compute a shared secret, and send an initial message to Bob
3. Bob receives the initial message that Alice sent, and processes it

Name	Definition	Duration	
IK_A	Alice's identity key	Long-term	bound to identity
EK_A	Alice's ephemeral	One-time	never reused
IK_B	Bob's identity key	Long-term	bound to identity
SPK_B	Bob's signed prekey	Medium-term	reused across sessions
OPK_B	Bob's one-time prekey	One-time	never reused

Table 2.1: Keys used in X3DH

After a successful protocol run, Alice and Bob will share a shared key (SK), that will be used later in the second and third part of the Signal protocol.

2.3.1 Publishing the keys

The very first thing that is done when a party decides to use an application based on the Signal protocol is publishing several specific keys onto a server. Everyone **needs** to publish these keys in order to ensure authentication and privacy in future message exchanges. This set of keys of commonly called a *prekey bundle*.

The keys that any new party places into this common server **For enhanced clarity, it would be preferable to indicate the public/private nature of each key**

- IK : An identity key
- SPK : A signed prekey, with its signature, using IK
- $\{OPK^i\}_{i=1}$: A set of n one-time prekeys

Obviously, the owner of these public keys possesses all the corresponding private keys, but doesn't show them ; they are kept secret. The identity key is a long-term key and it should stay the same as long as possible. The signed prekey is a medium-term key. Bob updates it at some previously defined interval. And the one-time keys are, as their names indicates, used only once, and then deleted. Once all the keys are used (or shortly before), Bob should upload a new set of public one-time keys.

2.3.2 Sending the initial message

In order to perform an X3DH key agreement with anyone, Alice first needs to fetch his prekey bundle, by making a request to the server. Therefore, Alice will be in possession of :

- Bob's identity key (IK_B)
- Bob's signed prekey (SPK_B) and its signature
- One of the Bob's one-time prekey (OPK_B)

First, Alice verifies the prekey signature. It is important to include signature, because without it, the system would allow a “weak forward secrecy” attack [PM16b] : a malicious server could give to Alice a prekey bundle containing forged prekeys, and later compromise Bob’s IK_B , by calculating SPK.

Next, she can calculate the following Diffie-Hellman¹ values (she uses her private keys, and Bob’s public keys) :

- $DH_1 = DH(ik_A, SPK_B)$
- $DH_2 = DH(ek_A, IK_B)$
- $DH_3 = DH(ek_A, SPK_B)$
- $DH_4 = DH(ek_A, OPK_B)$

where DH is the function applying Elliptic Curve Diffie-Hellman, in order to obtain a shared secret. Finally, Alice can compute the shared secret SK :

$$SK = KDF(DH_1 || DH_2 || DH_3)$$

where $||$ represents the concatenation operation, and KDF is the Key Derivation Function².

To sum up the 4 first operations using DH, we can use Figure 2.3

These combinations of keys have the following goals : DH_1 and DH_2 use identity keys to provide mutual authentication, while DH_3 and DH_4 use one-time and medium-term keys to provide forward secrecy.

Now, Alice can send Bob the initial message we talked about, containing :

- IK_A : Alice’s identity key
- EK_A : Alice’s ephemeral key
- i : identifiers indicating which Bob’s prekeys that Alice used
- C : an initial ciphertext, encrypted with the shared key SK

2.3.3 Receiving the initial message

The initial message that Bob received contains, ie. (IK_A, EK_A, i, C) , Alice’s identity key and ephemeral key, with the ciphertext and the indication i . With this information, Bob is able to repeat the DH and KDF computations, by using its private keys, as did Alice, and can derive SK. And finally, he can decrypt the secret message by using the shared secret. At the same time, he authenticates Alice as being the sender.

The Figure 2.4 sums up the different phases in the X3DH protocol.

¹Diffie-Hellman is a method to securely exchange cryptographic keys : given the key pairs (a, g^a) and (b, g^b) , respectively owned by Alice and Bob, the shared secret is g^{ab} . g is a public parameter

²see Section 2.1.1

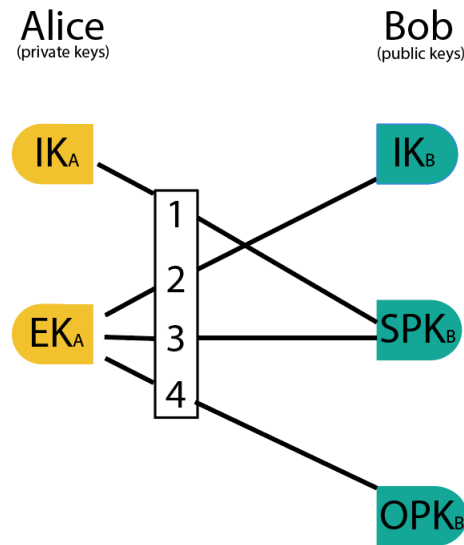


Figure 2.3: Four shared secrets between Alice and Bob

2.4 Double Ratchet

The Double Ratchet algorithm is the second main part of the Signal Protocol. It was developed by Trevor Perrin and Moxie Marlinspike in 2013, and is used in order to provide end-to-end encryption for instant messaging. The major difference with another usual symmetric encryption algorithm is that Double Ratchet manages the ongoing renewal of session keys, in order to provide Future and Post-compromise secrecyes.

The algorithm is a combination the Diffie-Hellman Ratchet and the Symmetric-key Ratchet, the latter being based on Key Derivation Functions. All these concepts will be explained in the following sections.

2.4.1 KDF chains

As defined earlier, a KDF is a cryptographic function that takes a secret and a random value, to return data [PM16a]. The reason of the random value is to obtain an indistinguishable output from random data. KDF chain is an important concept employed in the Double Ratchet Algorithm. This term designs the process of using a part of the output from a KDF as input of the following KDF step. The other part is used as output key.

In each Double Ratchet session, Alice and Bob stores a KDF key for three chains : a root chain, a sending chain and a receiving chain (Alice's receiving chain matches Bob's sending chain, and vice versa).

As Alice and Bob exchange messages, they also communicate their new respective Diffie-Hellman

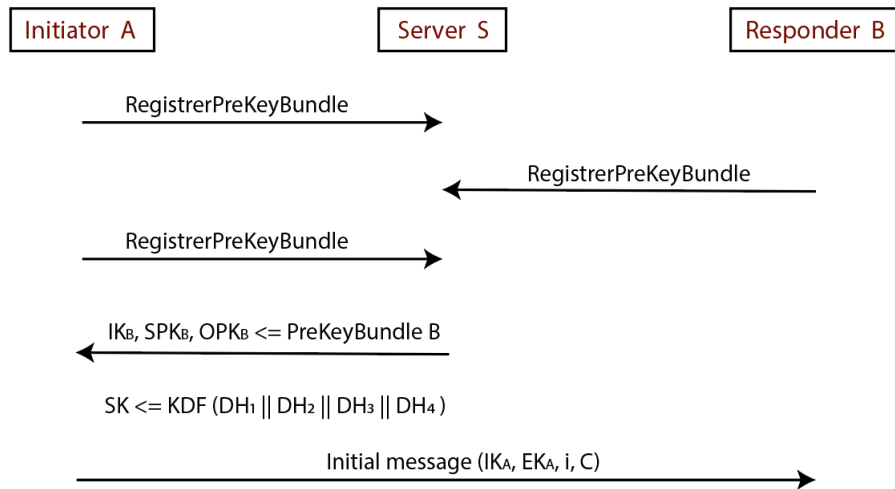


Figure 2.4: Example of message flows during the execution of X3DH between two clients A and B via a server S, initiated by A

key (the public one), and the Diffie-Hellman secrets calculated become the inputs to the root chain. Then, the output keys from the root chain become new KDF keys for the sending and receiving chains. This process is called the Diffie-Hellman ratchet.

The sending and receiving chains both rely on the root chain, and each one of them can be updated. Indeed, the sending and receiving chains are rebooted when the root chain is updated, and the latter event happens when a party receives a new Diffie-Hellman public key. With this new public key, and by generating a new private key, one can then recompute a DH shared secret, apply KDF, and eventually update the root chain value. Nevertheless, as long as the root key is kept the same, the sending and receiving chains are also updated each time a message is sent or received. The output message keys obtained are used for the encryption or decryption of the messages. This process is called the Symmetric-key Ratchet.

2.4.2 Symmetric-key ratchet

The Double Ratchet algorithm is used by two parties to exchange encrypted messages, based on an initial shared secret key, like the one Alice and Bob derived with the X3DH protocol.

For every message sent, the parties derive new keys, so that earlier keys cannot be computed by using later ones. Moreover, the parties send Diffie-Hellman public values as well, attached to their messages.

This latter value is used in the KDF (with the previous key) as input, to generate a receiving chain key and a sending chain key. On both chain, Alice applies a symmetric-key ratchet, resulting in

a new chain key (receiving or sending) and a message key, which is the key used to encrypt (or decrypt) a message. The result can be seen in the Figure 2.5.

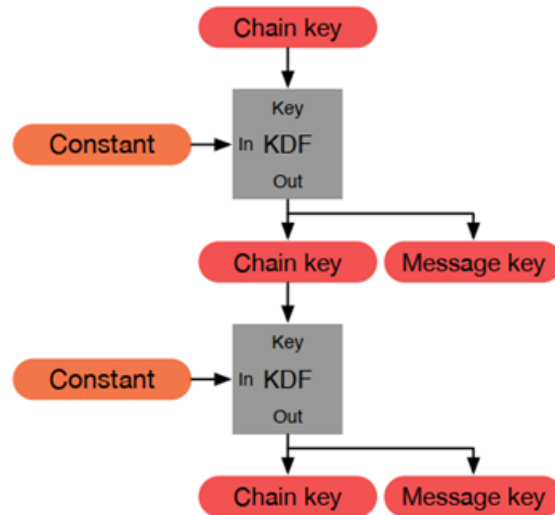


Figure 2.5: Symmetric Ratchet

2.4.3 Diffie-Hellman Ratchet

A possible algorithm that one could use in order to achieve Forward and Post-Compromise Secrecy is the Diffie-Hellman Ratchet algorithm. The method enables both Alice and Bob to encrypt every message with a different symmetric key. As one would expect, it relies on the DH key exchanges.

The basic idea of DH Ratchet is that each party generates a DH key pair (k, K) , where k is the private key and K the public one. We call this pair their current ratchet key pair. When a new ratchet public key is received by a party, a DH ratchet step is performed, consisting of updating the receiving party's current ratchet key pair with a new pair of keys. Then, the party computes the DH output, based on its private key and the public key received.

If Alice sends, after that, her public key, Bob should be able to derive the same DH output, as illustrated in Figure 2.6.

Let's say Alice wants to send the first message to Bob. Then, she first needs Bob's public key B_1 , and follows the next steps :

1. Generate a new DH key pair : (A_1, a_1)
2. Create a shared secret $\text{DH}(B_1, a_1) = ss_1$
3. Derive a symmetric key : $S_1 = \text{KDF}(ss_1)$

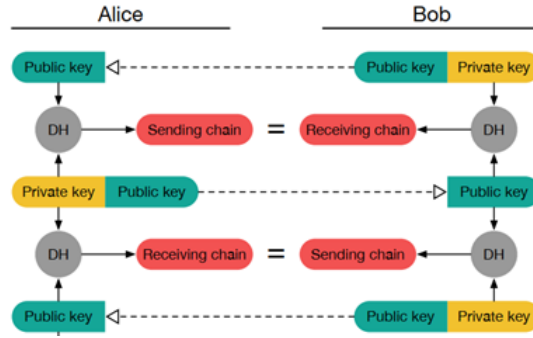


Figure 2.6: Diffie-Hellman Ratchet chain

4. Encrypt the message with the key : $CT_1 = \text{encrypt}(M_1, S_1)$
5. Send the tuple (ciphertext, A's public key) = (CT_1, A_1) to B

The decrypt the message, Bob follows the next steps :

1. Create the shared secret : $DH(A_1, b_1) = ss_1 = (DH(B_1, a_1))$
2. Derive the symmetric key
3. Decrypt the message

These steps described the example in Figure 2.7

This means that we could say the shared secret derived by Alice is a sending chain, and the one used by Bob to decrypt is a receiving chain (as shown in Figure 2.6)

If Bob wants to send Alice an answer, he follows the same steps Alice did, but before he needs to generate a new DH key pair (B_2, b_2) and a new symmetric secret $DH(A_1, b_2)$, with A's public key. He will eventually send the ciphertext alongside his public key B_2

With such an algorithm, the keys are updated after every message, meaning that if a private key a_i leaks, only the ciphertext CT_i and CT_{i+1} won't be protected anymore, since the symmetric key used to encrypt CT_i is based on (a_i, B_i) and for CT_{i+1} , it is based on (a_i, B_{i+1}) .

An example is given with the illustration in Figure 2.7.

2.4.4 The Double Ratchet

If we want to be precise, the above drawings and explanations are a simplification of what's really happening, because in reality, after calculating the DH output, Alice and Bob apply the symmetric ratcheting in order to derive the receiving chain key and sending chain key, as shown in Figure 2.8.

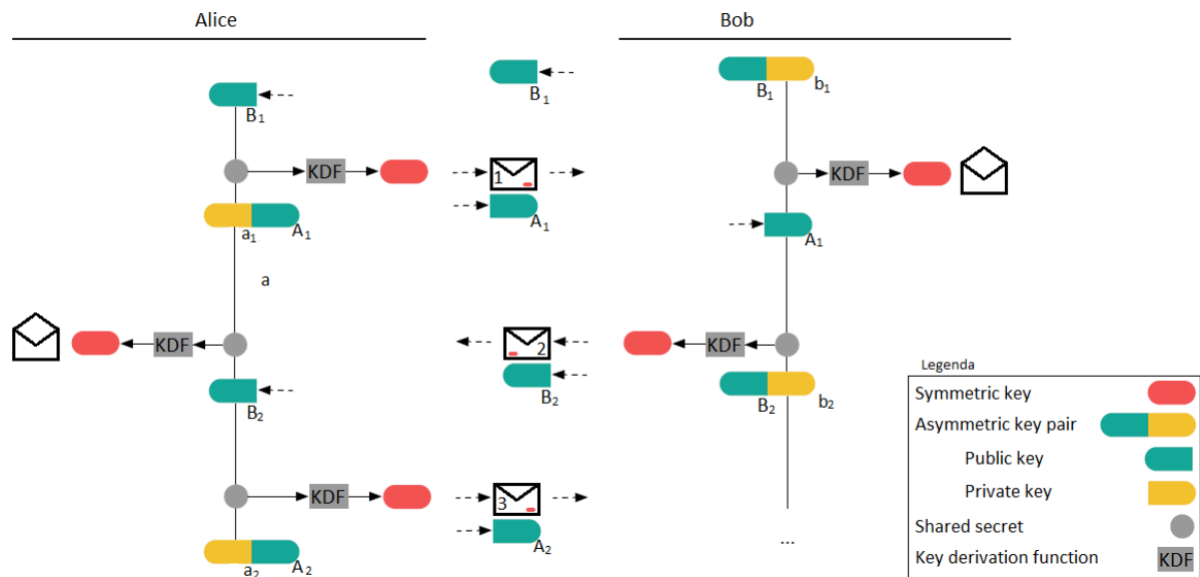


Figure 2.7: Diffie-Hellman Ratchet chain

Then, each time Alice receives a message, her receiving chain key advances (she just apply KDF on it), and she gets a new receiving chain key and a message key, the latter one being used to decrypt the message. And similarly, when she wants to send something, she updates her sending chain key. By following these steps, Alice's receiving chain key should be equal to Bob's sending chain key.

Hence, by combining the symmetric-key and DH ratchets, we obtain the **Double Ratchet** :

1. When a message is sent or received, a symmetric-key ratchet step is performed to the sending or receiving chain to derive the message key
2. When a new ratchet public key is received, a DH ratchet step is applied before the symmetric-key ratchet, to replace the chain keys.

A last example is shown in Figure 2.9.

Here is how it goes :

- (i) An initial root key (RK) is obtained after X3DH
- (ii) Alice receives Bob's DH public key, creates a new pair for her, and calculates the DH value
- (iii) Alice applies the KDF chain, with the DH value and RK as inputs, and obtains a sending chain key.
- (iv) She wants to send a message M_1 (A). She then applies KDF on her sending chain key, and gets a symmetric key KA_1
- (v) Alice sends the encrypted message, with her public DH key, to Bob

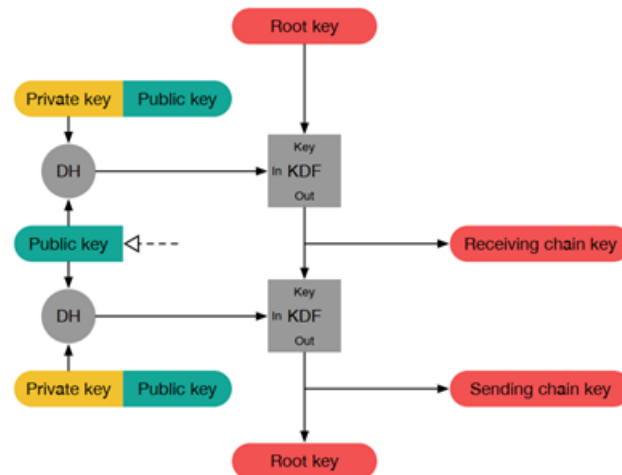


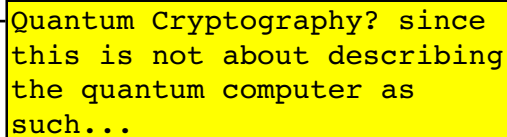
Figure 2.8: Double Ratchet

- (vi) Alice receives two messages from Bob, with his public DH key. She derives a new DH value
- (vii) With this value, advances the KDF chain, to obtain a receiving chain key, and apply twice the receiving chain ratchet, to derive two message keys, used to decrypt the corresponding ciphertext
- (viii) Alice create a new DH pair, calculate the DH shared secret, and apply a KDF chain ratchet, to derive a new sending chain key
- (ix) She applies three times the sending chain ratchet, to get multiple message keys, and encrypt three messages with different keys, and then send them.

Thus, we obtain Forward secrecy and Post-compromise secrecy, but also resilience (the output keys appear random to an adversary without knowledge of the KDF keys and break-in recovery (if an adversary learns the KDF key at a time t , the future output keys will still appear random [PM16a]).

Chapter 3

Quantum Computer



Quantum Cryptography? since this is not about describing the quantum computer as such...

Today's computers, both in theory (with Turing machines) and practice (PCs, laptops, smartphones, etc...), are based on classical physics. So, they are limited by the ability to be in only one state at the time. With quantum physics, things are different. Indeed, a quantum system can be in a superposition of many different states at the same time, enabling much more efficient computation, and thus shorter time for some algorithm, compared to classical computers. The main threat that represents these kind of computers rely on their capacity to undermine common cryptographic defenses.

Even though the machines aren't powerful enough to do this today, they are still evolving rapidly. That is why a new defense mechanism is necessary, as soon as possible.

3.1 Introduction

A quantum computer could simply be described as a computer using quantum mechanics so that it can perform certain kinds of computation more efficiently than a regular computer can [Sug18].

To be more precise, while a classic computer uses bits to store information, a quantum computer uses qubits, elements that can be set to 0 or 1, but also to 0 and 1, and thus can compute algorithm once on multiple different inputs simultaneously, enabling faster computations, even though the risk of errors increases with the size of the input.

The idea of a quantum computer appears first with Richard Feynman, in 1981, who mentions that simulating quantum mechanical systems on a new kind of machine following the quantum principles could result in great computation performance [Str11].

But the first really major development in quantum computation was brought by Peter Shor, with his publication in 1994 of a quantum algorithm that could factorize prime integers in polynomial time. This paper provided a way to reach exponential speedup over the fastest classical algorithms, meaning that a lot of public-key cryptography systems were threaten by this, such as RSA which depends on the fact that there is no known efficient classical algorithm that factors integers into prime numbers.

The second breakthrough that impacted some security algorithms is the Lov Grover's quantum database search. This algorithm searches, in an unordered database, a specific entry, using amplitude amplification to achieve a polynomial speedup over the best existing classical algorithms [Sug18]. Grover's discovery can solve some NP problem, such as the SAT problem and graph colouring. To sum up, because of Grover's algorithm, one need to double the key sizes of symmetric key cryptography in order to obtain the same security level than with classical computers.

3.2 Risks

3.2.1 Quantum computer consequences

With the revolution provoked by quantum algorithms, some cryptographic primitives are not secure anymore. Table 3.1 lists the main algorithms used in encryption, and the consequence of quantum computers.

Cryptographic primitive	Type	Purpose	Impact from quantum computer
AES-256	Symmetric key	Encryption	Larger key sizes needed
SHA-256	One-way	Hash functions	Larger key sizes needed
RSA	Public key	KEP, Signature	No longer secure
ECDSA, ECDH	Public key	KEP, Signature	No longer secure
DSA	Public key	KEP, Signature	No longer secure

Table 3.1: NIST impact analysis of quantum computing on encryption schemes [BW17]

RSA : Breaking the RSA encryption scheme is possible on a classical computer, through factoring, since inverting the RSA function is categorized as hard as a factorization. The ways in which RSA might be attacked are well studied, since quite a long time [BW17], and the attacks are well documented. But the problem for each method is the efficiency. This is what makes RSA secure to classical computer.

Peter Shor's algorithm is designed to factor integers into their prime number components, using Euler's method, i.e. determining the period of the function. This method runs on a quantum computer in polynomial time, whereas the General Number Field Sieve, which is fastest factoring algorithm we currently have on conventional computers, runs "only" in a super-polynomial time [BW17] [LL93].

Diffie-Hellman : DH key-exchange protocol is at the foundation of public-key cryptography. The objective of such a protocol is to enable two parties to derive a shared secret, from each other's public key and their own private keys. Diffie-Hellman problem is said to be as hard as computing discrete logs, since an adversary's goal is to find a from $A = g^a$. The best discrete logarithm algorithm effective today is from Barbulescu, Joux and Thomé, but it runs in quasipolynomial time [BGJT14], which is slow enough to keep DH secure from classical computing.

Better address the extension of this risks to computations on elliptic curves. The issue is oversimplified...

Shor's quantum algorithm is able to solve the Discrete Logarithm Problem (DLP), and thus breaks Diffie-Hellman, in polynomial time. So, a lot of cryptographic protocols that are based on Diffie-Hellman are directly threatened, such as ECDH (Elliptic Curve Diffie-Hellman).

DSA : since a lot of Digital Signature Algorithms are based on using public keys pair (private key to sign, public key to check), it becomes easy to an adversary using quantum algorithm to forge a signature into a message, i.e. fabricating a digital signature for a message without have access to the respective signer's private signing key. The security property non-repudiation is then lost.

3.2.2 Risks with Signal

As we have seen, some important cryptographic protocols are more or less impacted by quantum computers. We then need to adapt the Signal protocol to those new risks, in order to get a resistant protocol, by checking the primitives at risk.

Table 3.2 lists the cryptographic primitives used either in X3DH or in Double Ratchet, with the corresponding algorithm ([PM16a] [PM16b]).

Signal Protocol's part	Primitives	Algorithm
X3DH	Key Exchange	ECDH (Curve25519)
	KDF	HKDF (SHA-512)
	Signature	XEdDSA (Curve 25519)
Double Ratchet	Key Exchange	ECDH (Curve25519)
	KDF	HKDF (SHA-512)
	Encryption	AES-256 (CBC mode)

Table 3.2: Cryptographic primitives used in the Signal Protocol

The hash function SHA-512 and the symmetric encryption primitive AES are not directly threatened by quantum computers. Nevertheless, it is necessary to take larger keys in order to keep an acceptable level of security.

The most concerning aspect here is ECDH Curve, which is no longer secure. It is absolutely necessary to trade it for a post-quantum key exchange protocol. SIKE is one of them, and is detailed in Chapter 5.

3.3 Solutions

As briefly explained in Section 3.2, with quantum computers :

1. Shor's algorithm could break all classical factorization and discrete log-based public key crypto

Too schematized... elaborate
bit more....

2. Grover's algorithm could force doubling the key sizes of symmetric key cryptography.

Two alternatives have been proposed [Mat19]:

- the physicists say : "Use quantum technologies to fight quantum technology !"
⇒ Quantum cryptography
- the mathematicians say : "Base your cryptographic systems on mathematics that quantum computers can't break"
⇒ Quantum resistant or Post-Quantum cryptography.

The main purpose of Quantum cryptography has been to reduce as much as possible the breaking of public key cryptography from quantum computers, with Quantum Key Distribution, or QKD.

Based on fundamentals in quantum physics, QKD enjoys a secure way to distribute keys through insecure channels. [QQL10]

But Quantum cryptography has several drawbacks :

- mainly limited to quantum key distribution
- provides no authentication
- has problem with large distances
- does not scale well

That's why Post-Quantum (PQ) cryptography is preferred.

In order to be efficient and easily used by as many systems as possible, PQ cryptographic systems should respect some basic criteria. First, they should run efficiently on classical computer (time, memory, ...). Then, they must be hard to break by both quantum and classical algorithms (since it is *Post-Quantum*, by definition). Eventually, they need rely on different mathematical problems than integer factorization or discrete logarithms, since these kind of problem are pointless again quantum computers.

There are many Post-Quantum cryptographic algorithms, but we can categorize them, and the types we are interested in are :

- Code-based
- Lattice-based
- Isogeny-based

There are other types, like Multivariate and Hash-based, but are designed for multivariate statistics or signature scheme, which we won't focus on it in this paper.

work

3.3.1 Code-based

Code-based cryptography is an old method, based on coding theory, used for the last 40 years. Even though it was considered unusable for a long time due to the large key size necessary, in particular compared to number-theoretical cryptosystems, like RSA or ECDSA, code-based systems are nowadays reconsidered, because of their capacity to resist to quantum computations.

Code-based cryptography is based on using error correcting codes. These codes were originally developed to improve communication accuracy, since transmitting information over an unreliable channel would often provoke noise and errors. In order to correct an inaccurate received message, an error-correcting code is added to the message before sending it, so that the receiver can retrieve the exact message, or at least notice that the message received is incorrect and ask for a retransmission. The method used to do so is based on linear codes, i.e. we can use matrix-vector multiplication to encode and decode messages (see [MLS19] for more explanations).

This concept has been developed in order to be resistant to quantum computers : the message is, as before, converted into a code, but then, one deliberately adds a secret error to it. And because the receiver knows the parameters used (it's a shared secret between the two parties), he is able to retrieve the original code, and thus the message. The security relies on the fact that an adversary should not be able to distinguish the exact code from a randomly generated one. This is made possible by having a public key which is a scrambled version of a generator matrix, the latter being used to encrypt the message.

McEliece cryptosystem :

In general, public key cryptography relies on the idea that messages should be easy to encrypt, but hard to decrypt without private information. Let's assume that G is the generator matrix of a code, meaning that by multiplying with a vector, we obtain a code, i.e. an encrypted message. So, we can multiply the message M by the generator matrix G , to get a code word $c = M \cdot G$ and then add some arbitrary error vector e . Hence, the ciphertext, denoted as T_{Mc} , is :

$$T_{Mc} = M \cdot G + e$$

If one knows the generator matrix G , with the fast decoding algorithm, they can easily decrypt the message M , by finding the closest vector to $c + e$.

Nevertheless, with such a method, there is no security, since an adversary could apply the same algorithm, and retrieve the plaintext M . The McEliece Cryptosystem makes the fast decoding algorithm undoable for someone who doesn't know the correct coding parameters.

The idea of the McEliece cryptographic system is to scramble G , by multiplying it by some matrices. The result is published as the public key [MLS19]. These matrices are S_{Mc} , random and invertible, and P_{Mc} , a permutation matrix, and represent the shared secret. Thus, we get :

$$G' = S_{Mc} \cdot G \cdot P_{Mc}$$

By constructing G' this way, G is very well hidden, and one can eventually compute the new ciphertext :

$$T_{Mc} = M \cdot G' + e$$

The decryption is done by following the next rules :

1. Since P_{Mc} and S_{Mc} are known from both parties, the recipient can inverse the former matrix and multiply it with T_{Mc} :

$$T_{Mc} \cdot P_{Mc}^{-1} = (M \cdot S_{Mc} \cdot G \cdot P_{Mc} + e) \cdot P_{Mc}^{-1} = MS_{Mc}G + eP_{Mc}^{-1}$$

2. A fast decoding algorithm Δ retrieves the nearest code word c from a vector of the form $w = c + e$, using Hamming distance.

If the ciphertext was exactly $MG + e$, this algorithm would give us :

$$\Delta(MG + e) = M$$

3. One finally multiplies it with the inverse of S_{Mc} :

$$\Delta(T_{Mc} \cdot P_{Mc}^{-1}) \cdot S_{Mc}^{-1} = \Delta(MS_{Mc} \cdot G + eP_{Mc}^{-1}) \cdot S_{Mc}^{-1} = MS_{Mc} \cdot S_{Mc}^{-1} = M$$

Figure 3.1 sums up these operations.

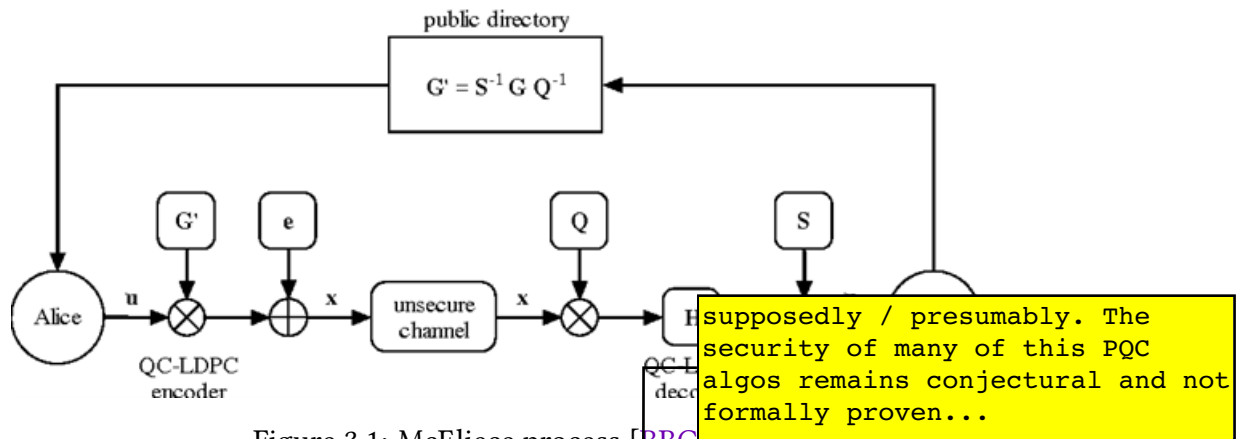


Figure 3.1: McEliece process [BBC15]

The advantages of McEliece cryptosystem are that is efficient and secure against quantum computers, but the main problem is the large key sizes necessary, even though attempts have been made to reduce it, like introducing more structure into the codes.

3.3.2 Lattice-based

This topic is probably the most active one among all the research made in post-quantum cryptography. Indeed, it only involves simple linear operations, like additions and multiplications with modulo, similarly to RSA, and enjoy strong security proofs, thanks to the worst-case hardness of lattice problems, which has been conjectured to be hard to solve in both classical and quantum configurations.

The ideas behind lattice-based cryptographic systems are similar to the ones in code-based.

Shortest Vector Problem

Lattice : Let $\{b_i\}_{i=1}^m \in \mathbb{R}^n$ be m independent vectors. Then, an n -dimensional lattice L is the span of all these vectors b_i , i.e. :

$$L = \left\{ \sum_{i=1}^m x_i b_i \mid x_i \in \mathbb{Z} \right\} (= L(B))$$

So, intuitively, a lattice is a set of points, with a base B that can be considered as a matrix whose columns are the vectors b_i .

The idea behind lattice-based cryptography is that one can use this well-formed space as a secret key, and a scrambled version of it as a public key (like we scrambled the matrix G in the Code-based section). The sender can now map the message to a point on the lattice, and then add an arbitrary error, such as the new point is still closer to the original one than to any other point. Since the recipient knows the correct well-formed base, he can decrypt it by finding the closest vector to the received point. The security relies on the idea that it is hard for an attacker to find the closest point used only a kind of random base.

This can be rephrased as the following problem :

Shortest Vector Problem (SVP) : Given a basis B of a lattice L , find the shortest nonzero vector in $L(B)$.

For the approximation variant of SVP is reduced at finding a vector v whose length is lower than γ the length of the shortest nonzero vector, i.e. : $\|v\| \leq \gamma \cdot \lambda_1(L)$, where λ_1 design the shortest vector in L .

What makes SVP important is that they are classified as hard problems, and quantum algorithms are not much more efficient at solving this kind of problem. Hence, a polynomial time quantum algorithm that can approximate these problems in polynomial time does not exist.

But the reason we are talking about finding a shortest vector in a given lattice is that the decoding process of a received message consists in finding the closest code c from a given non-exact code $w = c + e$. This problem is called Closest Vector Problem (CVP).

CVP is a generalization of the Shortest Vector Problem. That is why it is important that SVP is NP-hard : quantum cannot solve easily this kind of problem neither without complementary information.

elaborate more on this difficulty for Quantum computers to address NP-hard problems...

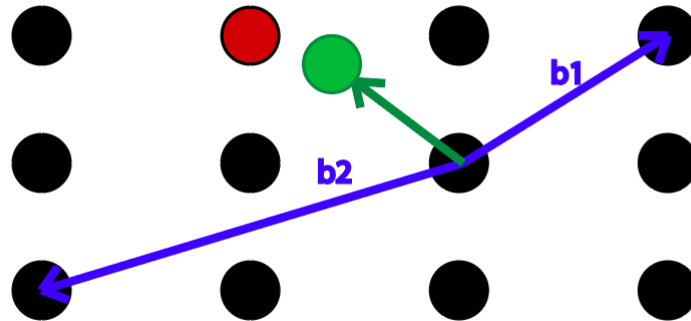


Figure 3.2: Closest Vector Problem : the basis is given by $\{b_1, b_2\}$, and the recipient receive the green vector ; they need to find the closest vector, or closest point, which is the red one here

3.3.3 Isogeny-based

While the classic Elliptic Curve Cryptography (ECC) could be summarized as finding hidden relation between two points P and P' on a given elliptic curve E , Isogeny-based cryptography is more about discovering connection and operations between two elliptic curves E and E' inside a certain large pool. These operations that map a curve onto another one with certain properties are called isogenies. The difficulty is that it is supposed to be hard to find the isogeny between two specific EC without more information about them. This information is kept secret into the private key, while the public information is the two elliptic curves.

Hence, the problem that a quantum computer would have to solve is the following :

Problem : Given an elliptic curve E' obtained by applying an isogeny into a base curve E , find how to get from E to E' , using the same map [Mat19].

Supersingular Isogeny Diffie-Hellman exchange uses conventional elliptic curve operations, provide perfect forward secrecy, and can be used for computing a shared secret between two parties. [JDF11] More information in Chapter 5.

same here: elaborate a bit more...

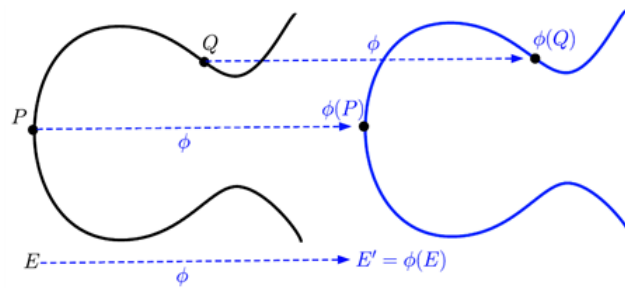


Figure 3.3: Example of Isogeny : here the isogeny is just a translation [[Aza](#)]

Chapter 4

Group messaging and ART

three or more entities

Until now, we only talked about two-parties end-to-end encryption communication, and the security properties we obtained using the Signal Protocol. But what is true with a two-members conversation may not be with three persons.

In a two-parties messaging protocol like Signal, one can create a session with his future partner offline. It doesn't prevent him to prepare everything and then send an initial message with all the important information to the other party. We can have asynchronicity. Moreover, we also have the following properties : privacy, Forward secrecy, Post-compromise secrecy, deniability and authentication.

But what about a n -parties group communication, with $n > 2$? Can we still have the same properties?

deniability and how it is achieved in this framework needs to be explained in previous sections...

4.1 Introduction

From now on, we must consider that there are more than two members of a group that would like to exchange messages. How can it be done ?

Well, a very basic solution would be to create pairwise channels between all the members, in which it is possible to apply the exact Signal Protocol. This means that, in a group of n persons, if Alice wants to send a message, she has to send it to everybody using each common secret she shares with a member. An obvious advantage is that everybody keeps the security properties from the Signal Protocol, since it is the same approach, but with more people.

But the major problem is that more members means more message to send each time, and for voluminous messages, it can become problematic, especially if the bandwidth is limited.

★ another solution would be to use a similar protocol in order to create a unique shared group key, used by everyone to encrypt and decrypt, but of course kept private to members outside the group. Then, when the message is encrypted, one party just need to send it to the server, which will re-send it to everyone else. This is what we call server-side fanout.



Let's explore some already existing solutions.

4.2 Existing solutions

Since group messaging is already existing in most messenger app, this means that each one choosed a method to implement it. We present here some of them [CGCG⁺18]

- **OTR-style** : Off-The-Record protocol relies on the idea of deniability, and this was maintained for group communication. The idea is the following : first, members of the group conduct some interactive rounds of communication to derive a group key ; then, parties communicate online, with additional cryptographic operations, using the mpOTR protocol (multiparty OTR) that still enable deniability
- **Physical approaches** : physical constraints, such as verifying visually security code (like Whatsapp in two-parties messaging), may be a way to restrict malicious group members, since it forces to clearly know all the persons involved. This allows to derive strong security properties.
- **n -party DH** : if we want to keep the Diffie-Hellman idea and generalize it, this would be the method to consider : given n elements $\{g^{x_i}\}_{i=0}^n$ and a single x_i , derive a shared DH secret. As the basic DH, it is hard to compute without knowing one of the x_j . However, with large size groups, it quickly become expensive to calculate every value.
- **Tree-based group DH** : this kind of protocol create a binary balanced tree, where the leaves are associated with parties and contain a private DH key, such as (i) g^{xy} is the secret key of a parent whose children's secret keys are x and y ; and (ii) $g^{g^{xy}}$ is the public key. By computing recursively these secret keys up to the root, we obtain a tree key, used either directly to encrypt, or as an input for a KDF in order to obtain a message key.

The problem of most of these solutions are the lack of PCS (Post-Compromise Security), since they don't allow an updating of the keys.

A last example, which doesn't support keys updating, is the Whatsapp group messaging system, explained here [Whi16] :

Just before a member, let's say Charlie, joins a group, he applies the following steps :

1. Charlie generates a random Chain Key
2. He generates a random Signature Key pair
3. He combines both the Chain Key and the public key from the Signature Key pair into a Sender Key message

4. Finally, Charlie individually sends to each member its Sender Key, using the pairwise channels with the Signal Protocol

Then, for all subsequent messages sent to the group :

1. The sender derives a Message Key from the Chain Key, and updates the Chain Key, similarly to the symmetric ratchet in the Signal protocol
2. He encrypts the message with the AES256 algorithm
3. He signs the ciphertext using the private key of the Signature Key pair
4. And eventually sends to the server the message, which will transmit to the other members of the group

In order to be able to decrypt the received ciphertext, the other parties will have to update the Chain Key of the sender, in a ratchet way. By updating, the protocol provides Forward secrecy, and when a group member leaves, the other parties clear their Sender Key and start over.

Although this protocol seems efficient, it still has some drawbacks. The biggest one is the lost of the self-healing property : a sending key is directly linked to the following one, meaning that a leaked key allows an adversary to compute the next keys and thus directly threaten the security of the ciphertexts and impersonate the party whose key has leaked.

4.3 Asynchronous Ratchet Tree protocol

Explain the implication when an entity is leaving the group...

4.3.1 Introduction

As indicated at the beginning of the chapter, up to now, only two-parties communications were studied, and with the Signal protocol, one can get end-to-end encryption, with numerous security properties : privacy, forward secrecy, post-compromise secrecy, deniability, authentication, and non-repudiation.

But for group messaging, some of these properties are not there ; and even worse, for some system an adversary who compromised a single group member can read and write message with no limit. The major property missing is Post-Compromise Secrecy (PCS), and that is what we will try to get, in this chapter with the ART protocol, alongside the asynchronous aspect.

The ART protocol, or Asynchronous Ratchet Tree protocol [CGCG⁺18], was developed by Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner. The objective of such a protocol is to fix the absence of these two properties missing (PCS and asynchronous) for group messaging application. The protocol is based on the Signal Protocol and tree-based Diffie-Hellman key exchange.

The Asynchronous Ratchet Trees can be split in two important phases :

1. ART Construction

2. ART Update

In the first part, we describe how the initiator, or creator, of the group derives a tree-based group Diffie-Hellman, and then apply a Key Exchange Protocol (KEP) in order to send all the information necessary to the other member of the groups, such as the initial shared secret key.

In the second part, we detail the method allowing an ART group member to update their personal information and establish a new shared group key. It is with this update that we obtain PCS

The main cryptographic primitives that will be used are :

1. Diffie-Hellman group operations, using Curve25519
2. Symmetric encryption, using AES-256, with GCM mode
3. Key Derivation Function (KDF), using SHA256
4. Public signature schemes, using Curve25519

4.3.2 Notations

Before going more into details about both phases, let's mention some cryptographic indications and notations used here. DH groups :

Assumption

We use lowercase values k to represent DH secret keys and uppercase values $K = g^k$ for their corresponding public keys.

We assume that the Pseudo-Random Function – Oracle Diffie-Hellman problem (PRF-ODH) is hard : the probability to distinguish the tuple (g^x, g^y, g^{xy}) from (g^x, g^y, g^z) , with z randomly generated, is $\frac{1}{2} + \epsilon$, with $\epsilon > 0$ very small and bound.

DH groups

In this protocol, we work in a DH group G with a mapping $\iota(\cdot) : G \rightarrow \mathbb{Z} / |G|\mathbb{Z}$, from a group element to integers.

Signature

Two signature schemes are used : one to authenticate the initial group setup message, and a MAC to authenticate subsequent updates

Trees

We call a binary balanced tree as binary tree in which the depth of the two subtrees of every node never differ by more than 1, with a binary tree being a tree where a node has at most two children [LP07]. All nodes have a parent node and a sibling, except the root.

The path of a node is the set containing the node, its parent, the parent of this parent, and so on, until the root.

The copath of a node is a set of nodes which are the sibling of each element in the path from the node to the root (see Figure 4.1).

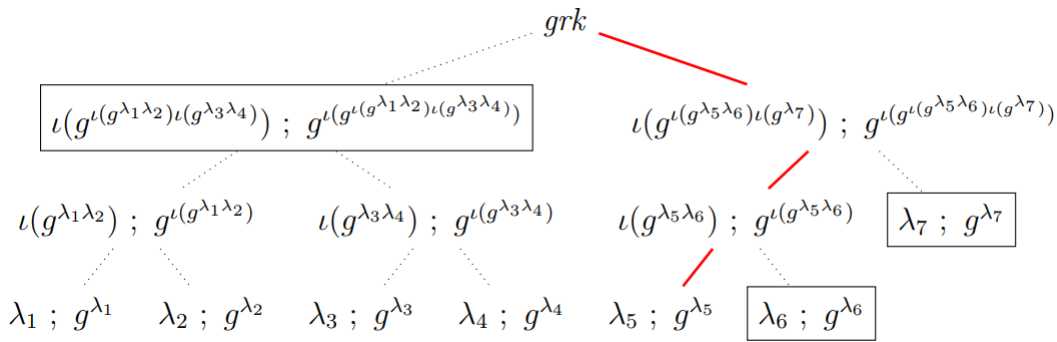


Figure 4.1: ART tree example : In red is the path, and the boxed nodes constitute the copath

Derived keys

ART contains four different kind of keys :

- Leaf keys λ_j : secret DH keys, included in each leaf
- Node keys nk : secret DH keys, included in each non-leaf nodes
- Tree keys tk : secret values derived at the root
- Stage keys sk : keys derived on a combination of the latest tk and the previous sk , with a hash chain

Here, the stage keys sk are similar to the root keys in a two-parties Signal protocol conversation, but we avoid saying root to avoid confusion with the root of the DH tree.

4.3.3 ART Construction

During the rest of this chapter, we consider that Alice is the party that creates the group, and that she wants to add n other members : Bob, Charlie, ..., until Zoe.

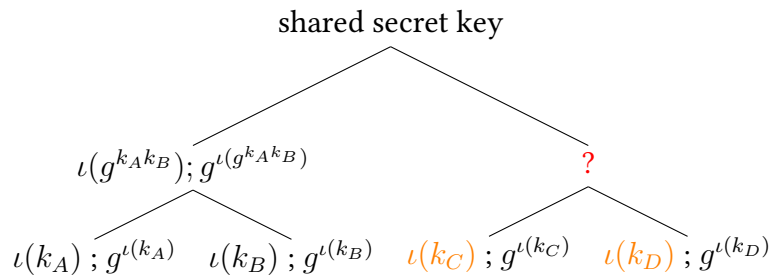


Figure 4.2: Tree-based Diffie-Hellman : Alice cannot compute the keys of the parent of C and D, since she doesn't know neither C's or D's private key, colored in **orange** (based on [CGCG⁺18], p.12)

We have compared the ART tree to the tree-based DH protocol, but in reality, to allow the creation of a group asynchronously, the creator (let's say Alice) would need the private DH keys of every other members, which is not possible since they are not online.

Indeed, in a four-group person, Alice cannot compute the parent node of C and D, because she would either need to know the public key of C and the private key of D (K_C and k_D) or the inverse (k_c and K_D). See Figure 4.2.

Instead, it will directly be Alice that will derive initial private keys for C and then for D, and for each member, she will send the information necessary allowing these members to also derive the secret key.

With these knowledges, Alice is now able to calculate the parent node keys, and thus the shared group key.

Let's show how it works.

Cryptographic primitives

The cryptographic primitives necessary in the ART protocol are :

- Key Encapsulation Mechanism (KEM) : used for Hybrid Public Key Encryption (HPKE) in group operations into the tree
- Authenticated Encryption with Associated Data (AEAD) : used for HPKE and message protection
- Hash function : used for HPKE as well
- Signature algorithm : used for message authentication

reference

The paper [BMO⁺19] proposes several possibilities of set of cryptographic primitives, or ciphersuites as it is called in it, with different possibilities.

The Table 4.1 lists all the combination presented.

Level of security	KEM	AEAD	Hash	Signature
128 bits	DH KEM with Curve25519	AES-128	SHA-256	Ed25519
128 bits	DH KEM with P256	AES-128	SHA-256	P256
128 bits	DH KEM with Curve25519	chacha20poly1305	SHA-256	P256
256 bits	DH KEM with Curve448	AES-256	SHA-512	Ed448
256 bits	DH KEM with P521	AES-256	SHA-512	P521
256 bits	DH KEM with Curve448	chacha20poly1305	SHA-512	Ed448

Table 4.1: Possibilities of Ciphersuite ([BMO⁺19], p.64)

Initialization

For our initial authenticated key exchange, we use the X3Dh algorithm ([CGCG⁺18], p.36).

Similarly to the Signal Protocol, a shared secret is calculated based on the information the members have published on a common, non-trusted server.

This PreKey Bundle is called a *KeyPackage*, and it specifies a ciphersuite that the client supports, and a pack of keys :

1. IK : identity key
2. $\{EK^i\}_{i=1}$: a set of n one-time ephemeral keys

This set of keys is almost the same as the one indicated in Section 2.3.1. Only the *SPK* (Signed PreKey) is missing, but it could easily be added.

Calculating the leaf keys

Let's say Alice wants to create a group with n members, i.e. herself and $n - 1$ other persons. After generating an ephemeral pair of keys suk_a and SUK_a (also called *setup key*), she requests to the server the set of public keys of each expected new members of the group.

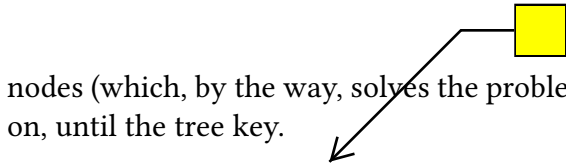
With Bob's public information, and with her secret keys, she derives the $n - 1$ leaf keys $\lambda_1, \lambda_2, \dots, \lambda_{n-1}$. To do so, Alice computes:

1. $DH_1 = DH(ik_A, EK_B)$
2. $DH_2 = DH(suk_A, IK_B)$
3. $DH_3 = DH(suk_A, EK_B)$

as in the Signal Protocol, and calculates a shared secret :

$$\lambda_i = SK_i = KDF(DH_1 || DH_2 || DH_3)$$

This shared secret key corresponds to the secret key of the leaf i . She thus needs to do so with the $n - 1$ members. Once all keys have been calculated, Alice is able to derive the keys of the parent



nodes (which, by the way, solves the problem in Figure 4.2), then to keys of parent's parents, and so on, until the tree key.

Alice can then calculates all the nodes keys and the group key (key of the root).

Once it's done, she can deletes every leaf's keys, since she doesn't need them anymore.

Sending the initial messages

After calculating all these values, Alice send the following information to each group member i :

1. the public prekeys EK_i (or just an index indicating the one used)
2. Alice's public identity key IK_A and her public key SUK_A corresponding to the private key SUK_A used by Alice
3. the tree T containing all the public keys
4. a signature of 1), 2) and 3), under her identity key

After receiving such a message and verifying the signature, each party will be able to reproduce the same computations Alice did, and derive the same leaf keys, and finally the tree key.

Hence, a tree has been constructed, with the same view from every member.

Calculating the node's keys

How a leaf can calculate the tree key, only knowing its pair of keys and the public keys of the nodes in its copath ?

As described earlier, a leaf owns a pair of keys : a public and a private one. The corresponding member, let's say Bob, also knows the public keys of all the other leaves. In particular, Bob knows the public key $SK_C = Y = g^y$ of its sibling Charlie (y being Charlie's private key). Then, with Bob's private key $sk_B = x$, Bob can calculate a shared DH value :

$$S = DH(x, Y) = Y^x = g^{xy} = g^z$$

with $z = xy$.

This value represents the new private key of the parent of Bob and Charlie. And the corresponding public key is :

$$g^{\iota(g^z)}$$

, with $\iota(\cdot)$ as defined in Section 4.3.1

Hence, Bob knows the pair of keys of its parent. Since he knows the public key of his parent's sibling as well, he can also calculate the key pair of his parent's parent, by computing the same operation

he did for his parent.

Iteratively, he is able to derive the tree key.

The result of an ART construction, with Alice the initiator and 3 other members is shown in Figure 4.3

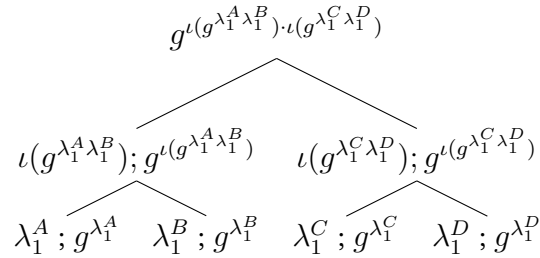


Figure 4.3: Tree-based Diffie-Hellman : Alice computes the tree key, since she knows the secret leaf keys of the other members

4.3.4 ART Update

With asynchronicity, the goal of this protocol is achieving PCS. To do so, any group member must be able to update stage keys, based both on state from previous stages and on newly exchanged messages.

Such a change could be provoked by one or a combination of these major operations :

- Adding a member
- Updating the secret leaf of a member
- Removing a member

Once one or several of these operations are done, a *Commit message* is broadcasted, to provide the group this new information

Adding a member

An Add proposal requests that a client can be added to the group. A new client is characterized by a KeyPackage, that they should have previously published onto the server.

The added member's position in the tree, indicated by an index, should be decided in order to keep the tree left-balanced as much as possible. If necessary, the tree can be expanded.

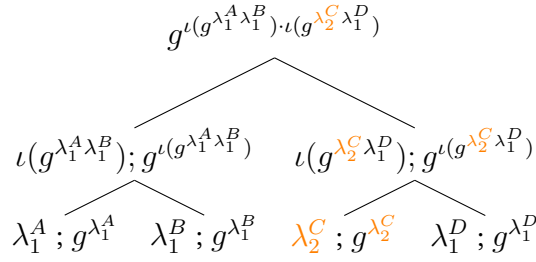


Figure 4.4: ART Update : Charlie changes his leaf key, from λ_1^C to λ_2^C

Updating a key

An Update proposal is similar to the previous one, with the difference that it is the sender's leaf KeyPackage which will be updated, through a new KeyPackage.

Once the leaf's key is updated, the corresponding member calculate the keys of the nodes in its path, and sends these new information to the other nodes in its copath.

If Charlie decides to update its leaf key, every other group member should be able to derive all the intermediate values in the new updated tree, by using only :

1. their view of the tree before the change
2. the list of updated public DH keys of nodes along the path from Charlie's leaf node to the root of the tree

For example, let's say that Charlie has decided to change his leaf key, from λ_C to λ'_C . This means that he can now derive all the values of the node in its path to the root. After having done it, he broadcasts to the group his public key with the new intermediate node he just computed. He also needs to authenticate the message with a MAC under a key derived from the previous stage key.

At the end, if everything was done correctly, every member of the group should have derived the same tree key, and thus the same stage key.

Moreover, to enforce Forward and Post-Compromise Secrecy, each member should periodically updates their leaf secret.

An example is given in Figure 4.4

Removing a member

A Remove proposal requests that a member at a given index in the tree should be removed from the group. The position in the tree is replaced with a blank node. The keys of the parent is then only calculated based on the sibling of the former member, meaning this node gets a new pair of key, and thus the tree key changes as well.

Commit

A *Commit message* is based on a collection of Proposals. This message instructs to all the members of the group to update their view of the tree, by applying the proposals and deriving the tree key, through the computation of the intermediate nodes. As long as a regular member does not receive any Commit message, its representation of the state of the tree should not change.

For simplicity, we consider that a Commit is done after each operation.

To perform an update for a path, i.e. a Commit, the "committer" sends to each member of the group these informations :

- The sender's public key
- Encrypted copies of the path secret of the node, i.e. the new keys of the nodes in the path of the sender.

4.3.5 Complementary information

The message keys used in the symmetric-key encryption algorithm (usually AES) are taken directly from the stage keys of each stages, instead of computing chain keys, like in the Double Ratchet algorithm. A signature of the ciphertext is also sent, signed by the send private identity key.

In the paper talking about the ART [CGCG⁺18], it is mentioned the possibility to only use the basic Diffie-Hellman algorithm, but we decided not to use it, since X3DH provides much more security.

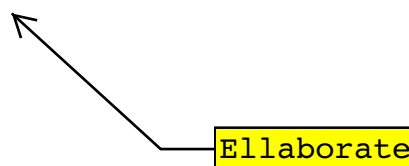
We could also talk about :

- Tree hash
- Nonce, for the random aspect

4.3.6 Quantum risks

As explained, the Asynchronous Ratchet Tree protocol consist of two parts : ART construction and ART update. The cryptographic primitives used in the protocol are : Key Exchange, Signature schemes and symmetric encryption. The algorithms used in a primitive are cited in the previous section.

In the same way as in the Signal Protocol, only ECDH Curve25519 is directly threatened by quantum computers, while SHA and AES only need to adjust their parameters.



(SIKE)

Chapter 5

Supersingular Isogeny Key Exchange

Among the three Post-Quantum protocols studied as replacement to the current ones, isogeny-based cryptography is probably the most recent. Its security relies on the fact that, given two elliptic curves, it is hard to find an isogeny to go from one curve to the other.

And quantum computers are not able to solve this problem in subexponential-time, making this protocol interesting for a possible Post-quantum Signal protocol, but also a Post-quantum group protocol.

Shor

allegedly/presumably

5.1 Introduction

With ~~Grover~~'s algorithm, the use of Diffie-Hellman to create a shared secret became obsolete in quantum systems. That is why it could be swapped with a protocol with a similar idea, based on supersingular isogeny, meaning working with a specific kind of elliptic curve.

The idea of using isogeny in cryptography first appears with Couveignes, in 1997 [Cou06]. In 2010, Jao, Childs and Soukharev demonstrated that it was possible to break this scheme with quantum algorithms [CJS14]. The main problem of Couveignes' protocol was the group containing the isogeny is commutative, hence making the attack much easier.

This is why Jao and De Feo [JDF11] decided to use supersingular elliptic curves, whose ring of endomorphisms is non-commutative. This led to the key-agreement scheme, called Supersingular Isogeny Diffie-Hellman. The current state-of-the-art implementation is SIKE, created in 2017, and was submitted to the NIST competition, and is now at the Round 3. Most of the following explanation about this protocol comes from the corresponding paper [ACC⁺17]

(SIDH)

5.2 Elliptic curve

Let's start with the general definition of an elliptic curve.

An elliptic curve E over a field K is defined as the equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, a_2, a_3, a_4, a_6 \in K$.

There exist several specific forms of elliptic curve, such as :

- Weierstrass form : the most common, often used in papers talking about Elliptic Curve Theory. Its equation is

$$E : y^2 = x^3 + Ax + B$$

- Edwards form : a recent form studied for Elliptic Curve Diffie-Hellman cryptography, used to simplify formulas in the theory of elliptic curves and functions [EH18]. Its equation is

$$x^2 + y^2 = 1 + dx^2y^2$$

- Montgomery form : this is the one we are going to use through the rest of the chapter. It is the most used in Isogeny-based cryptography. Its equation is

$$By^2 = x^3 + Ax^2 + x$$

Unless specific indication, we consider $B = 1$.

A couple of examples are given in Figure 5.1.

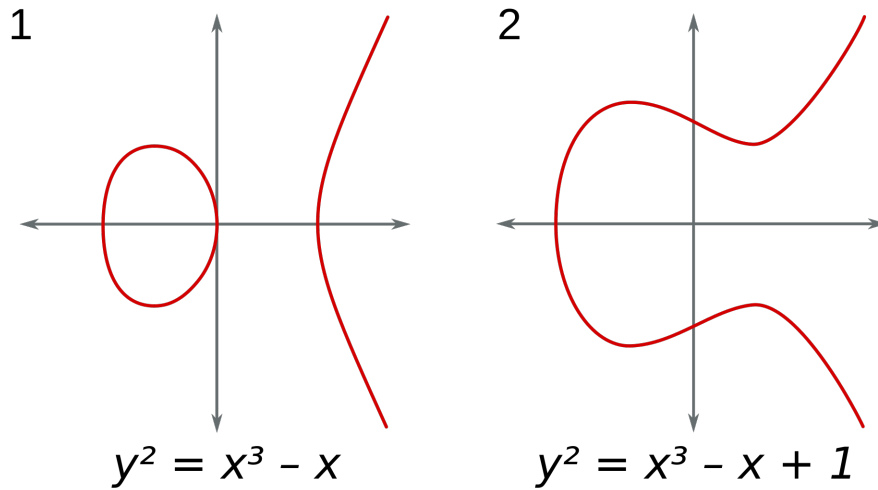


Figure 5.1: Two examples of elliptic curves, in the Weierstrass form

5.2.1 Fundamental theory

Before going more in detail about Elliptic Curve Theory, it could be good to first express mathematic basics, necessary for the rest of the chapter.

Finite field

A field, often denoted K , is a set associated with two operations $(+, \cdot)$, called addition and multiplication, respecting the following conditions, $\forall x, y, z \in K$:

Addition :

1. Commutativity : $x + y = y + x$
2. Associativity : $(x + y) + z = x + (y + z)$
3. Existence of an additive identity : $\exists 0 \in K$, called zero, such that $x + 0 = x$
4. Existence of additive inverses : $\forall x, \exists -x \in K$ such that $x + (-x) = 0$

Multiplication :

1. Commutativity : $x \cdot y = y \cdot x$
2. Associativity : $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
3. Distributivity : $(x + y) \cdot z = x \cdot z + y \cdot z$
4. Existence of a multiplicative identity : $\exists 1 \in K$, such that $x \cdot 1 = x$
5. Existence of multiplicative inverses : if $x \neq 0$, then $\exists x^{-1} \in K$ such that $x \cdot x^{-1} = 1$

For more details about fields, and field theory in general, see [Mar03]

The finite field \mathbb{F}_p

The elements of the finite field \mathbb{F}_p are all the positive integer, up to p :

$$\mathbb{F}_p = \{0, 1, \dots, p-1\}$$

All the field operations $(+ \text{ and } \cdot)$ are defined modulo p , meaning $a + b \equiv r[p]$, meaning r is the remainder of $a + b$ divided by p . Same goes for the multiplication.

Moreover, all elements in \mathbb{F}_p have additive and multiplicative inverse in \mathbb{F}_p :

- Addition : finding the additive inverse of an element $a \in \mathbb{F}_p$ is easy : one just need to add p to $-a$
- Multiplication : finding the multiplicative inverse of a is harder with modulo. The default algorithm used is the Extended Euclidean algorithm, but with high numbers, it can quickly become expensive. That is why, in a lot of optimized algorithms, programmers tend to avoid computing them, with some equivalent calculations.

be more precise, indicate/
summarize these alternatives

Finite field \mathbb{F}_{p^2}

The elements used in SIDH come from the finite field \mathbb{F}_{p^2} , which is an extension¹ of \mathbb{F}_p .

An element of the finite field \mathbb{F}_{p^2} is written in the following form :

$$c = c_0 + c_1 \cdot i \quad \text{where } c_0, c_i \in \mathbb{F}_p \text{ and } i^2 = -1$$

This writing can remind the complex numbers. Indeed, we have $\mathbb{F}_{p^2} = \mathbb{F}_p(i) := \{a + i \cdot b \mid a, b \in \mathbb{F}_p\}$, similarly to the complex set : $\mathbb{C} = \mathbb{R}(i)$. Moreover, the arithmetic operations (+, ·) are defined similarly to the complex operations (for details, see [Byl90]).

5.2.2 Montgomery curve

We have defined the general form of an elliptic curve (EC) in the introduction of this section. Nevertheless, we use a specific form of elliptic curve in SIKE : the Montgomery curves.

Let $A, B \in \mathbb{F}_q$ with $B(A^2 - 4) \neq 0$, with q a power of a prime number ($q = p^n, n \in \mathbb{N}$).

A *Montgomery curve* $E_{A,B}$, defined over \mathbb{F}_q , often denoted $E_{A,B}/\mathbb{F}_q$, is the set of point $P = (x, y)$ satisfying the equation

$$By^2 = x^3 + Ax^2 + x \tag{5.1}$$

with an extra point \mathcal{O} , called the point at infinity, or natural point.

To be more general, we denote $E(K)$ as the set of points in curve E over the field K

$$E(K) = \{P = (x_P, y_P) \in K^2 \mid B \cdot y_P^2 = x_P^3 + A \cdot x_P^2 + x_P\}$$

As indicated at the beginning of this section, we set $B = 1$ for the rest of this paper.

We will explain later why we prefer Montgomery-form elliptic curves.

5.2.3 Group law

We are now presenting the arithmetic operations applied in Montgomery curves, such as Point addition, Point Doubling, Point Tripling and Multiplication.

Addition

In any elliptic curve, starting with two points, it is possible to create a new one. It is even possible to do this with only one point.

¹A field H is an extension of a field G if $G \subset H$ and H has the same field operations than G

We start with two points, let's say :

$$P = (x_P, y_P), \quad Q = (x_Q, x_Q)$$

on an elliptic curve $E : By^2 = x^3 + Ax^2 + x$. For now, we assume that $P \neq Q$ and $x_P \neq x_Q$ in particular.

To find a new point R based on P and Q , we do as follow :

1. Draw the line L going through P and Q
2. Set R' the intersection between the line L and the curve E
3. Reflect R' across the x -axis, changing the y -coordinate, to obtain R

We thus write for now $P \oplus Q = R$.

An visual example is given Figure 5.2.

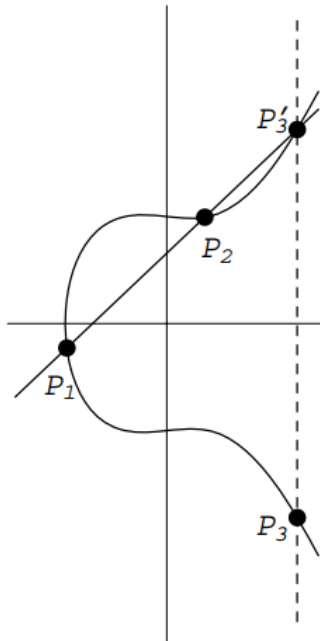


Figure 5.2: Adding points on an Elliptic curve

So, given the previous steps, how do one compute a points addition ?

To explain the process, we take over these steps, and "translate" them in mathematics calculations.

1. L is the line going through P and Q . Its slope is then :

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

The equation of L is then given by the equation :

$$y = \lambda \cdot (x - x_P) + y_P$$

which is also equivalent to $y = \lambda \cdot (x - x_Q) + y_Q$

2. To find the intersection with E , we substitute y in 5.1, and we get :

$$B(\lambda \cdot (x - x_P) + y_P)^2 = x^3 + Ax^2 + x$$

If we develop the left side and move everything to the right side, we can rearranged (with $B = 1$) the equation to :

$$0 = x^3 + (\lambda^2 - A) \cdot x^2 + \dots$$

The three roots of this equation of degree 3 correspond to the three points of intersection between L and E . Usually, it is not easy to solve a cubic equation, but here, we already know two of the roots, namely x_P and x_Q . Moreover, an equation of degree 3 which has 3 distinct solutions r, s, t , can be factorized :

$$x^3 + ax^2 + bx + c = (x - r)(x - s)(x - t) = x^3 - (r + s + t)x^2 + \dots$$

This means :

$$r + s + t = -a$$

If two roots are known, such as r and s , we can recover the third :

$$t = -a - r - s$$

In our case, we have : $r = x_P, s = x_Q, -a = (\lambda^2 - A)$, and $t = x_{R'}$ the point we want to find. Thus, we obtain :

$$x_{R'} = \lambda^2 - A - x_P - x_Q$$

and

$$y_{R'} = \lambda \cdot (x_{R'} - x_P) + y_P$$

3. Eventually, we just have to reflect across the x -axis the point R' in order to obtain R , which is made by changing the sign of $y_{R'}$, which gives us :

$$x_R = \lambda^2 - A - (x_P + x_Q), \quad y_R = \lambda \cdot (x_P - x_R) - y_P$$

To sum up, we can rewrite the addition map as

$$\oplus : E_A \times E_A \rightarrow E_A$$

$$(x_P, y_P) \oplus (x_Q, y_Q) \mapsto (\lambda^2 - A - (x_P + x_Q) \quad ; \quad \lambda^2 \cdot (x_P - x_R) - y_P) \quad (5.2)$$

where

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

In the situation where $x_P = x_Q$ but $y_P \neq y_Q$, it means that the line L going through P and Q is vertical. We then say that the line intersects E in ∞ .

Therefore, in this case :

$$P \oplus Q = \infty$$

Point doubling

The case where $P = Q$ means that we are doing $P \oplus Q = [2]P$, which is called *Point doubling*.

When two points on a curve are really close one to another, the line through them approximates a tangent line. Therefore, when the two points coincide, we consider that the line L is the tangent line to the point P .

To find the slope at this point, we apply implicit differentiation on equation (5.1) :

$$2y \cdot \frac{dy}{dx} = 3x^2 + 2Ax + 1$$

$$\Rightarrow \lambda = \frac{dy}{dx} = \frac{3x_P^2 + 2Ax_P + 1}{y_P}$$

As before, the equation of the line L is

$$y = \lambda(x - x_P) + y_P$$

We obtain the third degree equation

$$0 = x^3 - (\lambda^2 - A)x^2 + \dots$$

This time, we know only one root, namely x_P , but it is a double root. Thus, we can proceed as previous, and we obtain :

$$x_R = \lambda^2 - A - 2x_P, \quad y_R = \lambda \cdot (x_P - x_R) - y_P$$

To sum up, we can rewrite the doubling map as

$$[2] : \begin{array}{ccc} E_A & \rightarrow & E_A \\ (x_P, y_P) & \mapsto & (\lambda^2 - A - 2x_P, \lambda^2 \cdot (x_P - x_R) - y_P) \end{array} \quad (5.3)$$

with

$$\lambda = \frac{3x_P^2 + 2Ax_P + 1}{y_P}$$

Point tripling

The last operation is point tripling, meaning we want to compute $P \oplus P \oplus P = [3]P$.

This time, we are not going to explain the calculations made, but just show the result :

$$x_{[3]P} = \frac{(x_P^4 - 4Ax_P - 6x_P^2 - 3)^2 \cdot x_P}{(4Ax_P^3 + 3x_P^4 + 6x_P^2 - 1)^2}$$

and

$$y_{[3]P} = y_P \cdot \frac{(x_P^4 - 4Ax_P - 6x_P^2 - 3)(x_P^8 + 4A(x_P^7 + x_P) + 28(Ax_P^6 + x_P^5 + Ax_P^3 + x_P^2) + (16A^2 + 6)x_P^4 + 1)}{(4Ax_P^3 + 3x_P^4 + 6x_P^2 - 1)^3} \quad (5.4)$$

For more details about these computations, see [DIK06].

Nevertheless, one can remarks that $x_{[3]P}$ only depends on x_P and A . We could have decided to make it depends on y_P as well, like in Point Addition and Point Doubling, but the raison behind this choice is that it is possible to optimize the computations of the elliptic curves operations. We explain more in detail in Section ??.

Multiplication

Finally, if one wants to compute $[n]P$, with n higher than 3, they just need to decompose n in power of 2, and then apply the square computation.

For example, to calculate $[11]P$, we first compute

$$P, \quad [2]P, \quad [4]P = [2]([2]P) \quad \text{and} \quad [8]P = [2]([4]P)$$

And then we apply Point addition :

$$[11]P = P \oplus [2]P \oplus [8]P$$

which is much faster than doing $P \oplus P \oplus \dots \oplus P$ eleven times.

This is the same concept applied in Elliptic Curve Diffie-Hellman (ECDH).

5.3 Isogeny graph

The central concept in SIDH is Isogeny graph. It is this kind of graph that is used between two parties to exchange keys.

An *isogeny graph* is a graph, whose nodes consist of all elliptic curves in \mathbb{F}_{p^2} , up to \mathbb{F}_p -isomorphism, represented by their j -invariants, and the edges represent isogenies between these curves.

A lot of concept are new here, and we are going to introduce all of them in the following parts.

5.3.1 j -invariant



As indicated, SIKE works in the quadratic extensions \mathbb{F}_{p^2} . Such a group contains p^2 elements (since $\mathbb{F}_{p^2} = \{a + ib | a, b \in \mathbb{F}_p\}$), but we are only interested in a subset of it, of size $\lfloor \frac{p}{12} \rfloor + r$, where $r \in \{0, 1, 2\}$. The value of r depends on $p \bmod 12$. For more information, see [Sil09] (Theorem V.4.1 (c)).

This subset is the set of supersingular j -invariants in \mathbb{F}_{p^2} .

Supersingular elliptic curve

Characteristic : the characteristic of a field F is the smallest integer n such as $n \cdot x = 0, \forall x \in F$, \cdot being the multiplication operation of the field.

Based on this definition, the characteristic of an elliptic curve E over a field K is simply the characteristic of K .

The set of all the elliptic curves can be separated into two : the subset of the *supersingular* EC, and the set of the *ordinary* EC. This classification depends on the characteristic of the elliptic curve ([Gal01]).

We won't explain how exactly we determine the category of an EC, but to sum up, the supersingular case offers a couple of advantages :

1. an efficient method was set up to construct and instantiate them easily
2. and must of all, attacks against this kind of curve have exponential complexity

j -invariant

As we have said, in the field \mathbb{F}_{p^2} , there exist p^2 elliptic curves. In practice, p can sometimes measures hundreds of bites, meaning that $|\mathbb{F}_{p^2}|$ would be clearly too large to consider in a single graph².

Instead, we are going to regroup the supersingular elliptic curves together into groups. But to do so, we need a criteria to compare EC.

² $|G|$ gives the order (or cardinality) of the group G

underline/bold

This is where the j -invariant function arrives.

Given an elliptic curve E_a in the Montgomery form

$$E_a : y^2 + ax^2 + x$$

we define the j -invariant of E_a as

$$j(E_a) = \frac{256(a^2 - 3)^3}{a^2 - 4} \in \mathbb{F}_{p^2}$$

With this application, we are going to group together the isomorphic curves.

Isomorphism : an isomorphism from G to H is a bijection (ie. injective and surjective application) $\phi : G \rightarrow H$ with the property that $\phi(ab) = \phi(a)\phi(b)$, $\forall a, b \in G$. This means that ϕ preserves the group (or field) operations [Vin03].

Therefore, G and H are said to be isomorphic if there exists an isomorphism going from one group to the other.

Maybe the most important thing to say about j -invariant is the following theorem.

Theorem 1 *Two curves are isomorphic over a field K if and only if they have the same j -invariant.*

Proof : See [DF17], Section 2.

This is why, in the introduction of the section, we said that a node contains all the curves up to \mathbb{F}_{p^2} -isomorphism : we said we only work with the supersingular EC, and among this subset, we regroup the curves into the same group if they have the same j -invariant, and therefore in the same node in the graph.

Let's illustrate this part with an example.

Let $p := 431$. This means that we have $\lfloor \frac{p}{12} \rfloor + 2 = 37$ different j -invariants in \mathbb{F}_{p^2} (we say there are 37 supersingular j -invariants).

They are represented in Figure 5.3

If we take $a_1 = 161 + 208i$ and $a_2 = 162 + 172i$, then we obtain :

$$j(E_{a_1}) = 304 + 364i = j(E_{a_2})$$

meaning that $E_{a_1} : y^2 + x^3 + (161 + 208i)x^2 + x$ and $E_{a_2} : y^2 + x^3 + (162 + 172i)x^2 + x$ are isomorphic, and both correspond to the node $304 + 364i$ in Figure 5.3.

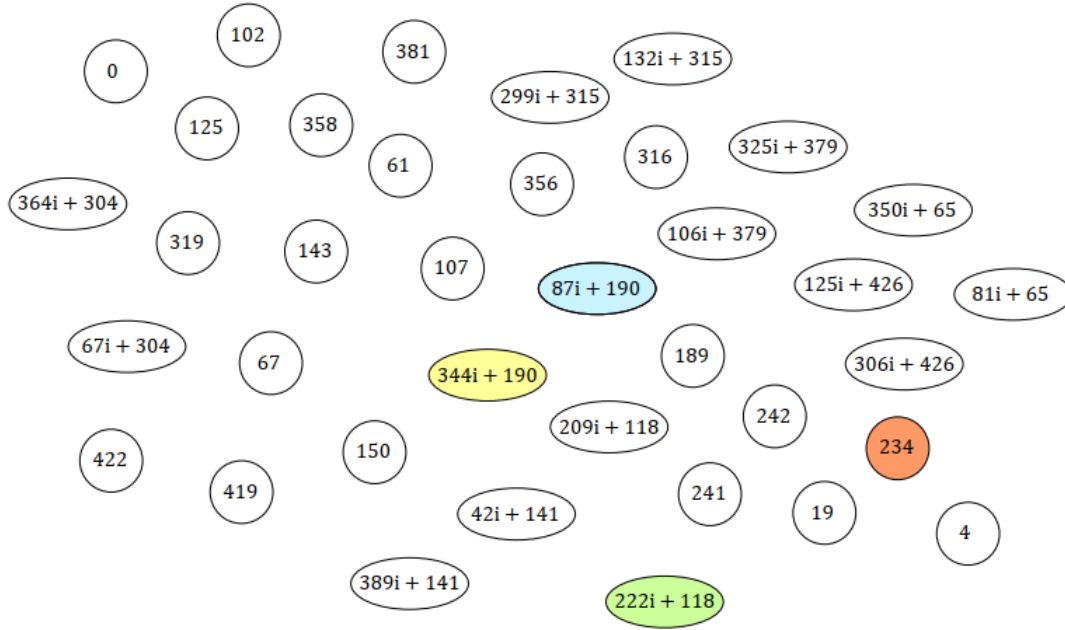


Figure 5.3: Set of the 37 supersingular j -invariants in \mathbb{F}_{431^2} [Cos19]

5.3.2 SIDH in a nutshell

Before continuing talking about the isogeny graph and its edges, the isogenies, we could first sum up the Supersingular Isogeny Diffie-Hellman algorithm with our current knowledge.

As its name indicate, there is a link between SIDH and the traditional Diffie-Hellman protocol.

Let's just remind Diffie-Hellman :

In a cyclic group G , let g be a public parameter, accessible by anyone. Let Alice and Bob be the two parties implied in the protocol, each possessing a secret key, a and b respectively. First, both party compute g^{sk_a} , g to the power of their secret key. Alice obtains g^a , while Bob gets g^b .

They exchange their value, and apply again their secret key. The secret key obtained is $g^{ab} = g^{ba}$.

SIDH is, in a way, similar. Referring back to Figure 5.3 :

- Alice and Bob start both from the same (blue) node, corresponding to the j -invariant $190 + 87i$ (g in Diffie-Hellman)
- Alice chooses a secret integer that, after some calculations, make her move from a node to another (starting with the blue one), until she reaches the (green) node, with j -invariant equal to $118 + 222i$ (g^a in DH)
- She then sends to Bob this value, that corresponds to her public key

- Similarly, Bob also chooses a secret value, making him move until the yellow node, of j -invariance $190 + 344i$ (g^b in DH **orange**)
- He sends this value to Alice, his public key
- With her secret key and Bob's public key, Alice can perform another sequence of moves, to end up at the (red) j -invariant 234. This final value is the secret shared secret, g^{ab} , and Bob does the same, with his private key and Alice's public key.

To be accurate, the public keys contain more information necessary to ensure that Alice and Bob land at the same node, but we will explain it later.

Each party is able to move from one node to another, ie. between j -invariants, by using the edges which are isogenies.

5.3.3 Isogeny

Presentation of Isogenies

Now, let's talk about the vertex of the graph, namely the isogenies.

Isogeny : Let E_1 and E_2 be elliptic curves. An *isogeny* from E_1 to E_2 is a morphism

$$\phi : E_1 \rightarrow E_2$$

satisfying $\phi(\mathcal{O}) = \mathcal{O}$.³ ([Sil09])

An isogeny is not necessary invertible.

Similarly to the relation between isomorphism and isomorphic curves, two curves are said to be *isogenous* if there exists an isogeny going from one to the other.

We have already seen two examples of isogeny : the *multiplication-by-2* (or Point Doubling) and the *multiplication-by-3* (or Point Tripling). We also saw the translation as a first example, much earlier, in Figure 3.3.

The isogenies going from a curve to itself are called *endomorphism*. The basic example of it is the *multiplication-by- m* , as seen earlier

$$[m] : P \mapsto [m]P = \underbrace{P \oplus P \oplus \dots \oplus P}_{m \text{ times}}$$

In most examples cited before, we had endomorphisms. But in general, an isogeny is a map

$$\phi : E \rightarrow E'$$

³To be accurate, we should write $\phi(\mathcal{O}_{E_1}) = \mathcal{O}_{E_2}$ since the neutral element in each elliptic curve may be different. But in the Montgomery curves, we say that in both case, it corresponds to the point at infinity

from one elliptic curve to another.

A question is still unanswered : how does one move from an elliptic curve to another one, ie. how is computed an isogeny ?

Finding the Isogenies

An isogeny $\phi : E_1 \rightarrow E_2$ can be expressed with two rational functions f and g over \mathbb{F}_p , such that

$$\phi((x, y)) = (f(x), y \cdot g(x))$$

Since $f(x)$ is rational, it can be written as $f(x) = \frac{q(x)}{r(x)}$, with $q(x), r(x) \in \mathbb{F}_p[x]$ with no common factor. Same thing goes for $g(x)$.

We define the degree of ϕ as

$$\deg(\phi) = \max\{\deg(p), \deg(q)\}$$

Thus, a ℓ -isogeny is simply an isogeny ϕ such as $\deg(\phi) = \ell$.

We also define the kernel of the same isogeny as

$$\ker(\phi) = \{P \in E_1 : \phi(P) = \mathcal{O} \in E_2\}$$

where \mathcal{O} is the neutral element.

For isogeny computations, Vélu's formula [Vél71] is used most of the time. However, this formula was originally computed for Weierstrass-form EC ; if the elliptic curve is of another form, this formula may not preserve its form.

Nevertheless, in the original SIDH paper, De Feo, Jao and Plût applied Vélu's formula and composed with the appropriate isomorphisms to return to Montgomery form. The problem is that this approach is very expensive for producing 2-isogenies, and even worse for 3-isogenies.

Instead, Renes [Ren18] adapted a Theorem from [[CH17], Theorem 1] for the Montgomery curves.

Theorem 2 *Let K be a field, with $a \in K$ such that $a^2 \neq 4$, and $E/K : y^2 = x^3 + ax^2 + x$ a Montgomery curve. Let $G \subset K$ be a finite subgroup such that $(0, 0) \notin G$ and let ϕ be an isogeny such that $\ker(\phi) = G$.*

Then, there exists a curve

$$E/G : y^2 = x^3 + Ax^2 + x$$

such that

$$\phi : \begin{array}{ccc} E & \rightarrow & E/G \\ (x, y) & \mapsto & (f(x), c_0 y \cdot f'(x)) \end{array}$$

where

$$f(x) = x \cdot \prod_{Q \in G^*} \frac{x \cdot x_Q - 1}{x - x_Q}$$

Moreover, writing

$$\pi = \prod_{Q \in G^*} x_Q \quad \text{and} \quad \sigma = \sum_{Q \in G^*} \left(x_Q - \frac{1}{x_Q} \right)$$

we have $A = \pi(a - 3\sigma)$ and $c_0^2 = \pi$

Proof: See [Ren18], Section 4.1 .

2- and 3-isogenies

We need to change a last thing to the previous Theorem to really be able to apply in the key exchange protocol in SIKE.

Indeed, SIKE prefers simply using 2-isogenies and 3-isogenies, easier to compute, and also giving us what we want : a way to move between the j -invariant nodes.

These choices of the two smallest primes currently give the most efficient instantiation of SIDH.

Moreover, when we take $\ell \neq p$, we get that the number of edges of each node in the ℓ -isogeny graph is $\ell + 1$ ([dF19], Corollary 29).

For Alice, we usually take $\ell = 2$, meaning that every node in Alice's graph will be of degree⁴ 3, and for Bob, we take $\ell = 3$, meaning his 3-isogeny graph will be of degree 4. Nevertheless, the nodes are obviously the same. Otherwise, it would be impossible to obtain a shared secret.

A consequence of Theorem 2 is that we obtain a formula for 2-isogenies for points other than $(0, 0)$.

Corollary 2.1 *Let $E/K : y^2 = x^3 + ax^2 + x$ be a Montgomery curve over a field K . Let $P \in E(K)$ such that $P \neq (0, 0)$ and $[2]P = \mathcal{O}_E$ ⁵.*

Then

$$\begin{aligned} \phi : E &\rightarrow \tilde{E}/K : y^2 = x^3 + Ax^2 + x \\ (x, y) &\mapsto (f(x), yf'(x)) \end{aligned}$$

with

$$A = 2 \cdot (1 - 2x_P^2)$$

is a 2-isogeny with $\ker(\phi) = \langle P \rangle$, where

$$f(x) = x \cdot \frac{x \cdot x_P - 1}{x - x_P}$$

Proof: See [Ren18], Section 4.2 .

Remark : For the y -coordinate, we thus obtain

$$y \cdot f'(x) = y \cdot \frac{x^2 x_P - 2x x_P^2 + x_P}{(x - x_P)^2}$$

⁴a graph is of degree n if all the vertices are of degree n , ie. n edges are connected to each vertice

⁵This means that P is of order 2, or $\text{char}(P) = 2$

Similarly, here is what we get for 3-isogenies.

Let $(x_P, y_P) \in E_a : y^2 = x^3 + ax^2 + x$ be a point of order 3 and let $\phi_3 : E_a \rightarrow E_A$ be the unique (up to isomorphism) 3-isogeny with kernel $\langle P \rangle$.

Then, E_A can be computed as

$$A = (ax_P - 6x_P^3 + 6)x_P$$

We then have :

$$\phi(x, y) = \left(\frac{x(xx_P - 1)^2}{(x - x_P)^2} \quad , \quad y \cdot \frac{(xx_P - 1)(x^2x_P - 3xx_P + x + 1)}{(x - x_P)^3} \right)$$

We have now the formulas to compute the 2-isogenies and 3-isogenies, based on kernels. This means that we need a last thing : kernels.

Finding kernels

We have defined earlier the degree of an ℓ -isogeny as the maximal degree of the rational function. But based on [Sil09], Theorem III.4.10, it is also the number of elements in its kernel :

$$\deg(\phi) = \ell = |\ker(\phi)|$$

This means that the cardinality of the kernel of an 2-isogeny is 2. Moreover, we know that the neutral element \mathcal{O} is always included in the kernel of an isogeny, since we said in the definition of the isogeny $\phi(\mathcal{O}) = \mathcal{O}$

$$\mathcal{O} \in \ker(\phi)$$

Therefore, there is just another point to find to complete the kernel.

Let's go back to the isogeny *multiplication-by-2*, that we also called *point doubling* :

For a fixed Montgomery curve $E_a : y^2 = x^3 + ax^2 + x$, the isogeny is (for the x -coordinate)

$$\begin{aligned} [2] : E_a &\rightarrow E_a \\ x &\mapsto \frac{(x^2-1)^2}{4x(x^2+ax+1)} \end{aligned}$$

We observe that the doubling map has a denominator that might be equal to 0, meaning it would create "exceptional points".

To find them, we need to find the zeros of the denominator.

Since $4x(x^2 + ax + 1) = 4y^2$, we look at $4y^2 = 0 \Leftrightarrow y = 0$.

Then, the three points are $(0, 0)$, $(\alpha, 0)$ and $(1/\alpha, 0)$, where $\alpha^2 + a\alpha + 1 = 0$.
The last point comes from the second :

$$\alpha^2 + a\alpha + 1 = 0 \Leftrightarrow \alpha^2(1 + a/\alpha + 1/\alpha^2) = 0 \Leftrightarrow 1 + a \cdot \frac{1}{\alpha} + \left(\frac{1}{\alpha}\right)^2 = 0$$

These three points, of order 2 on E_a , with the neutral element \mathcal{O} form the entire kernel of the doubling map.

And from this, we can create 3 subgroups :

$$\{\mathcal{O}, (0, 0)\}, \quad \{\mathcal{O}, (\alpha, 0)\} \quad \text{and} \quad \{\mathcal{O}, (1/\alpha, 0)\}$$

Each of these subgroups is used to compute a 2-isogeny, meaning we obtain three 2-isogenies, as predicted earlier, when we said that in a 2-isogeny graph, each node would have three edges connected to it.

For the 3-isogenies, the idea is the same.

We take back to the isogeny *multiplication-by-3*. For a fixed Montgomery curve $E_a : y^2 = x^3 + ax^2 + x$, the isogeny is (for the x -coordinate)

$$\begin{aligned} [3] : E_a &\rightarrow E_a \\ x &\mapsto \frac{x(x^4 - 6x^2 - 4ax^3 - 3)^2}{4x(3x^4 + 4ax^3 + 6x^2 - 1)^2} \end{aligned}$$

Again, the denominator gives us four exceptional points. Let's suppose its four roots are β, δ, ψ and τ ; each of these values correspond to the x -coordinate of points of order 3 in E_a , but this time, there are two non-zero y -coordinates for each x .

Together with \mathcal{O} , we obtain nine points in the kernel of $[3]$:

$$\text{ker}([3]) = \{\mathcal{O}, (\beta, y_1), (\beta, -y_1), (\delta, y_2), (\delta, -y_2), (\psi, y_3), (\psi, -y_3), (\tau, y_4), (\tau, -y_4)\}$$

Eventually, we can form four subgroups, of size 3, that will give use the four 3-isogenies.

$$\{\mathcal{O}, (\beta, y_1), (\beta, -y_1)\}$$

$$\{\mathcal{O}, (\delta, y_2), (\delta, -y_2)\}$$

$$\{\mathcal{O}, (\psi, y_3), (\psi, -y_3)\}$$

$$\{\mathcal{O}, (\tau, y_4), (\tau, -y_4)\}$$

5.3.4 Example

Let's take back our example, with $p = 431$, giving us 37 j -invariants, shown in Figure 5.3.

As said before, both Alice and Bob have their own graph. For Alice, it will be a 2-isogeny graph, whereas for Bob, it will be 3-isogeny graph. But currently, there are only the nodes. And we are going to construct the vertex.

Alice starts from the curve $E_a : y^2 = x^3 + (161 + 208i)x^2 + x$, with $j(E_a) = 304 + 364i$. So, we are going to find the vertex coming from this node. With the 2-isogeny equation in (...), we find $E_{a'} : y^2 = x^3 + (423 + 102i)x^2 + x$ with $j(E_{a'}) = 190 + 364i$; the kernel was generated by $(\alpha, 0) \in E_a$ with $\alpha = 68 + 350i$.

Therefore, we can draw an edge between these two j -invariants.

Furthermore, there exist two other kernels : one generated by $(1/\alpha, 0)$ and the other by $(0, 0)$. These produce two additional edges, one connecting $j = 304 + 364i$ to $j = 319$ and one connecting $j = 304 + 364i$ to $j = 67$, respectively.

Remark : We cannot use the same formula for $(0, 0)$ than for the two other points, because we would divide by 0. The equation is thus different, and much more complexe. For more information, see [dF19].

By doing the same operations for the other j -invariants, we obtain all the vertex, depicted in Figure 5.4⁶.

The 3-isogeny graph is similarly constructed, but this time, there are four outgoing edges for each j -invariant⁷

With the same curve E_a as before, and by using the point $(\beta, \gamma) = (56 + 321i, 174 + 303i)$ of order 3 on E_a , we find the new curve $E_{a'} : y^2 = x^3 + 415x^2 + x$, giving $j(E_{a'}) = 189 = j_1$.

And with the three other points, we obtain three curves, giving us the j -invariant $j_2 = 19$, $j_3 = 141 + 42i$ and $j_4 = 379 + 106i$.

The final result is shown in Figure 5.5

5.4 SIKE protocol

Now that we have seen every theoretical aspects concerning SIKE, it's time to actually present the protocol itself.

5.4.1 Protocol's main steps

The SIKE protocol is composed of X steps :

⁶We remark that for $j \in \{0, 4, 242\}$, there are only 2 edges. These are exceptional points, and the reason of this difference is explained in [Cos19], Section 6

⁷Again, there is a small number of exceptions, for $j \in \{0, 4, 125, 242\}$

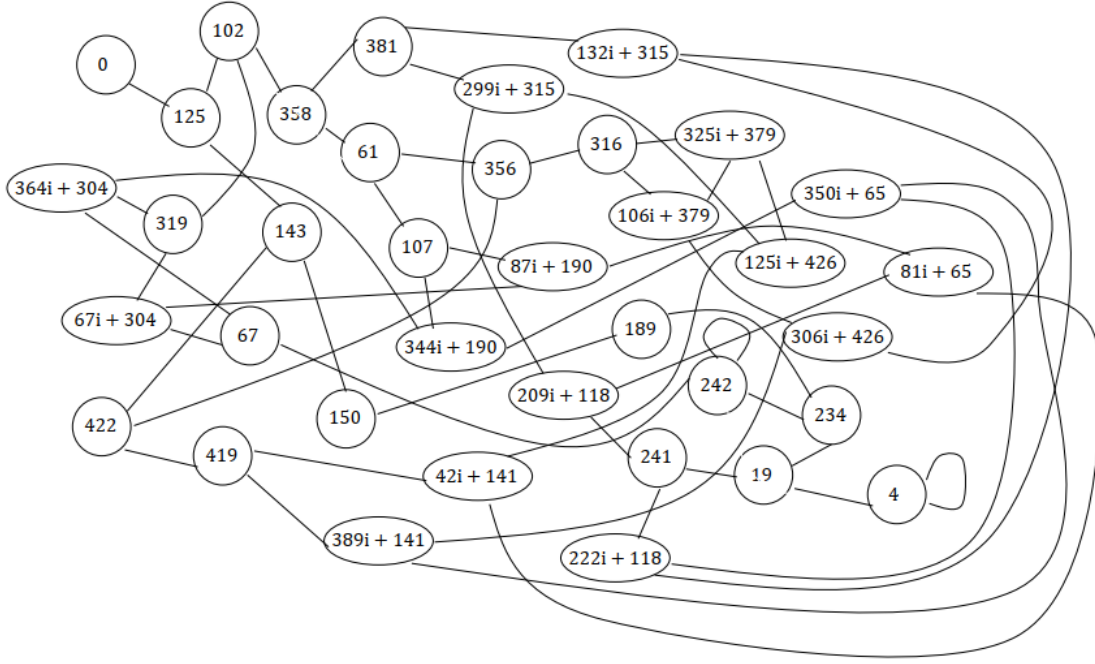


Figure 5.5: The 3-isogeny graph for $p = 431$. The 37 nodes are the supersingular j -invariants and the edges between them correspond to 3-isogenies

(ii) Secret key and generator

To compute her public key, Alice first chooses her secret key, namely $k_A \in 0, 1, \dots, 2^{e_A} - 1$. Then, Alice computes a secret generator S_A :

$$S_A = P_A + [k_A]Q_A$$

(iii) Public key

To reach the node set in her public key, Alice moves e_A times in her 2-isogeny graph, through the isogeny she computes.

Each 2-isogenies are calculated based on the kernel computed each time. After calculating an isogeny, she uses it to compute a new supersingular elliptic curves $E_{a'}$ and then compute its j -invariant $j(E_{a'})$, which correspond to the new node to go.

Her secret isogeny is

$$\phi_A : E \rightarrow E_A$$

where $E_A = E/\langle S_A \rangle$ (meaning that ϕ_A is based on the kernel $\langle S_A \rangle$), is a composition of the e_A isogenies of degree 2 found.

Eventually, Alice public key is

$$PK_A = (E_A, P'_B, Q'_B) = (\phi_A(E), \phi_A(P_B), \phi_A(Q_B))$$

where

- $E_A = \phi_A(E)$ is the image curve
- $P'_B = \phi_A(P_B)$ and $Q'_B = \phi_A(Q_B)$ are the images of Bob's public basis points.

This triplet of point composes the public key of Alice, and she will send it to Bob.

Similarly, Bob chooses $k_B \in \{0, 1, \dots, 3^{e_B} - 1\}$, calculates S_B and computes his secret isogeny $\phi_B : E \rightarrow E_B$, a composition of e_B isogenies of degree 3. His public key is then

$$PK_B = (E_B, P'_A, Q'_A) = (\phi_B(E), \phi_B(P_A), \phi_B(Q_A))$$

Shared secret

Once Alice received Bob's public key, she uses again her secret key k_A to compute the new secret generator S'_A

$$S'_A = P'_A + [k_A]Q'_A$$

As before, she computes another secret isogeny

$$\phi'_A : E_B \rightarrow E_{AB}$$

.

Finally, the shared secret is computed

$$j_{AB} = j(E_AB)$$

Bob do the same : he calculates $S'_B = P'_B + [k_B]Q'_B$, computes his new secret isogeny $\phi'_B : E_A \rightarrow E_{BA}$, and can eventually computes the shared secret $j_{BA} = j(E_{BA})$.

Remark : Why the image points ?

In Alice and Bob's public key, they put the image points of each other. The reason is that it is necessary to $\phi_{A'}$ on E_B or $\phi_{B'}$ on E_A , because it would not really make to consider a composition of the isogenies $\phi_A : E \rightarrow E_A$ and $\phi_B : E \rightarrow E_B$, given their domains and codomains.

These image of the basis points are used here in order to "describe" an isogeny whose domain is E_B , and an isogeny starting from E_A for Bob. That's why S'_A and S'_B are computed, based on the image points.

5.4.2 Detailing of each steps

5.4.3 Example

For the end of this section, let's finish with a full example of the SIKE protocol.

We continue with the same graphs as before. The public parameters are :

- $p = 2^4 3^3 - 1 = 431$, meaning $e_A = 4$ and $e_B = 3$
- $E_{a_0} : y^2 = x^3 + a_0 + x$, with $a_0 = 423 + 329i$ the starting curve
- $j(E_{a_0}) = 190 + 87i$ the starting node
- $P_A = (248 + 100i, 199 + 304i)$ and $Q_A = (394 + 426i, 79 + 51i)$, Alice's basis points
- $P_B = (275 + 358i, 104 + 410i)$ and $Q_B = (185 + 20i, 239 + 281i)$, Bob's basis points

Alice's public key computation

Let's say that Alice decides that her secret value is

$$k_A = 11$$

We indeed have $k_A \in \{0, 1, \dots, 2^4 - 1\}$.

The first step is the computation of her secret generator

$$\begin{aligned} S_A &= P_A + [k_A] Q_A \\ &= (248 + 100i, 199 + 304i) + [11] (394 + 426i, 79 + 51i) \\ &= (79 + 271i, 430 + 153i) \end{aligned}$$

Since P_A and Q_A are of order 16 on the curve E_{a_0} , so is S_A .

Alice now compute her public key, only by applying the Point Doubling Operation defined in ??, and the 2-isogeny operation in ??.

- Compute ϕ_0 :

Three repeated application of the doubling onto S_A produces the point

$$R_A^{(1)} = [8]S_A = (37 + 18i, 0)$$

which is of order 2 on E_{a_0} .

Inputting $R_A^{(1)}$ into ?? gives E_{a_1} , with $a_1 = 132 + 275i$ and $j(E_{a_1}) = 107$. We also obtain the map

$$\phi_0(x) = \frac{x((37 + 18i)x - 1)}{x - (37 + 18i)}$$

The basis points are also updated

$$P_B^{(1)} = \phi_0(P_B) = (85 + 118i, 150 + 274i)$$

$$Q_B^{(1)} = \phi_0(Q_B) = (124 + 62i, 269 + 64i)$$

$$S_A^{(1)} = \phi_0(S_A) = (111 + 36i, 67 + 175i)$$

$S_A^{(1)}$ is now of order 8 on E_{a_1} .

- Compute ϕ_1 :

Two repeated application of the doubling onto $S_A^{(1)}$ produces the point

$$R_A^{(2)} = [4]S_A^{(1)} = (49 + 7i, 0)$$

which is of order 2 on E_{a_1} .

Inputting $R_A^{(2)}$ into ?? gives E_{a_2} , with $a_2 = 76 + 273i$ and $j(E_{a_2}) = 190 + 344i$. We also obtain the map

$$\phi_1(x) = \frac{x((49 + 7i)x - 1)}{x - (49 + 7i)}$$

The basis points are also updated

$$P_B^{(2)} = \phi_1(P_B^{(1)}) = (251 + 274i, 59 + 316i)$$

$$Q_B^{(2)} = \phi_1(Q_B^{(1)}) = (94 + 214i, 193 + 354i)$$

$$S_A^{(2)} = \phi_1(S_A^{(1)}) = (374 + 274i, 77 + 84i)$$

$S_A^{(2)}$ is now of order 4 on E_{a_2} .

- Compute ϕ_2 :

One application of the doubling onto $S_A^{(2)}$ produces the point

$$R_A^{(3)} = [2]S_A^{(2)} = (27 + 245i, 0)$$

which is of order 2 on E_{a_2} .

Inputting $R_A^{(3)}$ into ?? gives E_{a_3} , with $a_3 = 136 + 93i$ and $j(E_{a_3}) = 65 + 350i$. We also obtain the map

$$\phi_2(x) = \frac{x((27 + 245i)x - 1)}{x - (27 + 245i)}$$

The basis points are also updated

$$P_B^{(3)} = \phi_3(P_B^{(2)}) = (209 + 77i, 79 + 75i)$$

$$Q_B^{(3)} = \phi_3(Q_B^{(2)}) = (356 + 339i, 419 + 12i)$$

$$S_A^{(3)} = \phi_3(S_A^{(2)}) = (150 + 227i, 0)$$

$S_A^{(3)}$ is now of order 2 on E_{a_3} .

- Compute ϕ_3 :

Since $S_A^{(3)} = (150 + 227i, 0)$ is of order 2, no scalar multiplication is necessary. Inputting $S_A^{(3)} = R_A^{(4)}$ into ?? gives E_{a_4} , with $a_4 = 179 + 423i (= a_A)$ and $j(E_{a_4}) = 118 + 222i$. We also obtain the map

$$\phi_3(x) = \frac{x((150 + 227i)x - 1)}{x - (150 + 227i)}$$

The basis points are also updated

$$P_B^{(4)} = \phi_3(P_B^{(3)}) = (183 + 142i, 360 + 119i)$$

$$Q_B^{(4)} = \phi_3(Q_B^{(3)}) = (314 + 220i, 10 + 289i)$$

And since $S_A^{(3)} \in \ker(\phi_3)$, we leave $S_A^{(3)}$ as it is.

Alice's secret 2^4 -isogeny is eventually the composition of the four 2-isogenies computed before.

$$\begin{aligned} \phi_A : E_{a_0} &\rightarrow E_{a_4} \\ \phi_A(x) &\mapsto (\phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0)(x) \end{aligned}$$

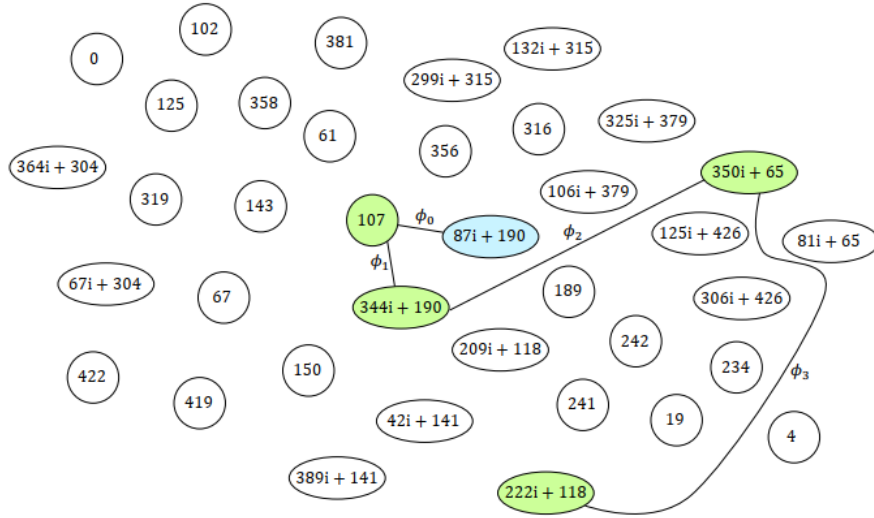


Figure 5.6: Alice's public key generation. The starting j -invariant is $190 + 87i$, and the final is $117 + 222i$

Eventually, her public key is

$$\begin{aligned} PK_A &= (a_A, \phi_A(P_B), \phi_A(Q_B)) \\ &= (179 + 423i, (183 + 142i, 360 + 119i), (314 + 220i, 10 + 289i)) \end{aligned}$$

Bob's public key generation

Let's say that Bob decides that his secret value is

$$k_B = 2$$

We indeed have $k_A \in \{0, 1, \dots, 3^3 - 1\}$.

The first step is the computation of his secret generator

$$\begin{aligned} S_B &= P_B + [k_B] Q_B \\ &= (275 + 358i, 104 + 410i) + [2] (185 + 20i, 239 + 281i) \\ &= (309 + 122i, 374 + 291i) \end{aligned}$$

Since P_B and Q_B are of order 16 on the curve E_{a_0} , so is S_B .

Bob now computes her public key, only by applying the Point Tripling Operation defined in ??, and the 3-isogeny operation in ??.

- Compute ϕ_0 :

Two repeated application of the tripling onto S_B produces the point

$$R_B^{(1)} = [9]S_B = (37 + 23i, 302 + 4i)$$

which is of order 3 on E_{a_0} .

Inputting $R_B^{(1)}$ into ?? gives E_{a_1} , with $a_1 = 2 + 134i$ and $j(E_{a_1}) = 379 + 106i$. We also obtain the map

$$\phi_0(x) = \frac{x((37 + 23i)x - 1)^2}{(x - (37 + 23i))^2}$$

The basis points are also updated

$$P_A^{(1)} = \phi_0(P_A) = (155 + 418i, 331 + 288i)$$

$$Q_A^{(1)} = \phi_0(Q_A) = (242 + 159i, 425 + 310i)$$

$$S_B^{(1)} = \phi_0(S_B) = (256 + 295i, 64 + 253i)$$

$S_B^{(1)}$ is now of order 9 on E_{a_1} .

- Compute ϕ_1 :

One application of the tripling onto $S_B^{(1)}$ produces the point

$$R_B^{(2)} = [3]S_B^{(1)} = (36 + 98i, 155 + 56i)$$

which is of order 3 on E_{a_1} .

Inputting $R_B^{(2)}$ into ?? gives E_{a_2} , with $a_2 = 54 + 117i$ and $j(E_{a_2}) = 379 + 325i$. We also obtain the map

$$\phi_1(x) = \frac{x((36 + 98i)x - 1)^2}{(x - (36 + 98i))^2}$$

The basis points are also updated

$$P_A^{(2)} = \phi_1(P_A^{(1)}) = (425 + 252i, 19 + 315i)$$

$$Q_A^{(2)} = \phi_1(Q_A^{(1)}) = (81 + 412i, 172 + 111i)$$

$$S_B^{(2)} = \phi_1(S_B^{(1)}) = (405 + 102i, 313 + 375i)$$

$S_B^{(2)}$ is now of order 3 on E_{a_1} .

- Compute ϕ_2 :

Since $S_B^{(2)} = (405 + 102i, 313 + 375i)$ is of order 3, no scalar multiplication is necessary. Inputting $S_B^{(2)} = R_B^{(3)}$ into ?? gives E_{a_3} , with $a_3 = 76 + 273i$ and $j(E_{a_3}) = 190 + 344i$. We also obtain the map

$$\phi_0(x) = \frac{x((405 + 102i)x - 1)^2}{(x - (405 + 102i))^2}$$

The basis points are also updated

$$P_A^{(3)} = \phi_2(P_A^{(2)}) = (226 + 187i, 360 + 43i)$$

$$Q_A^{(3)} = \phi_2(Q_A^{(2)}) = (415 + 325i, 254 + 322i)$$

And since $S_A^{(2)} \in \ker(\phi_2)$, we leave $S_A^{(2)}$ as it is.

Bob's secret 3^3 -isogeny is eventually the composition of the three 3-isogenies computed before.

$$\begin{array}{ccc} \phi_B : & E_{a_0} & \rightarrow E_{a_3} \\ & \phi_B(x) & \mapsto (\phi_2 \circ \phi_1 \circ \phi_0)(x) \end{array}$$

Eventually, his public key is

$$\begin{aligned} PK_B &= (a_B, \phi_A(P_B), \phi_A(Q_B)) \\ &= (273 + 76i, (226 + 187i, 360 + 43i), (415 + 325i, 254 + 322i)) \end{aligned}$$

Alice's shared secret computation

Alice starts over from the curve $E_B : y^2 = x^3 + a_B x^2 + x$, with a_B received within Bob's public key.⁸

Once again, Alice's first step is to compute a secret generator E_{a_0} based on her secret key k_A .

$$\begin{aligned} S_A &= \phi_B(P_A) + [k_A] \phi_B(Q_A) \\ &= (226 + 187i, 360 + 43i) + [11] (415 + 325i, 254 + 322i) \\ &= (357 + 125i, 249 + 415i) \end{aligned}$$

⁸Since we are doing the same process, we write E_0 for E_B and $a_0 = 76 + 273i = a_B$ as the new starting curve

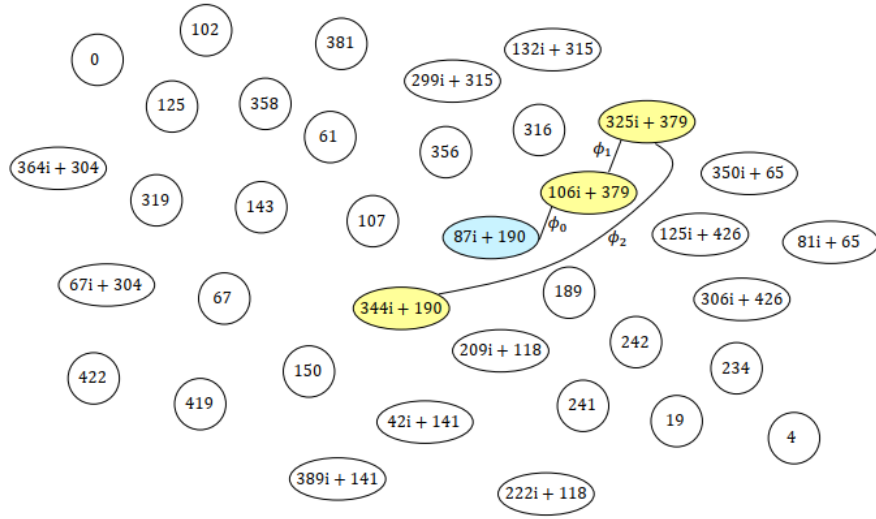


Figure 5.7: Bob's public key generation. The starting j -invariant is $190 + 87i$, and the final is $190 + 344i$

Then, she apply the exact same steps she did before, with the only difference that she doesn't need to compute the image of the basis points anymore, ie. $\phi_i(P_B^{i-1})$ and $\phi_i(Q_B^{i-1})$, because only the final node is wanted.

The four 2-isogenies computations are summarised as

$$\begin{aligned}
 \phi_0 : E_{a_0} &\rightarrow E_{a_1} & \text{with } a_1 &= 341 + 289i & \text{and } j(E_{a_1}) &= 304 + 364i \\
 \phi_1 : E_{a_1} &\rightarrow E_{a_2} & \text{with } a_2 &= 428 + 414 & \text{and } j(E_{a_2}) &= 67 \\
 \phi_2 : E_{a_2} &\rightarrow E_{a_3} & \text{with } a_3 &= 246i & \text{and } j(E_{a_3}) &= 242 \\
 \phi_3 : E_{a_3} &\rightarrow E_{a_4} & \text{with } a_4 &= 230 & \text{and } j(E_{a_4}) &= 234
 \end{aligned}$$

The shared secret she obtains is $j = 234$.

Bob's shared secret computation

Bob starts over from the curve $E_A : y^2 = x^3 + a_A x^2 + x$, with a_A received within Alice's public key.

Once again, Bob's first step is to compute a secret generator E_{a_0} based on her secret key k_B .

$$\begin{aligned}
 S_B &= \phi_A(P_B) & + & [k_B] \phi_A(Q_B) \\
 &= (183 + 142i, 360 + 119i) & + & [2] (314 + 220i, 10 + 289i) \\
 &= (124 + 393i, 380 + 187i)
 \end{aligned}$$

Then, he applies the exact same steps he did before, with the only difference that he doesn't need to compute the image of the basis points anymore. The three 3-isogenies computations are summarised as

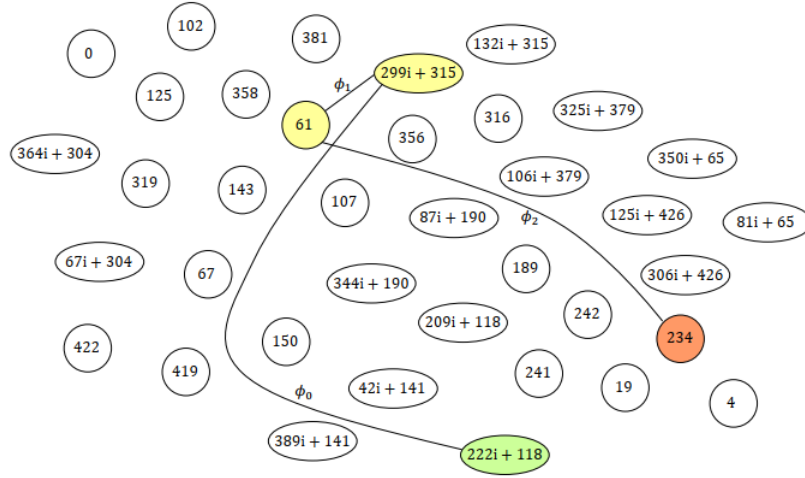


Figure 5.9: Bob's shared secret generation. The starting j -invariant is $118 + 222i$, and the final is 234

Algorithm 1: Encoding algorithm

Input : $PK_B, m \in M$

Output: (c_0, c_1)

$sk_A \leftarrow_R \mathbb{F}_{p^2}$

$PK_A \leftarrow isoPK(sk_A)$

$j \leftarrow isoSS(PK_B, sk_A)$

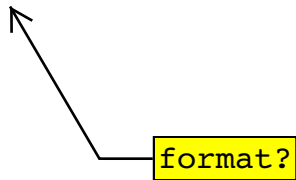
$h \leftarrow F(j)$

$c_1 \leftarrow h \oplus m$

Output: (c_0, c_1)

- $F(\cdot)$: hash function

And the decryption algorithm :



Algorithm 2: Decoding algorithm

Input : $sk_B, (c_0, c_1)$

Output: $m \in M$

$j \leftarrow isoSS(sk_B, c_0)$

$h \leftarrow F(j)$

$m \leftarrow h \oplus c_1$

Output: m

Chapter 6

SIKE and ART

Initially, we had two problems : creating a messaging group in an asynchronous way and making it resistant to quantum resistant. The first was resolved with the Asynchronous Ratchet protocol, while for the second one, a solution presented was the SIKE protocol.

Hence, we eventually have to combine these two protocols into one.

In ART, Diffie-Hellman is used in several steps of the protocol :

- In the initialization, when the creator calculates the leaf keys : Diffie-Hellman is used three times in X3DH, with the identity keys and some ephemeral keys.
- Still in the initialization, the parents' key pair are computed with DH, based on the public and private key of their children
- In the update of the keys : when a key leaf is modified, DH is used to re-calculates the leak key pair, and the key pair all the nodes in its path. The other nodes also need to update some of their parents node.

We will thus try to mix SIKE with each of these steps, in order to make them quantum resistant.

As a remained, the following parameters are public, and seen by every party :

- $p = 2^{e_A} \cdot 3^{e_B} - 1$, defining the field \mathbb{F}_{p^2}
- $\{(P_i, Q_i)\}_i$ a set of pair of basis points, used to compute a kernel $\langle S \rangle = \langle P_i + [k]Q_i \rangle$, based on a secret key k

6.1 X3DH and SIKE

As detailed in Section 5.1, the construction of the Asynchronous Ratchet Tree was initially based on Diffie-Hellman trees. This means it needs to be changed.

As a remainder, in ART, before computing any shared secret key, every party has publish some public keys : a public identity key IK and several public ephemeral keys $\{EK_i\}_i$. But before publishing them, one need to compute them.

- Given a pair (P_i, Q_i) , the party has to publish a public key IK , corresponding to the triplet (E'_{ik}, P'_i, Q'_i) , where E'_{ik} is a curve obtained by calculating and applying a 2-isogeny ϕ_2 onto an initial elliptic curve E_{ik} , ϕ_2 being based on the private key ik , and $P'_i = \phi_2(P_i)$, $Q'_i = \phi_2(Q_i)$.
- Similarly, given a new pair (P_j, Q_j) , the party has to publish a public key EK , corresponding to the triplet (E'_{ek}, P'_j, Q'_j) , where E'_{ek} is another curve obtained by calculating a 3-isogeny ϕ_3 based on the private key ik , and $P'_j = \phi_3(P_j)$, $Q'_j = \phi_3(Q_j)$.

The reason behind the choice of 2-isogeny for the identity keys and 3-isogeny for the ephemeral keys will be explained shortly after.

Once every party has publish their public keys, the creator of the group (let's say Alice) can start to compute the keys of each leaf.

To recall the process of X3DH, three shared secret have to be calculated, each one depending on a pair of key : the first is one of Alice's private key (ik_A or suk_A) and the second one is one of Bob's public key (IK_B or EK_B). The combination and the reason behind are shown in Table 6.1.

Secret	Alice	Bob	Main purpose
SS_1	ik_A	EK_B	Authentication of Alice, forward secrecy for Bob
SS_2	suk_A	IK_B	Authentication of Bob, forward secrecy for Alice
SS_3	suk_A	EK_B	Forward secrecy for both Alice and Bob

Table 6.1: The three shared secrets generated in X3DH

As we see, for the shared secret SS_1 and SS_2 , the identity of one party is combined with the ephemeral key of the other party. And in the presentation of the SIKE protocol between two party, one had to compute 2-isogenies, while the other one had to compute 3-isogenies. This is why we decided to link public identity keys with 2-isogenies and ephemeral keys with 3-isogenies.

Nevertheless, there is a problem with the last shared secret SS_3 : both party uses 3-isogenies, making it impossible to compute a shared secret based on SIKE. To solve this problem, we simply decide to leave the computation of this last shared secret. Moreover, we don't sacrifice any secure security property, since the two first shared secret already ensure forward secrecy for both parties.

Now, we are ready for the leaf keys. Given a pair (sk_a, PK_B) , where sk_a is one of Alice's private key, and PK_B one of Bob's public key, the creator computes a secret key as follow :

1. Alice computes the new kernel $\langle S_{A'} \rangle = \langle P'_B + [sk_a]Q'_B \rangle$, where $PK_B = (E_B, P'_B, Q'_B)$
2. She then computes the new 2- or 3-isogeny (according to her private key, the identity or private one), compute the elliptic curve E_{AB} and find the shared secret $j(E_{AB})$

By doing so, Alice obtains the shared secrets SS_1 and SS_2 , and eventually, she calculates Bob's leaf key :

$$\lambda_B = KDF(SS_1 || SS_2)$$

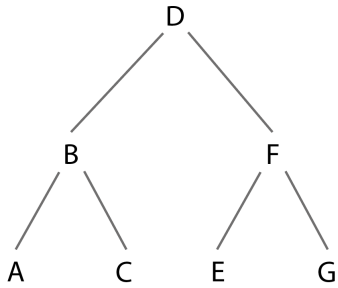
.

For all the other members, the computations are the same. Alice just need to use the public keys of each party.

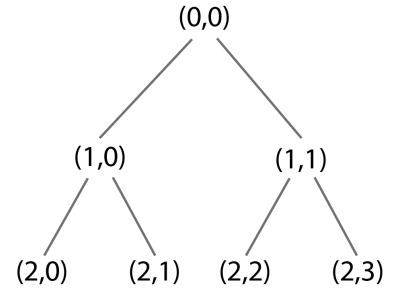
6.2 Nodes' keys computations

Once Alice has calculated all the leafs' keys and sent them to the corresponding party, she can calculate the nodes' keys. With its private key, a node can compute its corresponding public key, annotated PK_C , which is a triplet (E_C, P'_C, Q'_C) . So, now, each leaf is associated to a key pair. It is then possible to compute a node private key, based on the keys of its children.

Nevertheless, we are again faced with the problem of who computes 2-isogenies and who computes 3-isogenies. To decide this, we instaure the rule that, since each node is associated to an index (x, y) , where x is the depth of the node in the tree (the root being of depth 0), and y the position of the node for a given depth, starting at 0. Then, an "even" node (ie. $y \equiv 0[2]$) will compute 2-isogeny, while an "odd" even will deal with 3-isogenies. An example of indexing the node is given in Figure 6.1.



(a) Left-balanced binary tree



(b) Corresponding indexed tree

Figure 6.1: Indexation of a tree. The "even" nodes correspond to 2-isogenies, and the "odd" nodes represent the 3-isogenies

With this simple rule, it is thus possible to compute the root secret key, which is the secret shared between all members of the group, that will be used as key in a symmetric encryption algorithm, such as AES.

6.3 Updating the leaf's keys

Adding a member

Let's say the group initiator Alice wants to add a new member Charlie to the group. As indicated previously, Charlie has published into the server some public keys : its identity key and some ephemeral keys, each one of them composed of a triplet (an elliptic curve, and two image points). As in the construction, Alice first applies X3DH for authentication and in order to supply Charlie its secret key. Then, both Alice and Charlie can compute the corresponding public key, and the parents in Charlie's path.

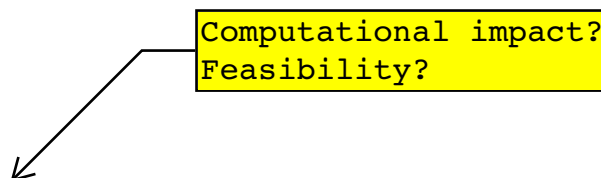
By updating all the keys of the nodes in Charlie's path, the root was also updated. This means that all the other members need to update the root key as well, by considering the new nodes' keys. To do so, Alice has to send to the other members of the group the public values for these intermediate nodes, encrypted by the public key of the leaf receiving the information. But Alice does not send all the new values to all the members. A member only receives the new public of a node if the latter is in the direct path of the leaf. But it is not necessary that they receive the new value of the parents of the first modified node, since they will re-calculate themselves.

EXAMPLE

Updating a key leaf If a member of the group wants to update its private key, he will obtain a new public key, and all the key pair of the nodes in its path will be updated as well. Therefore, similarly to before, he will have to send to the concerned member the value of the new public keys.

These computations for the new keys follow the rule established before, namely an "even" node would use 2-isogenies, while an odd would use 3-isogenies.

Deleting a key leaf Eventually, if a user leaves the group, or is deleted (we consider here that it is someone else that decides to take the member out of the group), then the parent of this late member is replaced by the sibling of the former. But since this sibling keeps its key pair, it is necessary to recompute all the keys in its path.



Bibliography

- [ACC⁺17] Reza Azarderakhsh, Matthew Campagna, Craig Costello, LD Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 2017.
- [ATAa14] MA Alia, AA Tamimi, and ONA Al-allaf. Cryptography based authentication methods, vol, 2014.
- [Aza] Reza Azarderakhsh. Performance of quantum-safe isogenies on arm processors. ETSI.
- [BBC13] Marco Baldi, Marco Bianchi, and Franco Chiaraluce. Security and complexity of the mceliece cryptosystem based on quasi-cyclic low-density parity-check codes. *IET Information Security*, 7(3):212–220, 2013.
- [BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–16. Springer, 2014.
- [BMO⁺19] Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The messaging layer security (mls) protocol. *Working Draft, IETF Secretariat, Internet-Draft draft-ietf-mls-protocol-07*, 2019.
- [Bro09] Daniel RL Brown. Sec 1: Elliptic curve cryptography. *Certicom Research*, page v2, 2009.
- [BW17] William Buchanan and Alan Woodward. Will quantum computers be the end of public key encryption? *Journal of Cyber Security Technology*, 1(1):1–22, 2017.
- [Byl90] Czesław Bylinski. The complex numbers. *Def*, 7(1C):1, 1990.
- [CGCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178. IEEE, 2016.

- [CGCG⁺18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1802–1819, 2018.
- [CH17] Craig Costello and Huseyin Hisil. A simple and compact algorithm for sidh with arbitrary degree isogenies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 303–329. Springer, 2017.
- [CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.
- [CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 453–474. Springer, 2001.
- [Cos19] Craig Costello. Supersingular isogeny key exchange for beginners. In *International Conference on Selected Areas in Cryptography*, pages 21–50. Springer, 2019.
- [Cou06] Jean Marc Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, 2006:291, 2006.
- [DF17] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2017.
- [dF19] Luca de Feo. Isogeny graphs in cryptography. 2019.
- [DIK06] Christophe Doche, Thomas Icart, and David R Kohel. Efficient scalar multiplication by isogeny decompositions. In *International Workshop on Public Key Cryptography*, pages 191–206. Springer, 2006.
- [DK07] H Delfs and H Knebl. Introduction to cryptography principles and applications, 2007.
- [Doe19] John Doe. Measurements of public-key cryptosystems, indexed by machine. <http://bench.cr.yp.to/results-stream.html>, 2019.
- [EH18] Youssef El Housni. Edwards curves. 2018.
- [Gal01] Steven D Galbraith. Supersingular curves in cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 495–513. Springer, 2001.
- [JDF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.

- [Len04] Arjen K Lenstra. Key length. contribution to the handbook of information security. 2004.
- [LL93] Arjen K Lenstra and Hendrik W Lenstra. *The development of the number field sieve*, volume 1554. Springer Science & Business Media, 1993.
- [LP07] Hyun-Yong Lee and In-Cheol Park. Balanced binary-tree decomposition for area-efficient pipelined fft processing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(4):889–900, 2007.
- [Mar03] John Levi Martin. What is field theory? *American journal of sociology*, 109(1):1–49, 2003.
- [Mat19] Dark Matter. Introduction to Post-Quantum Cryptography. https://crypto.polito.it/content/download/200/1205/file/slide_bellini.pdf, 2019.
- [MLS19] Michiel Marcus, T Lange, and P Schwabe. White paper on mceliece with binary goppa codes, 2019.
- [P⁺14] Mark Pecun et al. Quantum safe cryptography and security: An introduction, benefits, enablers and challenges, white paper. *European Telecommunications Standards Institute*, 2014.
- [Pau17] Eduard Paul. What is Digital Signature- How it works, Benefits, Objectives, Concept. <https://www.emptrust.com/blog/benefits-of-using-digital-signatures>, 2017.
- [PM16a] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. *GitHub wiki*, 2016.
- [PM16b] Trevor Perrin and Moxie Marlinspike. The x3dh key agreement protocol. *Specification*. Nov, 2016.
- [QQL10] Bing Qi, Li Qian, and Hoi-Kwong Lo. A brief introduction of quantum cryptography for engineers. *arXiv preprint arXiv:1002.1237*, 2010.
- [Ren18] Joost Renes. Computing isogenies between montgomery curves using the action of $(0, 0)$. In *International Conference on Post-Quantum Cryptography*, pages 229–247. Springer, 2018.
- [SBBB12] William Stallings, Lawrie Brown, Michael D Bauer, and Arup Kumar Bhattacharjee. *Computer security: principles and practice*. Pearson Education Upper Saddle River, NJ, USA, 2012.
- [Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

- [Spa16] John Spacey. What is a Key Derivation Function. <https://simplicable.com/new/key-derivation-function>, 2016.
- [Str11] Emma Strubell. An introduction to quantum algorithms. *COS498 Chawathe Spring*, 13:19, 2011.
- [Sug18] YK Sugi. What is a quantum computer? explained with a simple example. <https://www.freecodecamp.org/news/what-is-a-quantum-computer-explained-with-a-simple-example-b81>, 2018.
- [Vél71] Jacques Vélú. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.
- [Vin03] Èrnest Borisovich Vinberg. *A course in algebra*. Number 56. American Mathematical Soc., 2003.
- [Whi16] WhitePaper. Whatsapp encryption overview. 2016.
- [Yel16] Baris Yeldiren. What is End-to-end encryption (E2EE) ? <https://www.printfriendly.com/p/g/Sddiq3>, 2016. [Online; accessed 19-July-2008].