Contents

1 Preface ................................................................. 5

2 Basics of Cryptography ....................................... 6
   2.1 Information Security ........................................... 6
   2.2 Basic Terminology and Concepts ............................ 7
   2.3 Symmetric Key Cryptography ................................. 8
      2.3.1 Block Ciphers ............................................. 10
      2.3.2 Stream Ciphers .......................................... 11
   2.4 Public Key Cryptography ..................................... 11
      2.4.1 Digital Signature ......................................... 12
      2.4.2 Hashes ..................................................... 14
   2.5 Symmetric Key vs. Public Key ............................... 16

3 The SSL/TLS-Protocol ........................................... 18
   3.1 Introduction .................................................. 18
   3.2 Basic Structure ................................................ 19
   3.3 TLS-Record Protocol ......................................... 20
   3.4 ChangeCipherSpec protocol .................................. 22
      3.4.1 Diffie-Hellman .......................................... 22
      3.4.2 RSA ...................................................... 24
      3.4.3 MD-5 ..................................................... 25
      3.4.4 RC4 ..................................................... 27
   3.5 TLS-Handshake Protocol .................................... 29
   3.6 Alert Protocol ................................................ 33
   3.7 Application Data Protocol .................................. 34

4 Elliptic Curve Cryptography .................................. 35
   4.1 Why Elliptic Curve Cryptography? ......................... 35
   4.2 Introduction to Elliptic Curves .............................. 35
      4.2.1 Torsion Points .......................................... 38
   4.3 Elliptic Curves over finite Fields ......................... 39
   4.4 The Weil Pairing .............................................. 41
      4.4.1 Weil Divisors ........................................... 41
1 Preface

The field of cryptography has a long and fascinating history which has gone through numerous changes in its purpose and its application. While 4000 years ago, cryptography was essential for governmental communication via couriers, throughout the ages new technologies brought about effective ways to attack and secure all our daily communication. Science has developed and computers were built, which allow us to communicate via the internet. Not only governments and politics are dependent on cryptography now. Ordinary people using internet applications and e-mail communication are highly interested in maintaining their privacy and securing their communication.

The most striking development in the history of cryptography occurred in 1976, when Diffie and Hellman introduced the concept of public key cryptosystems. Until then, every cryptosystem was based on a mutual secret shared by both communication partners (symmetric key cryptosystems). This lead to a rather uncomfortable problem of sharing a secret key before the actual communication could start. The idea of public key cryptography was based on so called one-way-functions as they are easy to compute but hard to invert. The Discrete Logarithm Problem and the Factorization Problem are two examples of this phenomenon. Despite being arguably two of the most analyzed problems, no efficient solutions are yet known.

The key exchange protocol introduced by Diffie and Hellman was based on the Discrete Logarithm Problem. Two years later, in 1978, Rivest, Shamir and Adleman developed the public key encryption and signature scheme, which was based on the (FP) Factorization Problem. They also solved the issue of appropriately identifying yourself by using so called digital signatures, which are only possible by applying the principles of public key cryptography.

In 1985 Neal Koblitz and Victor Miller independently proposed the idea of using elliptic curves in public key systems. Elliptic curve cryptography schemes provide the same mechanisms as the RSA Systems mentioned above. However, they are not based on the DLP but on the ECDLP (Elliptic Curve Discrete Logarithm Problem) which is harder to solve. This allows for higher security with much smaller keys (a 160-bit EC key provides the same level of security as a 1024-bit RSA key).

All these theoretical concepts of public key, symmetric key and elliptic curve cryptography have to be realized in practice. That is, they have to be implemented in internet protocols in order to guarantee secure communication with websites or servers. The SSL/TLS-
Protocol is an internet protocol that combines all the above concepts in order to have the most efficient use of theoretical and practical knowledge.

This work tries to give an overview of the current state of art in the TLS-protocol and Elliptic Curve Cryptography. It is divided into three parts.

Chapter 1 is an introductory chapter providing all the basics in cryptography that are essential to understand further analysis. It provides examples of the most common public key, symmetric key and signature systems. Also hashes will be covered.

Chapter 2 presents the SSL-TLS protocol as it is used now (in 2015) in practice.

Chapter 3 gives a comprehensive introduction into the field of elliptic curves and how elliptic curves are used in cryptography. The core of this chapter will be the so called MOV-Attack, which tries to solve the ECDLP using Weil-Pairings and other advanced tools in elliptic curves. An implementation of the MOV-Attack in SAGE is provided as well.

2 Basics of Cryptography

This chapter will give some basic terminology and concepts of cryptography which is essential for the following chapters. We will compare the two main concepts (public key and symmetric key cryptography) and explain what other tools are used in practice (such as Hashe, Signatures, etc.).

2.1 Information Security

In this section we will formally introduce cryptography and its goals to give a concise overview of the the field.

**Definition 2.1.** Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication etc.

**Cryptographic Goals**

The following four cases build the framework for cryptography. Addressing these areas is the main goal when designing cryptographic primitives [1]:

1. **Confidentiality** is the service that provides the information to those who are authorized to access it and only to those.
2. **Integrity** is the service which addresses unauthorized alteration of data (data manipulation).

3. **Authentication** is the service which guarantees that the participating parties will identify each other. Furthermore, each data which is exchanged should be tractable in its origin, date of origin, data sent, etc.

4. **Non-repudiation** is the service which prevents an entity from denying commitments or actions.

A cryptosystem is supposed to address these 4 goals adequately.

### 2.2 Basic Terminology and Concepts

In this chapter we will define basic notations and concepts in cryptography, which will be used throughout this work. We will start with encryption domains and codomains.

#### Encryption domains and codomains

1. \( \mathcal{A} \) denotes a set which is called the alphabet. For example, \( \mathcal{A} = \{0, 1\} \) is called the binary alphabet. Note that every alphabet can be described by a binary alphabet. We will use this property when describing advanced private key cryptosystems.

2. \( \mathcal{M} \) is called the message space. It consists of strings of elements from the alphabet.

3. \( \mathcal{C} \) is called the ciphertext space. It consists of elements from an alphabet (which may differ to the alphabet of the message space). An element \( c \in \mathcal{C} \) is called ciphertext.

#### Encryption and decryption transformations

1. \( \mathcal{K} \) denotes a set called the keyspace. An element \( e \in \mathcal{K} \) is called key.

2. Each element \( e \in \mathcal{K} \) uniquely determines a bijection from \( \mathcal{M} \) to \( \mathcal{C} \), denoted by \( E_e \). \( E_e \) is called an encryption function.

3. For each \( d \in \mathcal{K} \), \( D_d \) denotes a bijection from \( \mathcal{C} \) to \( \mathcal{M} \). \( D_d \) is called a decryption function.
4. An encryption scheme consists of a set \( \{ E_e : e \in \mathcal{K} \} \) of encryption transformations and a corresponding set \( \{ D_d : d \in \mathcal{K} \} \) of decryption transformations. The latter set has to fulfill the property that for each \( e \in \mathcal{K} \) there is a unique key \( d \in \mathcal{K} \) such that \( D_d(E_e(m)) = m \) \( \forall m \in \mathcal{M} \). An encryption scheme is also called cipher.

5. The keys \( e \) and \( d \) are called the keypair and are denoted by \((e,d)\).

6. To construct an encryption scheme one has to select a message space \( \mathcal{M} \), a ciphertext space \( \mathcal{C} \), a key space \( \mathcal{K} \), a set of encryption transformations \( \{ E_e : e \in \mathcal{K} \} \) and a corresponding set of decryption transformations \( \{ D_d : d \in \mathcal{K} \} \).

Figure 1: This shows a basic encryption scheme. If Alice wants to send a message to Bob, she has to encrypt her plaintext message using the encryption function \( E_e \). This gives a ciphertext \( c \) which can be sent to Bob via an unsecured channel. The attacker (adversary) can eavesdrop the communication, but cannot get the plaintext message since the necessary keys are missing. Bob receives the ciphertext message \( c \), decrypts with the decryption function \( D_d \) and obtains the plaintext message back. [1]

### 2.3 Symmetric Key Cryptography

As described in the introduction, cryptographic systems can be divided into two groups: symmetric key and public key. In this section we introduce the basic idea of symmetric cryptography and some easy examples. Symmetric key cryptography was used until the
late 70’s, when public key systems were introduced. It was already used thousands of years ago when couriers had to deliver messages in person. If they were kidnapped the message should be unintelligible to the captors.

**Definition 2.2.** Consider an encryption scheme consisting of the sets of encryption and decryption transformations \( \{E_e : e \in \mathcal{K}\} \) and \( \{D_d : d \in \mathcal{K}\} \), with \( \mathcal{K} \) being the key space. The encryption scheme is said to be symmetric key if for each associated pair \((e, d)\), it is computationally “easy” to determine \(d\) knowing only \(e\) and vice versa.

Figure 2: This figure shows a symmetric key communication scheme, in which the key source is essential for encryption and decryption. Since the key is used on both sides, both parties have to share the secret key before actually communicating. This is why they have to agree upon a shared secret in person or via an unsecured channel. This problem is referred to as the key distribution problem and is solved by the Diffie-Hellman key exchange (public key cryptography in general). [1]

Now we want to classify symmetric key algorithms. Symmetric key encryption is divided into two subgroups: Block ciphers and Stream ciphers. We will shortly introduce both methods and give examples in Chapter 2.
2.3.1 Block Ciphers

Block ciphers are encryption schemes which split the plaintext message into strings of a fixed length (blocks) and encrypt one block at a time. Nowadays AES (Advanced Encryption Standard) is the most important symmetric key scheme. It is a block cipher and is used in the SSL/TLS protocol.

Example 2.3. Let $\mathcal{A}$ be an alphabet of $q$ symbols and $\mathcal{M}$ be the set of all strings of length $t$ over $\mathcal{A}$. For each permutation $e$ over the set $\mathcal{A}$ we define the encryption transformation $E_e$ as:

$$E_e(m) = (e(m_1)e(m_2)\ldots e(m_t)) = (c_1c_2\ldots c_t) = c$$

with $m = (m_1m_2\ldots m_t) \in \mathcal{M}$

$E_e$ is called a simple substitution cipher.

Substitution ciphers are not very effective since they provide a low level of security even for a large keyspace. Considering the english alphabet, the size of the keyspace is $26! \approx 4 \times 10^{26}$. Still using only few ciphertexts, the key can be derived by basic frequency analysis.

Example 2.4. Consider a symmetric key block encryption scheme as above (blocklength $t$). Let $\mathcal{K}$ be the set of all permutations over the set $\{1, \ldots t\}$. For each $e \in \mathcal{K}$ define the encryption function

$$E_e(m) = (m_{e(1)}m_{e(2)}\cdots m_{e(t)})$$

with $m = (m_1m_2\ldots m_t) \in \mathcal{M}$. The set of all transformations is called transposition cipher. The decryption to $e$ is the inverse permutation $d = e^{-1}$.

In other words: Transposition ciphers do not change the values of the plaintext message but change the locations of its characters. These two basic classes of symmetric key block ciphers correspond to two terms Shannon introduced in [2]: Confusion and Diffusion.

Definition 2.5. Confusion means that each character of the ciphertext should depend on several parts of the key. Diffusion means that if we change a character of the plaintext, then several characters of the ciphertext should change, and similarly, if we change a character of the ciphertext, then several characters of the plaintext should change. [3]

That is, diffusion spreads the information of the plaintext over the ciphertext. Transposition ciphers clearly have a good diffusion rate since changing one character in the plaintext
would change many characters in the ciphertext. Confusion, on the other hand, means that each character of the ciphertext should depend on several parts of the key. This is usually the case in substitution ciphers.

The ultimate goal of a good cryptosystem is combining these two principles as effectively as possible.

2.3.2 Stream Ciphers

Stream ciphers are block ciphers with length \( t = 1 \). That means that the encryption information can change from character to character. When transmission errors are likely, stream ciphers are effective, because they don’t have any error propagation. Alternatively they can be used when no buffers or no memory is available, since symbols are processed one at a time.

**Definition 2.6.** Let \( \mathcal{K} \) be the key space for a set of encryption transformations. A sequence of symbols \( e_1 e_2 e_3 \ldots e_n, \ e_i \in \mathcal{K} \ \forall i \leq n \) is called a keystream.

**Definition 2.7.** Let \( \mathcal{A} \) be an alphabet of \( q \) symbols and let \( E_e \) be a substitution cipher with block length 1 where \( e \in \mathcal{K} \). Let \( m_1 m_2 m_3 \ldots \) be a plaintext string and let \( e_1 e_2 e_3 \ldots \) be a key stream from \( \mathcal{K} \). A stream cipher takes the plaintext string and produces a ciphertext string \( c_1 c_2 c_3 \ldots \) where \( c_i = E_{e_i}(m_i) \). If \( d_i \) denotes the inverse of \( e_i \), then \( D_{d_i}(c_i) = m_i \) decrypts the ciphertext string.

2.4 Public Key Cryptography

Now we come to the counterpart of symmetric key cryptography, the public key cryptography. Introduced in 1967, the Diffie-Hellman key exchange solved the above mentioned key distribution problem and thus opened the door to a new world of protocols and algorithms. Two years later, the RSA-protocol was developed and was the first real public key primitive. In the following we will show the basic ideas and concepts of public key cryptography. In Chapter 3 we will analyze the RSA-Encryption in more detail.

Before we start, we have to define an important notion in order to understand this new system of encryption. We need so called trapdoor one-way functions:

**Definition 2.8.** A one-way function is a function \( f : X \rightarrow Y \) if \( f(x) \) is computationally easy (for example polynomial running time) for all \( x \in X \) but for most elements \( y \in Im(f) \) it is computationally hard to find \( x \in X \) such that \( f(x) = y \).
Example 2.9. Let $X = \mathbb{F}_{13}$ and $f(x) = 2^x$. It is easy to compute $2^x$ in $\mathbb{F}_{13}$ if $x$ is given. But for each $y \in \mathbb{F}_{13}$ it is hard to find $x \in \mathbb{F}_{13}$ with $2^x = y$.

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: The function $f(x) = 2^x$ on $\mathbb{F}_{13}$.

Definition 2.10. A trapdoor one-way function is a one-way function which can be easily inverted if a certain secret information is known.

This secret information will be the private parameter in public key schemes. How this secret information looks like can be seen in the RSA-algorithm (confer Chapter 3).

Consider any pair of encryption/decryption transformations denoted by $(E_e, D_d)$. Furthermore suppose that we know $E_e$ and a ciphertext $c \in \mathcal{C}$. We want to find a message $m \in \mathcal{M}$ such that $E_e(m) = c$. That means that given an $e$ it is hard to determine the corresponding key $d$. Here $E_e$ can be viewed as a one-way trapdoor function and $d$ as its trapdoor information necessary to compute the inverse function (i.e. the decryption function). This is the difference to symmetric key cryptography, where the decryption and encryption function are essentially the same, since they can be easily transformed into one another.

This leads to our formal definition of a public key scheme.

Definition 2.11. An encryption scheme consisting out of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$ is said to be a public key encryption scheme if for each associated pair $(e, d)$ the encryption key $e$ is public and the decryption key $d$ is held private. $e$ is called the public key and $d$ is called the private key.

For the scheme to be secure, it must be computationally hard to compute $d$ from $e$.

2.4.1 Digital Signature

After looking at the public key scheme, one might think that it would be safe to communicate over an unsecured channel, since $d$ cannot be computed knowing $e$. This is, however, not the case because it bears some other problems. The attacker could attack the system without cracking the private key $d$ but by impersonating both entities. This would work with a classic man-in-the-middle attack.
Figure 3: Bob selects a key pair \((e, d)\) and sends \(e\) to Alice. Note that in this case this information is sent via an unsecured channel, as opposed to the symmetric key case, where the key has to be delivered via a secure channel. The public key \(e\) can be sent unsecurely since it is computationally hard to derive the decryption key \(d\) (the private key) from the public key \(e\). Alice can now use the public key to encrypt a message and send the ciphertext to Bob over an unsecured channel. Since Bob is the only one in possession of the trapdoor information (the private key \(d\)), he is the only one able to decrypt the message. [1]

The attacker prevents the initial key exchange of the public key and sends Alice his own public key. On the other hand, he grabs the public key of Bob and has now two end-to-end connections with both Alice and Bob. None of whom realizing that the communication is going through a third entity, which makes the communication useless. This leads to the idea of a digital signature: Every message has to be uniquely identified via its sender. But how can one achieve this? First we explain the basic idea of a signature and verification function and then apply this to public key schemes.

**Definition 2.12.**
- \(\mathcal{S}\) is the set of elements (for example binary strings of a fixed length). These elements are called signatures.
- \(S_A\) is a function \(S_A : \mathcal{M} \rightarrow \mathcal{S}\) and is called a signing function for \(A\). \(S_A\) is held secret by \(A\) and will be used to sign messages out of the message space \(\mathcal{M}\).
• $V_A$ is a function $V_A : \mathcal{M} \times \mathcal{S} \rightarrow \{true, false\}$. It is called the verification function for $A$’s signatures and is used to verify signatures from $A$.

The functions $S_A$ and $V_A$ form a digital signature mechanism for $A$. The Sender $A$ selects a message out of the message space and computes $s = S_A(m)$ and transmits $(m, s)$. To verify the received pair $(m, s)$ the receiver computes $V_A(m, s)$ and checks if the output is true or false.

In order to construct a digital signature mechanism for a public key system with explicit functions $S_A$ and $V_A$, additional properties are required. We want the public key system to satisfy $\mathcal{M} = \mathcal{C}$ which leads to

$$D_d(E_e(m)) = E_e(D_d(m)) = m, \forall m \in \mathcal{M}.$$ (Otherwise $D_d(m)$ would not make any sense). A public key encryption scheme with this property is said to be reversible. We can now formally describe a digital signature scheme for reversible public key systems [1]:

1. Message space $\mathcal{M}$ for the signature scheme.
2. $\mathcal{M} = \mathcal{C}$ is the signature space $\mathcal{S}$.
3. $(e, d)$ is the key pair of the public key encryption scheme.
4. Define the signing function as $S_A := D_d$.
5. Define the verification function $V_A$ as

$$V_A(m, s) = \begin{cases} true & \text{if } E_e(s) = m \\ false & \text{otherwise} \end{cases}$$

Summing up this section we see that we can use any reversible public key scheme in order to construct a digital signature scheme. We will now proceed with hash functions.

2.4.2 Hashes

Digital signatures do not solve all problems when it comes to signing messages effectively. One arising problem, which we have not considered yet, is the unlimited message length. We do not want to set any restrictions to the message length, but considering our current
tools, we would have to split the message into equal parts (which are appropriate for our signing scheme) and then sign each part individually. This is not practical. Here one-way hash functions come into play.

**Definition 2.13.** A hash function is a computationally efficient function mapping strings of arbitrary length to strings of some fixed length, called hash values.

The idea is that hashes yield short representations for much longer strings. We can then easily sign the hash values. To be of cryptographic use, however, we want the hash function to be collision free. That means that it should be computationally hard to find \( x \) and \( y \) such that \( h(x) = h(y) \) and given \( y \) to find \( x \) such that \( h(x) = y \). Integrating this idea into our digital signature scheme would mean the following: The sender chooses his message \( m \in \mathcal{M} \) and hashes the message to get \( m_h = h(m) \). In the next step he signs the hash value, i.e. he uses his private key function on \( m_h \) to get \( s = D_d(m_h) \) and sends the tuple \((m, s)\) to the receiver. The receiver uses the senders public key, to "open" \( s \) by using \( E_e(s) = E_e(D_d(m_h)) = m_h \) and uses the hash function on the received message \( m \). If the hash of the message he computed and the hashvalue he received are the same, the identification was successful.
2.5 Symmetric Key vs. Public Key

At first glance one might find the public key scheme superior and could be inclined to use it exclusively. But there are some important advantages and disadvantages to both systems. That is why they are implemented in practice in specific ways, each of them with special areas of use. We will explore the pros and cons for both systems and learn their use in the next chapter when analysing the SSL/TLS protocol. [1]

**Advantages of symmetric key cryptography**

1. Symmetric key ciphers have a higher bit rate. That means that they are much faster and much more suitable for actual communication.

2. Their keys are usually much shorter.

3. Symmetric key ciphers can be designed to get stronger ciphers.

**Disadvantages of symmetric key cryptography**

1. When two parties are communicating, the key must remain secret at both entities.

2. Before the symmetric key communication starts, they have to secretly exchange a key (Key Distribution Problem).

3. In a big network, every communication pair has to exchange and select keys, which produces a huge number of keys. This makes a trusted supervisor of all communications important.

4. It is considered best to frequently change the key, which is harder to do because of the Key Distribution Problem.

5. It is rather complicated to construct a digital signature scheme from a symmetric key scheme.

**Advantages of public key cryptography**

1. Only the private keys have to stay secret.
2. In a big network, the organisation is way easier than in the case of symmetric keys. Each entity has an entry in an openly accessible table with his name and his public key. Everyone who wants to communicate with this entity can look up the public key in the table and can start communicating.

3. The keys do not have to be changed frequently.

4. It is easy for public key schemes to develop digital signature schemes.

**Disadvantages of public key cryptography**

1. The throughput rates for most public key systems are significantly slower than the throughput rates of symmetric key systems.

2. Key sizes are usually a lot larger than in the symmetric key case.

3. Public key schemes are based on trapdoor one-way functions which are not yet proven to exist. They are essentially assuming the hardness of some number theoretic problems such as the DLP or the ECDLP.

**Summary**

We see that the advantages and disadvantages are mostly complementary. This gives rise to the nice idea to use hybrid systems which make use of the advantages of both systems. Public key systems can be used to exchange keys for the symmetric key communication and used to design some digital signature schemes. The symmetric key system is more suitable for normal communications because of its high bit rates. This concludes the introductory chapter. Next we dive into the world of SSL/TLS where we will explore how all these mechanisms are implemented in practice.
3 The SSL/TLS-Protocol

So far we have looked into the world of cryptography from a theoretical point of view. But all these methods and primitives are useless if they are not properly implemented into practice. In this chapter we will analyze the SSL/TLS protocol from a cryptographic perspective.

3.1 Introduction

The SSL/TLS protocol is a hybrid encryption protocol for secure data transmission. SSL (Secure Socket Layer) was introduced in 1994 and has been improved over the years. In 2008 it was replaced by the TLS protocol, developed by IETF (Internet Engineering Task Force). Sometimes TLS 1.0 is also called SSL 3.1 since it mainly added some features to the current SSL protocol. However, they are not able to communicate with each other. That is why there is a downgrade function implemented in TLS which makes it possible to communicate with SSL 3.0. This downgrading also bears some risks and makes it vulnerable for attacks.

TLS combines all the methods introduced in Chapter 2 in a powerful manner. It uses positive properties of each primitive to get the best security possible. The goal of the design of the SSL/TLS protocol was to improve flexibility. That is, it should be very easy to adjust the protocol for specific tasks to make it easily applicable to many different areas. It is, however, vulnerable for man-in-the-middle attacks and only suited for end-to-end connections. It is commonly implemented in web-browsers and e-mail traffic.

It is important to note though, that the analysis of cryptographic algorithms only provides theoretical security, while an easy implementation is also a huge factor for security. Security in theory and in practise are two different things and both have to be considered with caution.

The SSL/TLS protocol provides connection security which has two basic properties:

1. The connection is private. Symmetric key cryptography is used for data encryption, with session keys agreed upon every time for each connection. They are based on a secret information in the TLS handshake protocol. Alternatively one can also use the record protocol without encryption.

2. The connection is reliable. Message identity check and authentication are used with MACs, HMACs and/or hash functions.
3.2 Basic Structure

The SSL/TLS Protocol is a feature for other applications and protocols, that communicate via an unsecured channel (end-to-end). The protocol is not suited for multihop communications since the cryptographic primitives make use of algorithms with a 1-to-1 communication. The protocol can be implemented (or added) in order to provide secure communication. Note that it provides transport level security, as opposed to persistent data protection. One example for this addition to an already existing protocol is the hypertext transfer protocol. It can be used without SSL/TLS (http) or with SSL/TLS (https). The latter provides the desired transport layer security. The security goals of the SSL/TLS protocol can be divided into four groups [5]:

1. Cryptographical security: TLS is used to guarantee secure communication between two entities.

2. Interoperability: It should be possible for different programmers to develop applications that use TLS (or exchange cryptographical parameters) without explicitly knowing the code of the other application.

3. Extensibility: TLS builds a framework that can be extended by new cryptographical algorithms and primitives easily.

4. Relative efficiency: Cryptographic algorithms can be very CPU consuming, especially public key systems. For this reason TLS has implemented additional options to dynamically reduce network activity by reducing the number of connections.

Thus we can say that the SSL/TLS protocol is a very flexible protocol: Many parameters are chosen dynamically, and the extensibility provides a flexible framework for future developments.

The SSL/TLS protocol is divided into 5 subprotocols: the Record protocol, the Handshake protocol, the ChangeCipherSpec protocol, the Alert protocol and the Application Data protocol.

Since SSL/TLS is a multi-layer protocol, these subprotocols are distributed between two layers in the following way:

The Record protocol provides end-to-end communication with symmetric encryption algorithms, that are already negotiated in the Handshake protocol. Additionally one can use MACs and other primitives to ensure integrity and authenticity.
Figure 4: Here we can see the embedding of the protocols in the OSI-Layers. The record protocol is lower layered, since the other 4 protocols need it for their execution. Note that the TLS-Application protocol is a multilayer protocol and operates on layers 5-7.

The Handshake protocol is the negotiating instance. It uses public key encryption for authentication of the entities. In the next step specific keys and desired algorithms are negotiated.

The ChangeCipherSpec protocol consists only of 1 byte and can optionally and dynamically change the cipher suite, which was negotiated in the handshake protocol. (We will see later in more detail how a cipher suite looks like.)

The Alert protocol is a very important tool to cope with errors. In practice errors are simply not avoidable and therefore constitute a very important aspect when constructing a protocol. There are tons of ways the communication can be jeopardized. The Alert protocol specifies each error type (warning types that can occur).

The last subprotocol is the application data protocol, where the actual communication takes place. Only when then cipher suite is successfully negotiated and the communication starts, the application data protocol transfers the desired payload.

### 3.3 TLS-Record Protocol

The TLS-Record Protocol is a multilayer protocol, which collects information about the message in each layer. These information include length, description and content. Since big blocks of information cannot be transmitted arbitrarily, the TLS-protocol receives the message to be sent and splits it into blocks, which are compressed (optionally), hashed and encrypted. Only then the message is ready to be sent. The receiver has to invert this
process: decrypt, verify, decompress and reassemble. This is shown in the following figure:

---

**Figure 5**: Based on [5] and [6]. The sender has to split the message into equal blocks, compress them, hash and encrypt them afterwards. The receiver has to do the inverse operations in reversed order to get the plaintext.

The protocols Handshake protocol, ChangeCipherSpec protocol, Alert protocol and the Application Data protocol use the Record protocol and are described in this section. But first we want to take a look at how the Record protocol looks like in detail.

**Fragmentation:**

The record layer receives data from higher layers and splits them into blocks of maximum size of $2^{14}$ bytes or less. The information contained in a record include the protocol version, the content type and the length:

---

**Figure 6**: One record consists of: 1 byte content type, 2 bytes protocol version, 2 bytes payload length and depending on the content type different payload fragments. In the case of a plaintext fragment one has $2^{14}$ bytes of payload. An optional compression algorithm should not exceed $2^{14} + 2^{10}$ bytes and a compress ciphertext (or ciphertext) should not be more than $2^{14} + 2^{11}$ bytes. Based on [5] and [6].
Compression:
All records can optionally be compressed with the compression algorithm which was ne-
gotiated before (The default compression algorithm is set to be Compressionmethod.null, for more on compression algorithms in TLS see [3]) A compression algorithm is required to be lossless and not increase the length for more that $2^{10}$ bytes. (Otherwise an "fatal decompression failure error" is reported, see Alert protocol).

MAC and Encryption:
Applying the MAC function and the encryption translates a compressed message into a ciphertext message, the length of which may not exceed $2^{14} + 2^{11}$ bytes in total. To avoid missing, extra or repeated messages, the MAC adds a sequence number. MAC functions are basically hash functions with an additional checksum. A more detailed view of encryption algorithms are given in the "ChangeCipherSpec protocol" section.

3.4 ChangeCipherSpec protocol
The ChangeCipherSpec protocol is a single 1-byte message. Its content is just "1". This tells the receiver that the cipher suite is being changed, according to the negotiations in the Handshake protocol. A cipher suite is a collection of cryptgraphical algorithms. Each cipher suite specifies four algorithms that have to be used for secure communication for the following aspects of secure communication:

1. Key exchange,
2. Authentication,
3. Hash functions,
4. Encryption.

In this subsection we will discuss one algorithm for each of these categories. Note that these are only examples to illustrate how the TLS protocol works.[7]

3.4.1 Diffie-Hellman
The first primitive to analyze is a convinient algorithm to exchange secrets without using symmetric key cryptography. As mentioned above, Diffie and Hellman solved the key dis-

tribution problem by developing an algorithm which uses the Discrete Logarithm Problem to exchange information without sharing a secret beforehand.
Two communication partners Alice and Bob want to communicate via an unsecured medium with symmetric key primitives. They need to agree upon a shared secret using this unsecured medium. Hence, a possible attacker should not be able to figure out this shared secret hijacking the communication. The idea of the protocol to use the Discrete Logarithm Problem can be extended to elliptic curves, yielding the ECDH (Elliptic Curve Diffie Hellman). Possible measures against the ECDLP (Elliptic Curve Discrete Logarithm Problem) are considered in the last chapter.

The Diffie Hellman Key exchange algorithm works as follows:

1. Alice and Bob agree upon the public parameters: a cyclic group $G$ of prime order $p$ and a generator $g$ for this group.

2. Both Alice and Bob randomly select an element in $G^*$. Alice selects $a \in G$, $a \neq 0$ and Bob selects $b \in G$, $b \neq 0$ respectively. Note that for big $p$, $a = b$ is very unlikely. The values $a$ and $b$ will be the private parameters, unknown to the attacker.

3. Alice and Bob compute $A = g^a \mod p$ and $B = g^b \mod p$ respectively. Alice sends $A$ to Bob and Bob sends $B$ to Alice.

4. Having received $B$, Alice can compute $K = B^a \mod p$. Bob computes $K = A^b \mod p$ analogously. It turns out that the values they compute are both equal $K$.

The last statement can be shown by:

\[
A^b \mod p = (g^a \mod p)^b \mod p
= g^{ab} \mod p
= g^{ba} \mod p
= (g^b \mod p)^a \mod p
= B^a \mod p
\]

A possible attacker could acquire both the public parameters as well as the information sent. So in this situation assume he knows $G$, $g$, $p$, $A$ and $B$. The private information he wants to get is the value of $K$. One way to find out $K$ would be able to acquire $a$ and $b$ (the private values). That would mean that he has to solve

\[
A = g^a \mod p.
\]
That is, solving the Discrete Logarithm Problem. Hence that knowing how to solve Discrete Logarithm Problems implies solving the Diffie Hellman key exchange. Since acquiring a and b is the only known way to find out K, it is conjected that these two problems are equivalent. This is, however, not proven yet. In 1994 Uli Maurer proved that the Diffie Hellman Problem and the DLP are equivalent under certain assumptions. [8]

3.4.2 RSA

Tackling the second point of our cipher suite, we need a public key encryption scheme with which one can construct an authentication mechanism as seen in chapter 2. RSA was developed in 1978 by Rivest, Shamir and Adleman and was the first public key encryption and signature scheme. Recall that public key mechanisms are based on a public key and a private key for each party using trapdoor one-way functions. The RSA Algorithm gives a recipe how to construct private and public keys for a party. The public key is a pair \((e, N)\) of two numbers. The private key is a pair \((d, N)\) of two numbers where \(N\) is the same as in the public key. It is called the RSA-module. The number \(e\) is used for encryption and \(d\) for decryption. The construction goes as follows:

1. Randomly select two prime numbers \(p, q\) with \(p \neq q\).
2. Compute \(N = p \cdot q\).
3. Compute the Euler-\(\varphi\)-function \(\varphi(N) = (p-1)(q-1)\).
4. Randomly choose \(e \in \{2, \ldots, \varphi(N) - 1\}\) with \(\gcd(e, \varphi(N)) = 1\).
5. Compute \(d\) such that \(e \cdot d \equiv 1 \mod \varphi(N)\).

The numbers \(p, q\) and \(\varphi(N)\) can now be deleted. In practice it is very useful to have \(p\) and \(q\) of approximately the same size. The reason is that this makes it harder for an attacker to decompose \(N\) into its primefactors. In practice, random pairs of numbers of a specified size are generated until they are prime. Furthermore \(e\) is often selected to be the fermat prime \(2^{16} + 1 = 65537\). The attacker would have to use \(e\) and \(N\) to find out \(d\). That is, he has to solve the equation

\[ e \cdot d \equiv 1 \mod \varphi(N). \]

This means that he has to compute \(\varphi(N)\). Since the only known way to compute \(\varphi(N)\) is \(N = (p-1)(q-1)\) the attacker needs the primefactors \(p\) and \(q\). That reduces RSA-systems
to the Factorization Problem. This system can also be used for authentication as seen in chapter 2.

3.4.3 MD-5

The Message Digest-5 algorithm is a cryptographic hash function developed in 1991 by Ronald L. Rivest [9], one of the developers of the RSA algorithm. The 5 in its name stands for the version: Analysis showed that the MD-4 algorithm did not provide collision freeness as required by cryptographic Hash Functions.

The algorithm takes a message of arbitrary length and returns a 128-byte message. MD-5 consists of five steps:

1. Append padding bits
2. Append length
3. Initialize MD-Buffer
4. Process message in 16-word blocks
5. Output

Note that in this context a "word" is a block of 4 bytes. We suppose that the input message is $b$ bits long, $b \in \mathbb{N}$. The message is then $m_0m_1 \ldots m_{b-1}$.

1. **Append padding bits** The algorithm needs a 512-bit length (or a multiple of 512) message and in Step 2 one adds 64-bits. Therefore on extends the message such that it is 64 bits away from being a multiple of 512. For this step, a single "1" is appended to the message and zeros are added until $448 \mod 512$ is reached.

2. **Append length** In this step one adds a 64-bit representation of the length $b$ to the message. We end up with a message of length multiple of 512. This is, in other words, a multiple of 16 (32-bit) words. Let $M[00 \ldots N - 1]$ denote the words of the resulting message, $N$ being a multiple of 16.

3. **Initialize MD-Buffer** Four words $A, B, C$ and $D$ are initialized for the message digest.
word $A: 01\ 23\ 45\ 67$
word $B: 89\ ab\ cd\ ef$
word $C: fe\ dc\ ba\ 98$
word $D: 76\ 54\ 32\ 10$

4. **Process message in 16-word blocks** Now we have to define four functions that get as an input three 32-bit words (such as $A, B, C$ and $D$).

$$F(X, Y, Z) = XY \lor XZ$$
$$G(X, Y, Z) = XZ \lor Z$$
$$H(X, Y, Z) = X \oplus Y \oplus Z$$
$$I(X, Y, Z) = Y \oplus (X \lor Z)$$

The next step is processing each 16-word block. For each block, the algorithm goes through 5 rounds, (similar to the Feistel Rounds in DES). Each round is specifically defined by 16 operations. Each round uses one function out of $F, G, H$ and $I$:

We define the operation

$$[abcd\ k\ s\ i] := (a = b + ((a + F(b, c, d) + X[k] + T[i]) \ll s)).$$

Here $T$ is a table of 64 values, which can be found in [9]. $T[i]$ denotes the integer part of

$$|\sin(i)|.$$

Furthermore $X[k]$ is set to be $M[i \cdot 16 + k]$ for each block $i$. The operator $\ll s$ indicates a shifting of $s$ to the right. The first round consists of the following 16 operations:
The following 4 rounds will look similar, using the function \( G, H \) and \( I \) instead of \( F \) in the definition of the function. Also the specifically designed 16 operations in each round are optimized.

It combines the preliminary defined values of \( A, B, C \) and \( D \) in order to alter them in every round. In the end of the fifth round, one ends up with new values for \( A, B, C \) and \( D \).

5. The Algorithm returns \( A, B, C, D \).

### 3.4.4 RC4

RC4 is a streamcipher, which was developed by Ronald Rivest in 1987. RC4 (Ron’s Code 4 or Rivest Chipher 4) has been known to be outdated for a long period of time. Its implementation in TLS, however, remained until February 2015, when the IETF forbid further use of RC4 in TLS cipher suits. [10]

The general Idea of RC4 is to generate a random sequence using an only one-time key. It may only be used once since each key uniquely generates a random sequence. This key has to be of length \( \leq 256 \) and the plaintext will be XORed bitwise with the random sequence to get the cipher text. The generation of this random sequence uses S-Boxes and random permutation and substitutions.

RC4 consists of two steps: First the S-Box has to be initialized and second the random sequence has to be determined using this S-Box:

1. **S-Box Initialization:** The S-Box is an array of length 256-bits. The \( i \)-th bit of the S-Box has the initialized entry \( i \). Now all these entries are shuffled around in the following way: One variable \( i \) is iterated from 0 to 255. Another variable is initialized with \( j = 0 \). This variable \( j \) will be changed in each step with a certain formula. Then the entries \( S[j] \) and \( S[i] \) are swapped. The iteration formula for \( j \) is
\[ j = (j + S[i] + k[i \mod L]) \mod 256, \]

where \( L \) is the length of the key and \( k[\ ] \) denotes the key itself.

2. **Pseudo-Random Sequence Computation:**

While both \( i \) and \( j \) are initialized as \( i = j = 0 \), \( i \) will be incremented normally \( i = i+1 \mod 256 \). The incrementation of \( j \) on the other hand is \( j = j + S[i] \mod 256 \). Note that here the \( i \) will always be incremented first.

Next \( S[i] \) and \( S[j] \) will be swapped.

![Diagram](image.png)

This figure illustrates the next step: One looks up the entry \( S[i] + S[j] \), that is

\[ S[S[i] + S[j]] \mod 256. \]

This value is XORed with the first entry plaintext sequence. This will be repeated bit by bit until all the plaintext sequence bits are XORed with the resulting random sequence.

Since this is a symmetric key stream cipher one can use the same key for decryption: the random sequence computed is deterministic and thus one can XOR the same random sequence for decryption.
3.5 TLS-Handshake Protocol

The Handshake protocol is the most important tool in the SSL/TLS protocol. Its purpose is the negotiation of important parameters and the identification of the parties. It specifies in detail how these negotiations take place and in which order they are agreed upon. The messages are embedded in the above described records and can be protected with cryptographical tools. Since the cryptographic tools are negotiated in the handshake protocol itself, the initial communication is unprotected for obvious reasons. This unsecured connection remains stable until the ChangeCipherSpec protocol specifies a change of the cipher suite.

The Handshake protocol itself can be divided into four phases, where the second phase is optional.

**PHASE 1**

The first phase provides the negotiation of important parameters. The Client generates a random number and appends it to a Client Hello message. This Client Hello message contains important parameters such as the TLS-Version, a timestamp to avoid replay attacks, a session-ID and all cipher suites the client supports. The cipher suite specifies how the key exchange should take place and what symmetric cryptosystem should be used. The Client sends this Client Hello message to the Server. The idea is that the Client reveals its capabilities and the Server specifies the communication parameters. The server also generates a random number and appends it to a Server Hello message which specifies one cipher suite, considering the supported algorithms of the Client and sends it to the Client.
A Client Hello message and its counterpart the Server Hello message have the following form:

<table>
<thead>
<tr>
<th>Client_Hello</th>
<th>Server_Hello</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_TLS_version;</td>
<td>server_TLS_version;</td>
</tr>
<tr>
<td>random_number;</td>
<td>random_number;</td>
</tr>
<tr>
<td>session_id;</td>
<td>session_id;</td>
</tr>
<tr>
<td>cipher_suites;</td>
<td>cipher_suite;</td>
</tr>
<tr>
<td>compression_methods;</td>
<td>compression_method;</td>
</tr>
</tbody>
</table>

**PHASE 2**

In the second phase the Server can authenticate itself for the Client. He sends its certificate the personal public key to the Client. The Client can now verify this certificate. This phase can only be skipped in an anonymous key agreement, where the Client does not require the Server to authenticate itself.
PHASE 3

Here the Client authenticates itself for the Server. It sends its certificate with its public key to the Server. The Server checks the certificate. Now all previous messages will be sent to the Server encrypted with its private key. Since the Server now knows the public key, it can verify its signature. Using the servers public key it received earlier, it will randomly generate a number and encrypt it with the Servers public key. This random number is called the Pre-Master-Secret. The Server decrypts the Pre-Master-Secret with its private key. Both parties now have the Pre-Master-Secret and the random numbers generated in Phase 1. They both compute now a Mastersecret, which will be symmetric key for further communication. The computation of the mastersecret from the Pre-Master-Secret is the same for all encryption schemes: A Pseudo-Random-Function (PRF) is applied to the random numbers exchanged in the Hello messages in Phase 1 as well as the Pre-Master-Secret. This gives a shared mastersecret, since they already exchanged the necessary values (The randomly generated numbers as well as the Pre-Master-Secret). In this step a public key theme is used to negotiate the Mastersecret. The Diffie Hellman key exchange algorithm can be used alternatively. This was specified in the Chapter 2. In this case, the PRF will have minor changes.
The ChangeCipherSpec protocol now indicates that further communication will take place with the agreed upon symmetric key protocol with the computed Mastersecret key. First the Client, then the Server switch to an encrypted communication and end the handshake.
### 3.6 Alert Protocol

One of the protocols that can use records is the Alert protocol. The Alert type is supported by the record layer. An Alert message indicates the severity of the alert (warning or fatal) and a description of the alert. An Alert message has the structure

```
struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

![Code taken from [5].](image)

where

```
enum {
    warning(4),
    fatal(2),
    (255)
} AlertLevel;
```

```
enum {
    close_notify(0),
    unexpected_message(10),
    bad_record_mac(20),
    decryption_failed_RESERVED(21),
    record_overflow(22),
    decompression_failure(30),
    handshake_failure(40),
    no_certificate_RESERVED(41),
    bad_certificate(42),
    unsupported_certificate(43),
    certificate_revoked(44),
    certificate_expired(45),
    certificate_unknown(46),
    illegal_parameter(47),
    unknown_ca(48),
    access_denied(49),
    decode_error(50),
    decrypt_error(51),
    export_restriction_RESERVED(60),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    user_canceled(90),
    no_renegotiation(100),
    unsupported_extension(110),
    (255)
} AlertDescription;
```

![Code taken from [5].](image)

One problem occurring when dealing with TLS is the process of ending a communication. An attacker may block a victim’s logout request so that the user unknowingly is kept logged in. This is called a truncation attack. In order to avoid this attack the exchange of a closing message is required: Either party can initiate a termination of the communication by sending a close notify alert. The other party receives this close notify alert and realizes that any further communication will be ignored. Before ending the session itself, it has to
send a close notify as well. That having said, it is not necessary for the initiator to wait for the close notify response of the other party.
All other AlertDescriptions (except for the close notify alert) indicate an error. Whenever an error occurs, the party that detected the error notifies the other party with an error alert. Each error is categorized by its AlertLevel, Fatal or Warning. Whenever a fatal error occurs, both parties immediately have to close the connection. All information of the connection has to be deleted by clients and servers (Such as keys, secrets, etc.). It may occur that the AlertMessage does not explicitly say whether an error was a warning or fatal. In this case, the receiving party may treat this how he wants (In his personal implementation). If, however, the detecting party wants to end the communication after sending an error alert, it has to explicitly identify the alert as a fatal error.
In the case of a warning, mostly the communication continues smoothly. The receiver is now informed by the error and can individually decide whether to continue with this warning or not. If not, he must send a fatal error himself, in order to close the communication. This is a problem: If the sender wants to continue the conversation although he has sent a warning, he does not know if the receiver will behave similarly. For this reason warnings are sometimes omitted.

3.7 Application Data Protocol
The Application Data protocol is the last protocol in the hierarchy. After the handshake was successful, the actual communication will start. This protocol contains the actual payload data.
4 Elliptic Curve Cryptography

4.1 Why Elliptic Curve Cryptography?

Before starting with the theory of elliptic curves and its applications to cryptography, it is important to understand why this topic is essential. Elliptic curves give, as we will see in this chapter, a very convenient group structure. We can use this structure when applying elliptic curves to our developed cryptographic methods. Instead of using finite fields as in RSA for example, we will use groups derived by elliptic curves. This begs the question: Does this help us at all? And the answer is yes. Cryptosystems based on RSA are relying on the hardness of the Discrete Logarithm Problem. It turns out that the same problem with elliptic curves, the ECDLP (Elliptic Curve Discrete Logarithm Problem) is even harder to solve. Currently the best known algorithms solving the ECDLP have full exponential running time. This means that the level of security given by an elliptic curve system with a small key, provides the same security as a RSA system with a big key. It is generally known that a 160-bit elliptic curve key has the same level of security as a 1024-bit RSA key. However, it turns out that certain elliptic curves are more vulnerable to the known ECDLP attacks than others. This makes the choice of the "right" elliptic curve of huge importance. All in all, elliptic curve cryptography is a topic worth investigating.

4.2 Introduction to Elliptic Curves

Definition 4.1. An elliptic curve \( E \) is the graph of an equation given by

\[
y^2 = x^3 + Ax + B
\]

where \( A \) and \( B \) are constants. This equation is called the Weierstrass equation for an elliptic curve.

Usually \( A, B, x \) and \( y \) are elements of a field. Usually \( \mathbb{R}, \mathbb{C}, \mathbb{Q} \) or finite fields \( \mathbb{F}_p \) for a prime \( p \). (Or more generally finite fields of the form \( \mathbb{F}_q \) with \( q = p^k, k \geq 1, p \) prime.) If \( K \) is a field with \( A, B \in K \), we say that the elliptic curve \( E \) is defined over \( K \).

For technical reasons we will add the point \((\infty, \infty)\) denoted by \( \infty \) to our elliptic curve. This will give us some nice properties we will use later on. It can be seen as a formal symbol satisfying certain properties.
Figure 10: These are two examples of elliptic curves (a) and (b) over $\mathbb{R}$. [11]

It turns out that on these elliptic curves we can define a very nice group structure. In this group we can perform our cryptographic methods we have seen in RSA and others.

**Group structure of elliptic curves**

We will first formally define the group structure and then describe its geometrical meaning. It does not always make sense to try to plot a graph of an elliptic curve. That is why it is convenient to think of elliptic curves over $\mathbb{R}$.

**Definition 4.2.** Let $E$ be an elliptic curve defined by $y^2 = x^3 + Ax + B$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E$ with $P_1, P_2 \neq \infty$. Define $P_3 = (x_3, y_3)$ as follows:

1. If $x_1 \neq x_2$, then $x_3 = m^2 - x_1 - x_2, y_3 = m(x_1 - x_3) - y_1$ where $m = \frac{y_2 - y_1}{x_2 - x_1}$

2. If $x_1 = x_2$ and $y_1 \neq y_2$, then $P_1 + P_2 = \infty$

3. If $x_1 = x_2$ and $y_1 = y_2 \neq 0$ (That means $P_1 = P_2$) then $x_3 = m^2 - 2x_1, y_2 = m(x_1 - x_3) - y_1$, where $m = \frac{3x_1^2 + A}{2y_1}$

4. If $P_1 = P_2$ and $y_1 = 0$, then $P_1 + P_2 = \infty$

5. $P + \infty = P \ \forall P$

Adding two points of an elliptic curve basically means that you draw the line through two points, mirror the resulting intersection point at the x-axis. All cases in definition 4.2 result out of special locations of the points.
Figure 11: Addition of two points on an elliptic curve: Add $P_1$ and $P_2$ by intersecting their line with the curve and mirror the result on the $x$-axis. [11]

If $P_1 = P_2$ you take the tangential line through the point. If the two points yield a parallel line to the $y$-axis, we define it to intersect the elliptic curve in $\infty$.

**Theorem 4.3.** The addition of points on an elliptic curve $E$ satisfies the following properties:

1. (commutativity) $P_1 + P_2 = P_2 + P_1 \forall P_1, P_2$ on $E$

2. (existence of identity) $P + \infty = P \forall P$ on $E$

3. (existence of inverse) Given $P$ on $E$, there exists $P'$ on $E$ with $P + P' = \infty$. This point $P'$ is denoted as $-P$.

4. (associativity) $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \forall P_1, P_2, P_3$ on $E$.

All in all: The points on $E$ form an additive abelian group with $\infty$ as the identity element.

**Proof.** 1. The commutativity is obvious since the line through the points $P_1$ and $P_2$ is the same as the line through $P_2$ and $P_1$.
2. This equation holds by definition.

3. Reflecting each point $P$ on the x-axis gives a point $P'$ with $x = x'$ and $y \neq y'$. Then $P + P' = \infty$.

4. The proof of the associativity can be verified by using the explicit formulas of the definition. There are many cases which have to be considered (whether $P_1 = P_2$ or not etc.). The proof can be seen in [12].

### 4.2.1 Torsion Points

We will now focus on so called Torsion Points. These are points on an elliptic curve which satisfy a specific property. One can start with an arbitrary point on an elliptic curve $P$ and add it to itself. The resulting point is $P + P$, denoted by $2P$. Repeatedly adding $P$ gives us a series of points on the elliptic curve. The number $n$, for which we get $nP = \infty$, is called the order of the point $P$.

**Definition 4.4.** An $n$-Torsion point is a point $P$ with $nP = \infty$. The set of $n$-Torsion points is denoted as $E[n] = \{P \in E(\overline{K}) : nP = \infty\}$ with $\overline{K}$ being the algebraic closure of $K$.

For the following we are interested in finite $n$-Torsion points.

**Example 4.5.** Let $K$ be a field with $\text{char}(K) \neq 2$. We are interested in the set $E[2]$ over $K$. We can describe the curve $E$ via the equation $y^2 = x^3 + (\ldots)$. That means that $y^2$ is a cubic function in $x$. We can write

$$y^2 = (x - e_1)(x - e_2)(x - e_3),$$

with $e_1, e_2, e_3 \in \overline{K}$. A point satisfies $2P = \infty$ if and only if the tangent is parallel to the $y$-axis. Obviously this means $y = 0$. This yields the set $E[2] = \{\infty, (e_1, 0), (e_2, 0), (e_3, 0)\}$.

As a group this is isomorphic to $\mathbb{Z}_2 \oplus \mathbb{Z}_2$. 

This result can be generalized with more theory on elliptic curves. We will skip the proof here, since we will only use the result in the analysis of the Weil Pairing.

**Theorem 4.6.** Let $E$ be an elliptic curve over a field $K$ and let $n$ be a positive integer. If the characteristic of $K$ does not divide $n$, or is 0, then

$$E[n] \simeq \mathbb{Z}_n \oplus \mathbb{Z}_n$$

If the characteristic of $K$ is $p \geq 0$ and $p \mid n$, write $n = p^r n'$ with $p \nmid n'$. Then

$$E[n] \simeq \mathbb{Z}_{n'} \oplus \mathbb{Z}_{n'}$$

or

$$E[n] \simeq \mathbb{Z}_{n} \oplus \mathbb{Z}_{n'}.$$

**Proof.** See [11] 3.2 S.76-82. \hfill \Box

### 4.3 Elliptic Curves over finite Fields

We will now start considering elliptic curves over finite fields. Some aspects are completely different in the case of finite fields. This also takes us a step further to elliptic curve cryptography. Elliptic curves defined over finite fields also give rise to problems as the Elliptic Curve Discrete Logarithm Problem. Since we are using finite fields, the elliptic curve over the finite field also has finite cardinality. Therefore it does not make sense to try to plot the curves. However, one can imagine the elliptic curve to be a finite set of points on a finite lattice structure. The addition arithmetics (definition 4.2) hold nevertheless.

**Example 4.7.** Let $E$ be the curve $y^2 = x^3 + x + 1$ over $\mathbb{F}_5$. To find the list of points of the curve, we compute for all values of $x$ the function $f(x) = x^3 + x + 1$ and see if the solution is a square in $\mathbb{F}_5$.

Let $x = 0$, then $f(x) = 1$. 1 is a square in $\mathbb{F}_5$ with solution $y = 1$ or $y = -1$. This gives us two points on the elliptic curve: $(0, 1)$ and $(0, -1) = (0, 4)$, respectively. We proceed with the same strategy for $x = 1, 2, 3, 4$ and get the points $(2, 1), (2, 4), (3, 1), (3, 4), (4, 2)$ and $(4, 3)$. $f(1) = 3$ is not a square in $\mathbb{F}_5$ and hence we don’t get any points on the curve for $x = 1$. Finally, we have to include infinity to obtain all points of the elliptic curve.

$$E = \{(0, 1), (0, 4), (2, 1), (2, 4), (3, 1), (3, 4), (4, 2), (4, 3), \infty\}.$$
Hence, the order of $E$ is 9. Now we want to show an example of how to add points on $E$ over finite fields. We will perform $(3, 1) + (2, 4)$. Remember that adding two points on a curve means intersecting the line through the two points with the curve and then reflecting it across the x-axis.

The slope through $(3, 1)$ and $(2, 4)$ is

$$\frac{4 - 1}{2 - 3} \equiv 2 \mod 5.$$ 

Hence we get the line through $(3, 1)$ and $(2, 4)$ by $y = 2(x - 3) + 1 = 2x - 6 + 1 \equiv 2x \mod 5$. We substitute $y$ in $E$ by $y = 2x$ and get

$$0 = x^3 - 4x + x + 1.$$

The generalized Theorem of Vieta for cubic functions states that $x^3 + a_2x^2 + a_1x + a_0$ with roots $x_1, x_2, x_3$ the following equation holds:

$$a_2 = -(x_1 + x_2 + x_3)$$

Using this in our case we get that the sum of the roots is 4. We already know the roots 2 and 3. We know this because the zeros of the function $x^3 - 4x + x + 1$ give exactly the intersections of the curve $E$ and the line $2x$.

That is why we can conclude that the last root is $x = 4$ which gives us the point $y \equiv 3 \mod 5$ on the curve. Reflecting the point $(4, 3)$ on the x-axis gives us $(4, 2)$. This is the solution of our sum:

$$(3, 1) + (2, 4) = (4, 2)$$

The last important thing to note is that elliptic curves over finite fields are isomorphic to the direct sum of two cyclic groups. That will help to guarantee the existence of specific necessary points later on, when we describe an attempt to solve the ECDLP.

**Theorem 4.8.** Let $E$ be an elliptic curve over the finite field $\mathbb{F}_q$. Then

$$E(\mathbb{F}_q) \simeq \mathbb{Z}_n$$

for some integer $n \geq 1$ or

$$E(\mathbb{F}_q) \simeq \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$$

for some integers $n_1, n_2 \geq 1$ and $n_1 \mid n_2$. 
Proof. From group theory we know that a finite abelian group (as \( E(\mathbb{F}_q) \)) is isomorphic to a direct sum of cyclic groups

\[
\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2} \oplus \cdots \oplus \mathbb{Z}_{n_r},
\]

with \( n_i \mid n_{i+1} \) for \( i \geq 1 \). For all \( i \) the group \( \mathbb{Z}_{n_i} \) has \( n_i \) elements of order dividing \( n_1 \). This means that \( E(\mathbb{F}_q) \) has \( n_1^r \) elements of order dividing \( n_1 \). Using Theorem 4.7 we see that there are at most \( n_1^2 \) such points. This means \( r \leq 2 \), which leaves us with the cases \( r = 1 \) and \( r = 0 \) which are both covered by the theorem. \( \square \)

4.4 The Weil Pairing

We now want to analyze the so called Weil Pairing. The Weil Pairing is a function which maps from the torsion points of order \( n \) to the \( n \)-th roots of unity. This pairing function has some nice properties. In this section we first develop a useful tool (Divisors) and then formulate the Weil Pairing formally. The proof for the properties of the Weil Pairing is important when we try to attack the ECDLP for special elliptic curves.

4.4.1 Weil Divisors

Definition 4.9. For each point \( P \in E(\overline{K}) \) we define a formal symbol \([P]\). A divisor \( D \) on \( E \) is defined as

\[
D = \sum_j a_j [P], a_j \in \mathbb{Z}.
\]

So \( D \) is a finite linear combination of such symbols with integer coefficients. It is easy to see that the set of divisors build a group which we denote as \( \text{Div}(E) \).

Definition 4.10. The degree function assigns an integer number to every divisor. Formally it is a function

\[
\deg : \text{Div}(E) \rightarrow \mathbb{Z}, D = \sum_j a_j [P_j] \mapsto \sum_j a_j.
\]

Similarly the sum function assigns a point on the Elliptic curve \( E \) to every divisor simply by adding the points of the symbol with the group law of the curve:

\[
\text{sum} : \text{Div}(E) \rightarrow E(\overline{K}), D = \sum_j a_j [P_j] \mapsto \sum_j a_j P_j.
\]
In other words, the degree function adds up all the integer values in the sum of the divisor and the sum function adds the points on the elliptic curve. The next step is defining functions on an elliptic curve $E$ and defining divisors for these functions.

**Definition 4.11.** A function on an elliptic curve $E$ is a rational function

$$f(x, y) \in \mathbb{K}(x, y),$$

that is defined in at least one point in $E(\mathbb{K})$. It takes its values in $\mathbb{K} \cup \{\infty\}$.

To show how to work with these functions we will give an example.

**Example 4.12.** Let $E$ be an elliptic curve defined by the function $y^2 = x^3 - x$. We define a function $f$ on $E$ by

$$f(x, y) = \frac{x}{y}.$$

This function is obviously not defined at $(0, 0)$. But we can rewrite the elliptic curve $E$ in such a way that one side is exactly the function $f$.

$$y^2 = x^3 - x \rightarrow \frac{x}{y} = \frac{y}{x^2 - 1}.$$

Interpreting the function $f$ on $E$ means that it is now defined at the point $(0, 0)$ and takes the value $0$. Similarly we can look at the function $f(x, y) = \frac{y}{x}$ and get

$$\frac{y}{x} = \frac{x^2 - 1}{y}.$$

In this case the function $f(x, y) = \frac{y}{x}$ is defined at point $(0, 0)$ and takes value $\infty$.

More generally it can be shown that there is always a way to transform a function such that we obtain an expression that is not $0/0$ and therefore a uniquely determined value in $\mathbb{K} \cup \{\infty\}$.

**Definition 4.13.** A function $f$ on an elliptic curve $E$ is said to have

1. a zero at point $P$ if it takes the value $0$ at $P$.
2. a pole at point $P$ if it takes the value $\infty$ at $P$. 

**Definition 4.14.** Let $f$ be a function on an elliptic curve $E$. A uniformizer at $P$ is a function $u_P$ such that $u_P(P) = 0$ and that every function $f(x,y)$ on $E$ can be written in the form

$$f = u_P^r g$$

with $r \in \mathbb{Z}$ and $g(P) \neq 0, \infty$. It can be shown that such a uniformizer function always exists. Furthermore we define the order of $f$ at $P$ to be

$$\text{ord}_P(f) = r.$$

**Example 4.15.** Let $E$ be the elliptic curve as in Example 4.12. This also can be written as

$$x = y^2 \frac{1}{x^2 - 1}.$$ 

The function $\frac{1}{x^2 - 1}$ is nonzero and finite at $(0,0)$. In this case $y$ is a uniformizer at $(0,0)$ with

$$\text{ord}_{(0,0)}(x) = 2 \text{ and } \text{ord}_{(0,0)}(x/y) = 1.$$

We now define divisors for functions on $E$. These are called principal divisors and are of important use in the proof of the properties of the Weil Pairing.

**Definition 4.16.** Let $f$ be a function on an elliptic curve $E$. A divisor for $f$ is defined as

$$\text{div}(f) = \sum_{P \in E(K)} \text{ord}_P(f)[P].$$

The divisor of a function is also called a principal divisor.

**Theorem 4.17.** Let $E$ be an elliptic curve and let $f$ be a function on $E$ that is not identically 0. Then the following statements hold.

1. $f$ has only finitely many zeros and poles.
2. $\deg(\text{div}(f)) = 0$.
3. $f$ has no zeros or poles ($\text{div}(f) = 0$) $\Rightarrow$ $f$ is constant.

In other words: The sum of $\text{div}(f)$ is finite and thus a divisor (an element in $\text{Div}(E)$).

**Proof.** See [14].
Example 4.18. Let $f(x, y) = ax + by + c$ with zeros $P_1, P_2, P_3 \in E$. The function $f$ has zeros on $P_1, P_2$ and $P_3$ and a triple pole at $\infty$. That means

$$\text{div}(f) = [P_1] + [P_2] + [P_3] - 3[\infty].$$

The line through $P_3 = (x_3, y_3)$ and $-P_3 = (x_3, -y_3)$ is $x = x_3 \Leftrightarrow x - x_3 = 0$. Its divisor is

$$\text{div}(x - x_3) = [P_3] + [-P_3] - 2[\infty]. \quad (4.19)$$

That means

$$\text{div} \left( \frac{f}{x - x_3} \right) = \text{div}(f) - \text{div}(x - x_3) = [P_1] + [P_2] - [-P_3] - [\infty].$$

Since $P_1 + P_2 + P_3 = \infty$ on $E$, we can rewrite that as

$$[P_1] + [P_2] = [P_1 + P_2] + [\infty] + \text{div} \left( \frac{f}{x - x_3} \right).$$

This generally means that we can rewrite a sum of symbols $[P_1] + [P_2]$ as $[P_1 + P_2] + [\infty] + \text{div}(g)$, $g$ being a function, of which we need to take the divisor. We will use this important property to proof the following theorem.

Before proving the main theorem of this section we need a lemma first. The proof is long and does not use tools we need to proceed to the Weil Pairing.

Lemma 4.20. Let $P, Q \in E(\overline{K})$ and suppose there is a function $h$ on $E$ with

$$\text{div}(h) = [P] - [Q].$$

Then $P = Q$.


Theorem 4.21. Let $E$ be an elliptic curve. Let $D$ be a divisor on $E$ with degree 0. Then

$$\exists f \text{ on } E \text{ with } \text{div}(f) = D \iff \text{sum}(D) = \infty.$$

Proof. Considering the function $g$ we saw in Example 4.18, the following equation holds

$$\text{sum} \left( \text{div}(g) \right) = P_1 + P_2 - (P_1 + P_2) - \infty = \infty.$$
Furthermore because of Equation 4.19 we know that \([P_1] + [P_2] = 2[\infty] + \text{div}(g_1)\) for a function \(h\) when \(P_1 + P_2 = \infty\).
Hence, the sum of all the terms in \(D\) with positive coefficients can be written as \(P + n[\infty] + \text{div}(g_1)\) for an integer number \(n\) and a function \(g_1\). This means that \(\exists P, Q\) on \(E\), a function \(g_1\) and a integer number \(n\) such that
\[
D = [P] - [Q] + n[\infty] + \text{div}(g_1).
\]

\(g_1\) is the quotient of products of functions \(g\) with \(\text{sum}(\text{div}(g)) = \infty\). Furthermore because of \(\text{deg}(\text{div}(g_1)) = 0\) (by Theorem 4.17) we have \(n = 0:\n0 = \text{deg}(D) = 1 - 1 + n + 0 = n.
\)
Therefore,
\[
D = [P] - [Q] + \text{div}(g_1).
\]

On the other hand
\[
\text{sum}(D) = P - Q + \text{sum}(\text{div}(g_1)) = P - Q.
\]
If now \(\text{sum}(D) = \infty\) then also \(P - Q = \infty\) which means \(P = Q\) and \(D = \text{div}(g_1)\). If we suppose \(D = \text{div}(f)\) for some \(f\), then \([P] - [Q] = \text{div}(f/g_1)\). Because of Lemma 4.20 we know \(P = Q\) and thus \(\text{sum}(D) = \infty\). This completes the proof.

4.4.2 The Weil Pairing

The aim of this section is to formulate the Weil Pairing and proof its general properties. The Weil Pairing is a function from the \(n\)-Torsion points to the \(n\)-th roots of unity. The idea is to find a mapping which, if needed, can transfer problems with elliptic curves to problems in normal finite fields. This is used by the MOV-Attack which we will see later. Before starting with the Weil Pairing however, we state one last lemma without proof, to be able to proof the Weil-Pairings properties.

**Lemma 4.22.** Let \(E\) be an elliptic curve over a field \(K\). Let \(f(x, y)\) be function from \(E\) to \(\overline{K} \cup \{\infty\}\) and let \(n \geq 1\) be an integer with \(n \nmid \text{char}(K)\).
Suppose \(f(P + T) = f(P) \forall P \in E(\overline{K}), T \in E[n]\). Then \(\exists\) function \(h\) on \(E\) such that
\[
f(P) = h(nP) \forall P.
\]

We want to construct a pairing which assigns two given \(n\)-Torsion points to a uniquely determined \(n\)-th root of unity. That means we are looking for a function

\[ e_n : E[n] \times E[n] \rightarrow \mu_n \]

with \(\mu_n\) being the set of \(n\)-th roots of unity in \(\overline{K}\).

Let \(T\) be an \(n\)-Torsion point on \(E\), i.e. \(T \in E[n]\). By Theorem 4.21 there is a function \(f\) such that

\[ \text{div}(f) = n[T] - n[\infty]. \] (4.23)

Let now \(T' \in E\) with \(nT' = T\). That means \(T' \in E[n^2]\).

We want to show that there exists a function \(g\), such that

\[ \text{div}(g) = \sum_{R \in E[n]} ([T' + R] - [R]) = \sum_{nT'' = T} [T''] - \sum_{nR = \infty} [R]. \]

The sum of the points in the divisor is \(\infty\), because there are \(n^2\) points \(R\) in the \(n\)-Torsion points of \(E\). The points \(R\) in \(\sum[T' + R]\) and \(\sum[R]\) cancel each other, which leaves \(n^2T' = nT = \infty\). We can also see that \(g\) is independent of the choice of \(T'\). Any two choices for \(T''\) differ by an element \(R \in E[n]\). This is why the second equation holds. Let \(f_n\) be a function that multiplies a point by \(n\) and then applies \(f\). It follows from equation 4.23 that

\[ \text{div}(f_n) = n \left( \sum_{R} [T' + R] \right) - n \left( \sum_{R} [R] \right) = \text{div}(g^n), \]

which means that \(f_n\) is a constant multiple of \(g^n\). Without loss of generality we write \(f_n = g^n\) where \(f\) is implicitly multiplied by this constant to satisfy the equality. We are now almost ready to define our Weil Pairing mapping. Let \(S \in E[n]\) and let \(P \in E(\overline{K})\). Then

\[ g(P + S)^n = f(n(P + S)) = f(nP) = g(P)^n \implies \frac{g(P + S)}{g(P)} \in \mu_n. \]

**Definition 4.24** (Weil Pairing). Let \(S, T \in E[n]\) and \(P \in E(\overline{K})\). Then the Weil Pairing is a function

\[ e_n : E[n] \times E[n] \rightarrow \mu_n \]
with
\[ e_n(S, T) = \frac{g(P + S)}{g(P)}. \]

It turns out that this definition is independent of the point \( P \) and the choice of \( g \). In the following we prove the main properties of the Weil Pairing:

**Theorem 4.25.** Let \( E \) be an elliptic curve defined over a field \( K \) and let \( n \) be a positive integer. Furthermore we let \( \text{char}(K) \nmid n \). Then the Weil-Pairing
\[ e_n : E[n] \times E[n] \rightarrow \mu_n \]
satisfies the following properties:

1. \( e_n \) is bilinear. That is,
\[ e_n(S_1 + S_2, T) = e_n(S_1, T)e_n(S_2, T) \]
and
\[ e_n(S, T_1 + T_2) = e_n(S, T_1) + e_n(S, T_2) \]
for all \( S, S_1, S_2, T, T_1, T_2 \in E[n] \).

2. \( e_n(T, T) = 1 \forall T \in E[n] \).

3. \( e_n(T, S) = e_n(S, T)^{-1} \forall S, T \in E[n] \).

4. \( e_n \) is non-degenerate. That is,
\[ e_n(S, T) = 1 \forall T \in E[n] \implies S = \infty \]

and
\[ e_n(S, T) = 1 \forall S \in E[n] \implies T = \infty. \]

**Proof.**

1. **Linearity in the first variable:** We know that the Weil-Pairing is independent of the choice of \( P \). We use \( P \) and \( P + S_1 \) and get
\[ e_n(S_1 + S_2, T) = \frac{g(P + S_1 + S_2)}{g(P)} = \frac{g(P + S_1)}{g(P)} \frac{g(P + S_2)}{g(P + S_1)} = e_n(S_1, T)e_n(S_2, T). \]

**Linearity in the second variable:** Let \( T_1, T_2, T_3 \in E[n] \) with \( T_1 + T_2 = T_3 \). Recall how we defined the Weil Pairing and let the functions \( f_1, f_2, f_3 \) and \( g_1, g_2, g_3 \) be the
functions we used for the definitions for $e_n(S, T_i)$. Theorem 4.21 states that there is a function $h$, such that

$$\text{div}(h) = [T_3] - [T_1] - [T_2] + [\infty].$$

With Equation (4.23) we get

$$\text{div} \left( \frac{f_3}{f_1f_2} \right) = n \text{div}(h) = \text{div}(h^n).$$

Since the left side and the right side of the equation have to be the same, we can conclude that we can write $f_3$ as a constant multiplied by $f_1f_2h^n$. Let $c \in \mathbb{K}$ be this constant. Then $f_3 = cf_1f_2h^n$ gives us information about $g_3$. More specifically $g_3 = c^n g_1 g_2 (h \circ n)$. We put this into the definition of the Weil Pairing and get:

$$e_n(S, T_1 + T_2) = \frac{g_3(P + S)}{g(P)} = \frac{g_1(P + S) g_2(P + S) h(n(P + S))}{g_1(P) g_2(P) h(nP)} = \frac{g_1(P + S) g_2(P + S) h(nP)}{g_1(P) g_2(P) h(nP)} = \frac{g_1(P + S) g_2(P + S)}{g_1(P) g_2(P)} = e_n(S, T_1) + e_n(S, T_2).$$

2. Let $\tau_{jT}$ be the function $P \mapsto P + jT$. This means also $f \circ \tau_{jT}$ represents the function $P \mapsto f(P + jT)$. As above the divisor can be computed similarly and we get

$$\text{div}(f \circ \tau_{jT}) = n[T - jT] - n[-jT].$$

Adding these divisors for all $j$ we get

$$\text{div} \left( \prod_{j=0}^{n-1} f \circ \tau_{jT} \right) = \sum_{j=0}^{n-1} (n[(1 - j)t] - n[jT]) = 0.$$

Because of Theorem 4.17(3) we know that $\prod_{j=0}^{n-1} f \circ \tau_{jT}$ has to be constant. Furthermore we can reformulate this term in the following way:
\[ \prod_{j=0}^{n-1} f \circ \tau_j T = \prod_{j=0}^{n-1} f \circ n \circ \tau_j T' = (\prod_{j=0}^{n-1} g \circ \tau_j T')^n. \]

Since the left side of the equation is constant, the right side of the equation has to be constant as well. Particularly it has the same value for \( P \) and \( P + T' \). That is

\[ \prod_{j=0}^{n-1} g(P + T' + jT') = \prod_{j=0}^{n-1} g(P + jT'). \]

Equal terms on the left and the right side cancel each other out and leaving us with

\[ g(P + nT') = g(P). \]

Because of \( nT' = T \) we get

\[ e_n(T, T) = \frac{g(P + T)}{g(P)} = 1 \]

which completes the proof for (2).

3. This part is straightforward since we directly apply (1) and (2):

\[ 1 = e_n(S + T, S + T) = e_n(S, S)e_n(S, T)e_n(T, S)e_n(T, T) = e_n(S, T)e_n(T, S) \]

\[ \implies e_n(T, S)e_n(S, T)^{-1} \]

4. Let \( e_n(S, T) = 1 \ \forall S \in E[n] \). Using the definition of the Weil Pairing this means that \( g(P + S) = g(P) \) for all \( P \) and for all \( S \in E[n] \). Using Lemma 4.24 there has to exist a function \( h \), such that \( g(P) = h(nP) = h \circ n \). Then

\[ (h \circ n)^n = g^n = f \circ n. \]

We can omit the multiplication by \( n \) (the circle Operator is a surjection on \( E(\overline{K}) \)) and get \( h^n = f \). The divisor of \( f \) is \( n[T] - n[\infty] \) (4.25) and the divisor of \( h^n \) is
\[ n \text{ div}(h). \text{ This means} \]
\[ n \text{ div}(h) = n[T] - n[\infty] \iff \text{div}(h) = [T] - [\infty]. \]

Because of Theorem 4.21 we conclude \( T = \infty. \) This proves the first part of the non-degeneracy. The second part can be easily shown by using (3) and applying the non-degeneracy in \( T. \) This completes the proof.

Additionally it can be proven that \( S, T \in E(\mathbb{F}_{q^l}) \implies e_n(S, T) \in \mathbb{F}_{q^l}^*. \) [18]. We will denote this as the fifth property of the Weil Pairing. We can now analyze the possible attacks on the ECDLP and the MOV-Attack, which uses the Weil Pairing.

### 4.5 Elliptic Curve Discrete Logarithm Problem

Now that we took a deeper look at the Weil Pairing and its properties, we are almost ready to formulate an attack on the Elliptic Curve Discrete Logarithm Problem (ECDLP). We have already seen that elliptic curves over finite fields build a finite abelian group. For each elliptic curve over such a finite field we can use the already known cryptosystems, such as Diffie-Hellman key exchange etc. Furthermore we have seen that these systems are secure, if the Discrete Logarithm Problem is hard to solve. In this chapter we present four of attacks on the Elliptic Curve Discrete Logarithm Problem and how we have to choose \( E(\mathbb{F}_q) \) so that the ECDLP is hard to solve.

So let \( P \) be an \( n \)-Torsion point on an elliptic curve \( E(\mathbb{F}_q). \) For a point \( Q \in \langle P \rangle \) (the cyclic subgroup which is generated by \( P \)) we look for the number \( k \) such that

\[ kP = Q. \]

Obviously it is sufficient to look for \( k \mod n \). We call \( k \) the discrete logarithm of \( Q \).

After presenting the well known general attacks we will, at last, present the MOV-Attack, which uses the Weil Pairing to reduce the problem to solving the Discrete Logarithm Problem on roots of unity.

#### 4.5.1 Baby-Step-Giant-Step

Let \( m = \lceil \sqrt{n} \rceil \). We can write every \( k \) as \( k = qm + r, k \in \mathbb{N}, r \in \{0, 1, \ldots, m-1\}. \) Knowing \( q \) and \( r, k \) will be determined uniquely. This is why it suffices to compute \( r \) and \( k \). Since
\[ Q = kP = qmP + rP \] we obtain the following equation by subtracting \( rP \):

\[ Q - rP = qmP. \]

The idea of the algorithm is to compute a list of values of the left side of the equation and then compute all possible values of the right side of the equation. Then searching the list for matches will give the result. Computing the list of the values for the left side of the equation is called “baby-steps”. Similarly computing the right side of the equation is called “giant-steps”. The list of the baby-steps, which is computed first and saved, is then

\[ B = \{(Q - rP, r) : 0 \leq r < m \}. \]

If one of the values is \( Q - rP = \infty \), then \( r = k \) is the discrete logarithm we are looking for. If this is not the case for any \( r \), we have to go to the “giant-step”. The first “giant-step” is computing \( R = mP \) and looking if it coincides with an entry in \( B \). If not, we compute \( 2R, 3R, \ldots, (m - 1)R \) until we find a match. As soon as we found such \( qR \), the corresponding entry of \( B \) yields an \( r \) that satisfies the equation. This entry gives us an \( r \), which holds the equation. And we get the discrete logarithm \( k \) by setting

\[ k = qm + r. \]

### 4.5.2 Pohlig-Hellmann

The Pohlig-Hellmann algorithm reduces the computation of the discrete logarithm in the group \( \langle P \rangle \) of order \( n \) to the computation of discrete logarithms in subgroups of \( \langle P \rangle \). These subgroups will have the order of prime divisors. Let

\[ n = \prod_{i=1}^{t} p_i^{\lambda_i} \]

with \( p_i \) being its prime divisors and \( \lambda_i \geq 1 \). Given a \( Q \in \langle P \rangle \) we are again looking for a \( k \) with \( Q = kP \). Because of the Chinese Remainder Theorem we have

\[ \mathbb{Z}/n\mathbb{Z} \simeq \mathbb{Z}/p_1^{\lambda_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_t^{\lambda_t}\mathbb{Z}, \]

and hence it is enough to only compute all the \( k \mod p_i^{\lambda_i} \ for i \in \{1, \ldots, t\} \). After that, we can use the Chinese Remainder Theorem to obtain the discrete logarithm we are looking for.
for. In the following we will describe how to compute \( k \mod p_i^\lambda \). Since we will go through the same procedure for all \( i \), we write in short: \( p = p_i \) and \( \lambda = \lambda_i \) and describe the procedure only once. We are looking for a \( z \in \{0, \ldots, p^\lambda - 1\} \) with \( z \equiv k \mod p^\lambda \). We consider the \( p \)-adic notation of \( z \) and compute all the \( z_0, \ldots, z_{\lambda - 1} \) in \( z = \sum_{i=0}^{\lambda-1} z_i p^i \). We will compute each of these \( z_i \) as a new discrete logarithm problem as a subgroup of \( \langle P \rangle \) of order \( p \). Let \( R = \frac{z}{p} P \). The point \( R \) is of order \( p \) \( (pR = \infty) \). We get the following equation using above properties:

\[
\frac{n}{p} Q = \frac{n}{p} k P = k R = z R = z_0 R.
\]

The last equality holds because \( R \) has order \( p \) and all terms except for \( z_0 \) vanish in its \( p \)-adic representation. This means that we can solve the discrete logarithm problem in the subgroup \( \langle R \rangle \) of order \( p \) to get \( z_0 \).

Knowing \( z_0 \) one can compute all the other \( z_i \) recursively. We will show this by the induction step: Having computed all values \( z_0, \ldots, z_{j-1} \) one can get \( z_j \) by the following method:

First of all we can compute

\[
Q_j = \frac{n}{p^{j+1}} (Q - (z_0 + z_1 p + \cdots + z_{j-1} p^{j+1}) P).
\]

Since \( P \in E[n] \) we have \( nP = \infty \) and thus \( \frac{n}{p^{j+1}} p^\lambda P = \infty \). This gives us

\[
\frac{n}{p^{j+1}} Q = \frac{n}{p^{j+1}} k P = \frac{n}{p^{j+1}} z P.
\]

All in all we get:

\[
Q_j = \frac{n}{p^{j+1}} (Q - (z_0 + z_1 p + \cdots + z_{j-1} p^{j+1}) P)
= \frac{n}{p^{j+1}} Q - (z_0 + z_1 p + \cdots + z_{j-1} p^{j+1}) \frac{n}{p^{j+1}} P
= \frac{n}{p^{j+1}} z P - (z_0 + z_1 p + \cdots + z_{j-1} p^{j+1}) \frac{n}{p^{j+1}} P
= \frac{n}{p^{j+1}} P (z - (z_0 + z_1 p + \cdots + z_{j-1} p^{j+1}))
= \frac{n}{p^{j+1}} P (z_j p^j + \cdots + z_{\lambda-1} p^{\lambda-1}) = \frac{n}{p^{j+1}} z_j P = z_j R.
\]

Since we can compute \( Q_j \) we get \( z_j \). We will do this for all \( p_i^\lambda \) to reach the solution of the
Discrete Logarithm Problem. This method is only efficient if all the prime divisors of \( n \) are sufficiently small.

### 4.5.3 Pollard-\( \rho \)

For the Pollard-\( \rho \) method we decompose our elliptic curve into \( s \) disjoint sets

\[
E(\mathbb{F}_q) = \bigsqcup_{i=1}^{s} G_i.
\]

Furthermore we define the function \( f : E(\mathbb{F}_q) \rightarrow \{1, \ldots, s\} \), that maps each point on \( E \) to the index of the \( G_i \) it belongs to. We then define for randomly chosen numbers \( a_1, \ldots, a_s \in \mathbb{Z} \) and \( b_1, \ldots, b_s \in \mathbb{Z} \)

\[
J_i := a_i P + b_i Q, \quad i = 1 \ldots s.
\]

We now define a series of points as follows:

\[
R_0 = x_0 P + y_0 Q \\
R_{i+1} = R_j + J_{f(R_i)}.
\]

This means that for every \( R_j \) we look up in which of the sets \( G_i \) it is lying and then define the next group element by addition of \( J_i \). Every \( R_j \) is of the form

\[
R_i = x_i P + y_i Q
\]

for some \( x_i \) and \( y_i \). Since \( \langle P \rangle \) is finite, because of the pigeon hole principle we will eventually find an element \( R_l \), which already occurred in our sequence. That is, we find two indices \( l \neq m \) with \( R_l = R_m \). Then

\[
(x_l - x_m)P = (y_m - y_l)Q = (y_m - y_l)kP
\]

since \( x_l P + y_l Q = x_m P + y_m Q \). This is equivalent to \( x_l - x_m \equiv (y_m - y_l)k \mod n \). If now \( \gcd(y_m - y_l, n) = 1 \) we can identify \( k \) as

\[
k = \frac{(x_l - x_m)}{(y_m - y_l)} \mod n \in \mathbb{Z}/n\mathbb{Z}.
\]
On the other hand, if $d = \gcd(y_m - y_l) = d \neq 1$ is small enough, one can find the right $k$ with following trial and error method:

Since $\frac{y_m - y_l}{d}$ is invertible in $\mathbb{Z}/n\mathbb{Z}$ there exists a number $y' \in \mathbb{Z}$ with

$$y'(y_m - y_l) \equiv d \mod n.$$ 

Furthermore we know, because $x_l - x_m \equiv (y_m - y_l)k \mod n$, that $x_l - x_m$ is a multiple of $d$. That is $\exists x' \in \mathbb{Z}$ such that $x_l - x_m = dx'$. We multiply $y'$ with this congruence and get

$$dy'x' \equiv dk \mod n.$$ 

Therefore $k - y'x' \mod n$ has to be congruent to one of the values

$$0, \frac{n}{d}, \ldots, (d - 1)\frac{n}{d}.$$ 

That is

$$k \equiv y'x' + i\frac{n}{d} \mod n$$

for an $i \in \{0, 1, \ldots, d - 1\}$. By computing all $kP$ one can check if $Q = kP$ and thus if $k$ is the discrete logarithm we are looking for. If $d$ is too big though, one has to start with another randomly chosen starting point $R_0$. In practice the Pollard-$\rho$ method is often used in combination with the Pohlig-Hellmann algorithm and one can assume that that $n$ is prime. The case $d \neq 1$ is very unlikely in that scenario.

Assuming that the sequence $(R_0, R_1, R_2, \ldots)$ is a random sequence, one can show with tools from probability theory that the collision of two $R_i$ for big $n$ is occurs after appropriately $\sqrt{n}/2$-many sequence elements [12].

The name of the algorithm comes from the following picture:

### 4.5.4 MOV-Attack

This attack was developed by Menenez, Okamoto and Vanstone. The idea of the attack is to reduce the problem of solving the ECDLP to a DLP in $\mathbb{F}_{q^l}^*$ for a certain $l \geq 1$. We will call $l$ the embedding factor. We will try to find the smallest possible $l$ so that the DLP in $\mathbb{F}_{q^l}^*$ is easy to solve. It turns out that specific elliptic curves have a small embedding factor which makes them vulnerable against the MOV-Attack. The idea is to use the Weil-Pairing $\epsilon_n : E[n] \times E[n] \rightarrow \mu_n(\mathbb{F}_q)$. Because of Theorem 4.6 $E[n]$ is isomorphic to $\mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ and so there exists a point
Figure 12: One starts with a random point $R_0$ and continues iterating to the next point, an $R_i$ is repeated. The picture looks like a $\rho$, hence the name of the algorithm.

$P \in E[n]$. Because of bilinearity of the Weil Pairing $e_n$ the set

$$\{ e_n(P, Q) : Q \in E[n] \}$$

is a subgroup of $\mu_n(\mathbb{F}_q)$. Because of Lagrange’s Theorem the order of this subgroup $d$ has to divide the order of $\mu_n(\mathbb{F}_q)$, which is $n$. Thus, for all $Q \in E[n]$ the following holds:

$$1 = e_n(P, Q)^d = e_n(dP, Q).$$

The second equation again holds because of the bilinearity of the Weil Pairing. $dP = \infty$ immediately follows from non-degeneracy. Since $P$ is of order $n$, we have $n = d$.

Given a point $P \in E(\mathbb{F}_q)$ of order $n$ and some $Q \in \langle P \rangle$ we want to find $k \mod n$ s.t.

$Q = kP$.

Without loss of generality we will assume that $\gcd(n, p) = 1$, where $p$ is the characteristic of $\mathbb{F}_q$. If the gcd is not 1, we can write $n = n'p^a$ for some $n'$ with $\gcd(n', p) = 1$ and $a \geq 1$.

We will obtain two Discrete Logarithm Problems by setting

$$P_1 = n'P \text{ and } P_2 = p^aP.$$ 

Now $P_1$ is of order $p^a$ and $P_2$ is of order $n'$. Consider:

$$n'Q = kn'P = kP_1 \text{ and } p^aQ = kp^aP = kP_2.$$
Knowing $k \ mod \ p^a$ and $k \ mod \ n'$ gives us $k \ mod \ n$, because of the chinese remainder theorem. For small $k$, we can solve the first Discrete Logarithm Problem by using the Pohlig-Hellman method combined with the Pollard-$\rho$ method: We decompose the problem into Discrete Logarithm Problems of order of their primepowers and solve these with the Pohlig-Hellman method. We solve the second problem with the MOV-Attack. So from now on we can assume without loss of generality that $\gcd(n, p) = 1$.

We know that $E[n]$ is a finite subgroup of $E(\overline{\mathbb{F}}_q)$. Furthermore any point in $E(\overline{\mathbb{F}}_q)$ is already in one of the subsets $E(\overline{\mathbb{F}}_{q^l})$ for some $l \geq 1$. Since $E[n]$ is finite, we just have to find a sufficiently large $l$ such that $E[n] \subseteq E(\overline{\mathbb{F}}_{q^l})$.

For the MOV-Attack we will do the following four steps:

1. Find an $l \geq 1$ such that $E[n] \subseteq E(\overline{\mathbb{F}}_q)$.
2. Find a point $R \in E[n]$ such that $a := e_n(P, R)$ is a primitive $n$-th root of unity in $\overline{\mathbb{F}}_q$.
3. Compute $b = e_n(Q, R)$.
4. Solve the Discrete Logarithm Problem $b = a^k$ in $\mathbb{F}_{q^l}^*$.

Why does this work? We can conclude from the properties of the Weil Pairing that the mapping $e_n(P, \cdot) : E[n] \longrightarrow \mu_n(\overline{\mathbb{F}}_q)$ is surjective. Because of that there exists a point $R \in E[n]$ for which $e_n(P, R)$ is a primitive $n$-th root of unity. Because of the fifth property of the Weil Pairing we know that $e_n(P, R)$ and $e_n(Q, R)$ will lie in $\mathbb{F}_{q^l}^*$. Because of the bilinearity of $e_n$ we have

$$b = e_n(Q, R) = e_n(kP, R) = e_n(P, R)^k = a^k.$$  

This yields a new Discrete Logarithm Problem $b = a^k$ in $\mathbb{F}_{q^l}^*$. Solving this DLP will give us the $k$ we were looking for.

But how do we know which $l$ we have to choose? There is an easy condition which guarantees that $l$ is suitable for the algorithm. Because of the last property of the Weil Pairing we know that for every $l$ the $n$-Torsion points are contained in $E(\overline{\mathbb{F}}_{q^l})$. Furthermore we know that $e_n$ is surjective, which makes $\mu_n(\overline{\mathbb{F}}_q)$ a subgroup of $\mathbb{F}_{q^l}^*$. 
Table 2: Supersingular Curves [12]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$l$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>$q + 1$</td>
</tr>
<tr>
<td>$\pm \sqrt{2}$</td>
<td>3</td>
<td>$\sqrt{q^2} \pm 1$</td>
</tr>
<tr>
<td>$\pm \sqrt{q}$</td>
<td>4</td>
<td>$q^2 + 1$</td>
</tr>
<tr>
<td>$\pm 2\sqrt{q}$</td>
<td>6</td>
<td>$q^3 + 1$</td>
</tr>
<tr>
<td>$\pm 2\sqrt{q}$</td>
<td>1</td>
<td>$\sqrt{q} \mp 1$</td>
</tr>
</tbody>
</table>

We also know that $\mu_n(\mathbb{F}_q)$ has $n$ elements and $\mathbb{F}_q^*$ has $(q^l - 1)$ elements. Again we apply Lagranges theorem and know that the order of the subgroup has to divide the order of the group. We hence get

$$n \mid q^l - 1.$$  

4.5.5 Practical Consequences

The above described MOV-Attack is only effective when the elliptic curve has a low embedding degree. Supersingular curves turn out to have an embedding degree of loss or equal 6, which makes them vulnerable for the MOV-Attack.

**Definition 4.26.** An elliptic curve $E(\mathbb{F}_q)$ is called supersingular, if

$$p = \text{char}(\mathbb{F}_q) \mid t,$$

where $t = q + 1 - \#E(\mathbb{F}_q)$.

If $q$ is just a prime (not a prime power), then $t = 0$. The computation of the number of points on the curve $E(\mathbb{F}_q)$ is very easy in that case. In other cases one has to use the Schoof algorithm or the Schoof-Elkies-Atkin algorithm. These are well known algorithms to count points on elliptic curves.

In general, one can say more about the group structure of supersingular elliptic curves. It turns out that for supersingular elliptic curves $t = q + 1 - \#E(\mathbb{F}_q)$ can only take exactly 9 different values:

$$0, \pm \sqrt{q}, \pm \sqrt{2q}, \pm \sqrt{3q}, \pm 2\sqrt{q}.$$  

Each of these values directly corresponds with a certain embedding degree. After using the Schoof algorithm, one can compute $t$ and look up the embedding degree in the Table 2:
As mentioned earlier $t = q + 1 - \#E(\mathbb{F}_q)$, $l$ is the embedding degree and $d$ denotes the smallest number such that $dR = \infty \forall R \in E(\mathbb{F}_{q^l})$. That means that one does not have to compute these values explicitly but only use the Schoof algorithm, compute $t$ and look up the rest of the values in this table.

An adapted version of the MOV-Attack for supersingular curves looks like the following:

1. Compute $t = q + 1 - \#E(\mathbb{F}_q)$ with an algorithm (Schoof) and look up $l$ and $d$ in the table above.

2. Randomly choose a point $R' \in E(\mathbb{F}_{q^l})$ and set $R = \frac{d}{n} R'$.

3. Compute the Weil Pairings $a = e_n(P, R)$ and $b = e_n(Q, R)$.

4. Solve $b = a^k$ in $\mathbb{F}_{q^l}^*$.

5. If $kP = Q$ then $k$ is the desired discrete logarithm. Otherwise, go to 2.

Since $p \in E(\mathbb{F}_q) \subseteq E(\mathbb{F}_{q^l})$ is a point of order $n$, $n$ has to divide the exponent $d$ of $E(\mathbb{F}_{q^l})$. Furthermore one can use the Weil Pairing because $R$ is a $n$-Torsion point $nR = dR' = \infty$.

Menezes, Okamoto and Vanstone showed in their paper [17] that the average number of recursions of the attack for supersingular curves is

$$\frac{n}{\varphi(n)} \leq 6 \log \log n$$

for $n \geq 5$.

My computations confirm that the prediction of the amount of recursion steps is rarely exceeded:

**Example 4.27.** Let $E$ be the supersingular elliptic curve defined by $y^2 = x^3 + 1$ over $\mathbb{F}_{773}$. The embedding degree is $l = 2$.

Let $P$ and $Q$ be the $n$-Torsion points

$$P = (6, 137)$$
$$Q = (277, 608),$$
where \( n = 258 \).
Using the implementation we find the solution to be \( k = 209 \), that is \( Pk = Q \) on \( E \). We predict the number of recursion steps as \( \frac{n}{\varphi(n)} = \frac{258}{\varphi(258)} \approx 3 \).
The numbers of iterations performed by the program are

\[
10, 1, 4, 1, 1, 6, 6, 4, 1, 1, 2, 1, 11, 3,
\]

averaging about 3.5, which comply with the prediction.

**Example 4.28.** Let \( E \) be the supersingular elliptic curve defined by \( y^2 = x^3 + 1 \) over \( \mathbb{F}_{743} \).
The embedding degree is \( l = 1 \).
Let \( P \) and \( Q \) be the \( n \)-Torsion points

\[
P = (78x + 73, 295x + 361)
\]
\[
Q = (54x + 591, 358x + 643),
\]
where \( n = 744 \).
Using the implementation we find the solution to be \( k = 361 \), that is \( Pk = Q \) on \( E \). We predict the number of recursion steps as \( \frac{n}{\varphi(n)} = \frac{744}{\varphi(744)} \approx 3 \).
The numbers of iterations performed by the program are

\[
13, 2, 7, 1, 1, 1, 9, 3, 1, 1, 1, 1, 1, 1,
\]

averaging about 2.9, which comply with the prediction.

**Example 4.29.** Let \( E \) be the supersingular elliptic curve defined by \( y^2 = x^3 + x \) over \( \mathbb{F}_{107^8} \). The embedding degree is \( l = 1 \).
Let \( P \) and \( Q \) be the \( n \)-Torsion points

\[
P = (81x^7 + 54x^6 + 50x^5 + 79x^4 + 10x^3 + 103x^2 + 61x + 66,
\]
\[
74x^7 + 41x^6 + 67x^5 + 41x^4 + 30x^3 + 33x^2 + 38x + 52)
\]
\[
Q = (46x^7 + 14x^6 + 91x^5 + 90x^4 + 18x^3 + 48x^2 + 59x + 97,
\]
\[
103x^7 + 7x^6 + 88x^5 + 58x^4 + 77x^3 + 84x^2 + 79x + 89),
\]
where \( n = 131079600 \).
Using the implementation we find the solution to be $k = 3351248$, that is $Pk = Q$ on $E$. We predict the number of recursion steps as $\frac{n}{\varphi(n)} = \frac{131079600}{\varphi(131079600)} \approx 3$. The number of iterations given by the program are

$$3, 2, 1, 2, 2, 5, 2, 1, 1, 3, 2, 1, 4, 2, 5,$$

averaging about 2.4, which comply with the prediction. The reader is invited to use the code in Appendix A to solve ECDLP on any desired supersingular elliptic curve.
A

Source Code

This is my implementation of the MOV-Attack on supersingular elliptic curves on SAGE.

def function(n):
    p = random_prime(n, lbound=5)
    if 3.divides(p-2):
        return p
    return function(n)

p = function(10^10)
e = 2
a = 0
b = 1
q = p^e

E = EllipticCurve(GF(q, 'x'), [a,b])
n = E.cardinality()

t = q+1-n

E.is_supersingular()

def table(t):
    if t==0:
        return 2,q+1
    elif t==sqrt(q):
        return 3,sqrt(q^3)+1
    elif t==-sqrt(q):
        return 3,sqrt(q^3)-1
    elif t==sqrt(2*q):
        return 4,q^2+1
elif t == -sqrt(2*q):
    return 4, q^2+1
elif t == sqrt(3*q):
    return 6, q^3+1
elif t == -sqrt(3*q):
    return 6, q^3+1
elif t == 2*sqrt(q):
    return 1, sqrt(q)-1
elif t == -2*sqrt(q):
    return 1, sqrt(q)+1

print "Elliptic Curve is not supersingular"

l = table(t)[0]
d = table(t)[1]

P = E.random_element()
m = P.order()
r = ZZ.random_element(m)
Q = P*r

K.<x> = GF(q^l)
E_ = EllipticCurve(K, [a,b])
P_ = E_(P)
Q_ = E_(Q)

def MOV(P,Q,i):
    i = i+1
    R = E_.random_element()*(d//m)
    W1 = P_.weil_pairing(R, m)
    W2 = Q_.weil_pairing(R, m)
    try:
        u=discrete_log(W2,W1,m)
        if P*u==Q:
            return u,i
except:
    MOV(P, Q, i)
    return MOV(P, Q, i)

a = MOV(P, Q, 0)
References


[16] FREEPICK: [www.flaticon.com](www.flaticon.com), Icons made by Freepik from www.flaticon.com

[17] MENEZES, OKAMOTO, VANSTONE:

*Reducing elliptic curve logarithms to logarithms in a finite field*


[18] SILVERMAN:

*The arithmetic of elliptic curves. Graduate Texts in Mathematics 106.*

Springer 1986
STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

................................. .................................
             date                   (signature)