Introduction to mathematical cryptography

Lecture notes for the Preliminary Arizona Winter School 2025

Sabrina Kunzweiler

This is a draft, and will be updated throughout the semester.

Abstract

These lecture notes accompany the biweekly lectures given at the Preliminary Arizona Winter School in 2025. The field of mathematical cryptography is vast, and the focus of this course is on the *discrete logarithm problem* (DLP) which will appear in different forms throughout the lectures.

After a general introduction to cryptography, Section 2 presents the Diffie-Hellman key exchange protocol in its original form which is based on the DLP in finite fields. In Section 3, we proceed to study elliptic curves. Here, the *elliptic curve discrete logarithm problem* ECDLP will be discussed in detail. Finally, the last section will provide some insight into isogeny-based cryptography. This is a very recent research topic, and curiously the security of isogeny-based protocols also relies on a DLP-type assumption.

This course will assume only a first undergraduate course in abstract algebra as a prerequisite. Prior knowledge of cryptography or elliptic curves is not required.

There exist many excellent textbooks and lecture notes on the topic. The main source for these notes is [12]. For Section 1, we also rely on [14], and Sections 2 and 3 draw inspiration from [32] and [35]. Furthermore, we often refer to [30] or [9] for more advanced topics that are not covered in this course.

Contents

1 A brief introduction to cryptography

Cryptography is a century old discipline that has evolved from being an art only used by few people, to a scientific subject of active research which we use in our everyday lives.

The goal of cryptography is to keep communication private. In this course, there will usually be two parties, Alice and Bob, who want to exchange secret information over a public channel. In order to avoid that an eavesdropper (Eve) can read the secret information, they encrypt the messages that they send to each other. A typical encryption scheme is sketched in Figure 1.

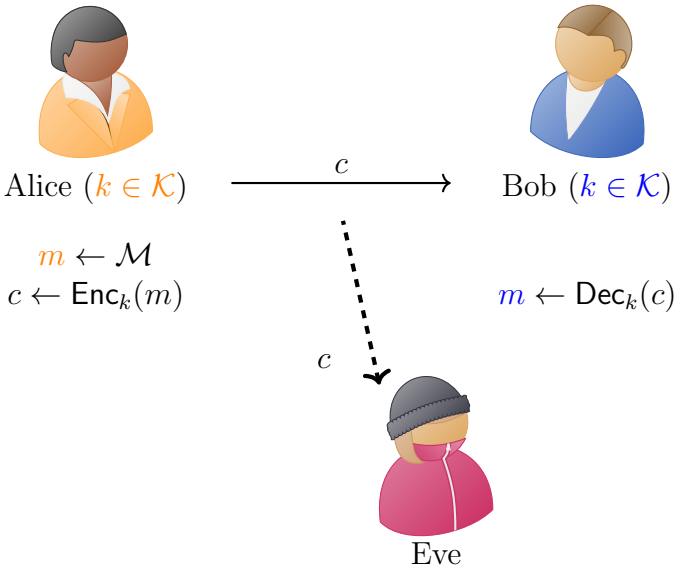


Figure 1: Encryption scheme with a shared private key k

In this encryption scheme, there are three predefined spaces, the *key space* $\mathcal{K}$, the *message space* $\mathcal{M}$ and the *cipher text space* $\mathcal{C}$ (often $\mathcal{M} = \mathcal{C}$). Both Alice and Bob share the same private key $k \in \mathcal{K}$ which has to be chosen in advance. Furthermore, the scheme consists of (publicly known) encryption and decryption functions

$$\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}, \quad \mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

We often denote $\mathsf{Enc}_k(m) = \mathsf{Enc}(k, m)$ and $\mathsf{Dec}_k(m) = \mathsf{Dec}(k, m)$. In Figure 1, Alice wants to transmit a message $m \in \mathcal{M}$. To do so she computes the cipher text $c = \mathsf{Enc}_k(m)$, and sends the latter to Bob. Upon receiving this message, Bob computes $m = \mathsf{Dec}_k(c)$. For this to work, it is required that

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m \quad \text{for all } m \in \mathcal{M}, k \in \mathcal{K}.$$

Note that Eve can read the cipher text c. Intuitively, the scheme is secure if Eve cannot learn anything about the plain text m from intercepting the cipher text c.

How can one construct secure *encryption and decryption algorithms?* An early example of an encryption scheme is the so-called *Caesar cipher*, named after Julius Caesar who used this method for secret communication. This encryption method is far from being secure by today's standards, but it serves us as a first example.

Example 1.1. *Caesar's cipher. According to historic sources Julius Caesar used to encrypt his messages by shifting every letter in a word by* 3 *positions. In other words, he used the following translation table.*

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

For instance PAWS *becomes* SDZV*, using this encryption method. In order to use the formalism introduced in Figure 1, we identify the alphabet with integers modulo* 26*, i.e.* $\mathcal{A} = \{A, \dots, Z\} = \{0, \dots, 25\}$*. Then the message and cipher text spaces are given by*

$$\mathcal{M} = \mathcal{C} = \mathcal{A}^* = \{a_1 \dots a_n \mid a_i \in \mathcal{A}, n \in \mathbb{N}\},$$

and the encryption function can be described as

$$\mathsf{Enc}(m_1 \dots m_n) = (c_1 \dots c_n), \quad \textit{with } c_i = m_i + 3 \pmod{26}.$$

Similarly, decryption is given by

$$\mathsf{Dec}(c_1 \dots c_n) = (m_1 \dots m_n) \quad \textit{where } m_i = c_i - 3 \pmod{26}.$$

Note that this method does not depend on a key k. Hence the security of the scheme only relied on the assumption that it is unknown to an enemy. This assumption is in conflict with *Kerckhoff's principle* which states that the security of an encryption scheme should never depend on the secrecy of the scheme, but only on the secrecy of the key [15]:

> *Il faut qu'il n'exige pas le secret, et qu'il [le système] puisse sans inconvénient tomber entre les mains de l'ennemi. (Kerckhoff 1883)*

In the words of Claude Shannon [28]:

> *The enemy knows the system. (Shannon 1949)*

While the principle is relatively old, it is still important for modern cryptography.

The next two exercises discuss generalizations of Caesar's cipher. In these generalizations, the security is based on the knowledge of the secret key. However, they will turn out to be insecure as well by today's standards.

Exercise 1.2. *Instead of using, a shift of exactly* 3 *letters as in Caesar's cipher, one could also use a secret shift depending on a key* $k \in \{0, \dots, 25\}$.

(a) Describe $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$, Dec *and* Enc *for the this new encryption method.*

(b) To increase the number of keys, one may also choose a key of the form $k = (a, b)$ *with* $a \in (\mathbb{Z}/26\mathbb{Z})^*$, $b \in \mathbb{Z}/26\mathbb{Z}$, *and*

$$\mathsf{Enc}_k(m_1 \dots m_n) = (c_1 \dots c_n) \quad \textit{with } c_i = am_i + b.$$

The resulting scheme is known as affine cipher.

(i) Describe the corresponding decryption function Dec_k. *Why is it necessary that* a *is a unit in* $(\mathbb{Z}/26\mathbb{Z})$*?*

(ii) You (Eve) read a cipher text starting with BMVVK, *and you think it means* HELLO. *Is it possible that Alice and Bob used an affine cipher in their communication? Can you recover their secret key?*

(iii) SAGE *Alice and Bob noticed that you found their secret, and chose a new private key. This time you intercept the cipher text*

IFELTKHURFENHAFEEFSFUTSVGEDNULTKFBF

Can you find the plain text message?

The ciphers from Example 1.1 and Exercise 1.2 fall into the larger category of *substitution ciphers*, where each letter in the alphabet is replaced by another letter. Note that as opposed to the ciphers from Exercise 1.2, the key space for general substitution ciphers is much larger. Using the standard Latin alphabet with 26 letters, there are $26! \cong 2^{88}$ different keys. While it would be infeasible (for most computers) to test all of these different keys, there are faster methods to break this encryption scheme. The main idea is to use the fact that the plain text message should make sense in some language (in our examples: the English language). The classical way to attack such a cipher is by performing a *frequency analysis*. One checks which letter appears the most, and how this compares to standard texts in English.[1] Furthermore it makes sense to also check for frequent letter pairs or repeats. The most frequent letters in the English alphabet are

$$E\ (13.1\%), \quad T\ (10.5\%), \quad A\ (8.1\%), \quad O\ (8.0\%), \quad N\ (7.1\%),$$

[1] The first description of frequency analysis to break cryptographic ciphers appeared in the 9th century for the Arabic language in a *manuscript on deciphering cryptographic messages* by Al-Kindi.

the most common bigrams in decreasing order are

$$TH, \quad HE, \quad AN, \quad RE, \quad ER.$$

For more details, we refer to [12, Tables 1.3, 1.6].

Exercise 1.3. SAGE *You intercepted the cipher text*

JIVQOJIV LEALAVQO KGOONDTV QOAELONE OAINYNGJ SOBVQODB CLAVQOKG OONDTJIV QOJIVLEA EIBHTBLO YBLEQPIG AA

from a conversation between Alice and Bob. You know that they used a substitution cipher. Can you recover the plain text m? Note that the spacing is only used for readability and does not coincide with the spacing of the original text.

The historical ciphers that we have seen so far, all belong to the field of *symmetric cryptography*. Both parties Alice and Bob possess the same secret key k which is used for encryption *and* decryption. A modern method in this category is the *Advanced Encryption Standard (AES)*. This cipher was selected as a new standard for encryption by the United States National Institute of Standards and Technology (NIST) in the year 2000, and it is widely used today. To learn more about AES and related modern ciphers, we refer to [14, Chapter 6].

Note that in order to use a symmetric encryption scheme, Alice and Bob need to first exchange the secret key k with each other. In most scenarios, it is not possible that they can meet in person and establish this key without any eavesdroppers present. This problem can be solved by using *public key cryptography*, also known as asymmetric cryptography. In public key cryptography, each party possesses their own secret key (sk) and a corresponding public key (pk). A typical key exchange protocol is sketched in Figure 2. In such a protocol, the secret keys of Alice and Bob, sk_A and sk_B remain private, they only exchange the

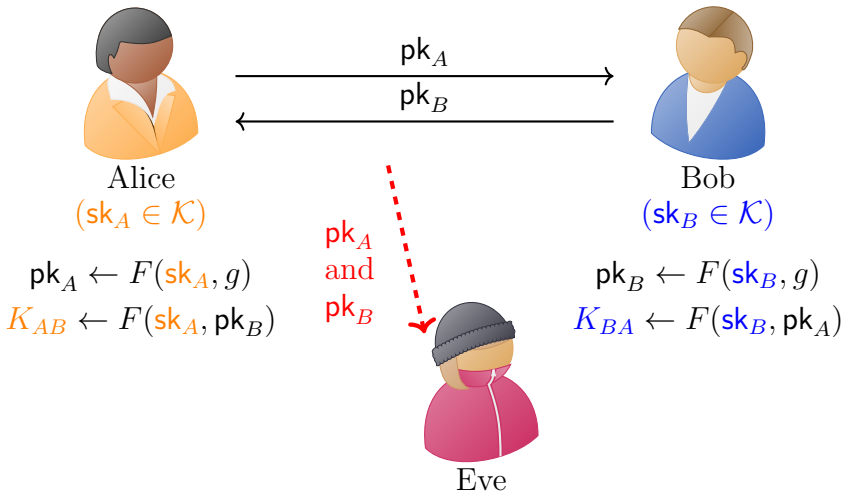


Figure 2: Key exchange protocol

public keys pk_A and pk_B with each other. The shared session key K_{AB} is derived by applying a function F with the property

$$F(\mathsf{sk}_A, \mathsf{pk}_B) = F(\mathsf{sk}_B, \mathsf{pk}_A) \quad \text{for all pairs } (\mathsf{sk}_A, \mathsf{pk}_A), (\mathsf{sk}_B, \mathsf{pk}_B).$$

Intuitively, the following properties must hold in order to get a secure scheme.

(a) Given a public key pk_A, it is hard to find the corresponding secret key sk_A.

(b) Given two public keys pk_A and pk_B, it is hard to compute the corresponding shared key K_{AB}.

In particular, we ask that F is a cryptographic one-way function.

Definition 1.4: Cryptographic one-way function

A function $f : X \to Y$ is a *cryptographic one-way function* if the following conditions hold.
(1) **Easy to compute**: There is a polynomial-time algorithm that on input $x \in X$ computes $y = f(x)$.
(2) **Hard to invert**: Given $y \in Y$ it should be computationally infeasible to find $x \in X$ with $f(x) = y$.

Cryptographic one-way functions typically stem from number theory. A one-way function that plays an important role in modern cryptography is simply given by multiplication.

Example 1.5. *Let $\mathcal{P}$ be the set of primes. Then the function*

$$F : \mathcal{P} \times \mathcal{P} \to \mathbb{Z}, \quad (p, q) \mapsto p \cdot q$$

is a (conjectured) cryptographic one-way function. The multiplication of two (large) prime numbers (or more generally integers) can be performed efficiently. More precisely, it is polynomial *in the input length of the binary representation of p and q. On the other hand, given an integer $N = pq$, it is hard to compute the factorization. For instance a trivial approach using trial division runs in time $\tilde{\mathcal{O}}(\sqrt{N})$. This is* exponential *in the length of the binary representation of N. The best known algorithms to solve factoring run in subexponential time, but no polynomial time method is known to solve the problem.*

Having a cryptographic one-way function, it is still not straight forward, how one would construct a key exchange protocol from it. In particular, no direct construction of a key exchange protocol from integer multiplication is known (Example 1.5). In the example below, we provide an unsuccessful idea for using multiplication in a key exchange protocol.

Example 1.6. *Consider the following setup for the public key exchange scheme in Figure 2. Let $\mathcal{K} = \mathcal{P}$ be the set of primes, $p \in \mathcal{P}$ a public parameter, and let*

$$\begin{aligned} F : \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \\ (a, b) &\mapsto a \cdot b \end{aligned}$$

With this function, the key exchange protocol from Figure 2 works correctly. We have $\mathsf{pk}_A = \mathsf{sk}_A \cdot p$, $\mathsf{pk}_B = \mathsf{sk}_B \cdot p$, and

$$F(\mathsf{sk}_A, \mathsf{pk}_B) = \mathsf{sk}_A \cdot (\mathsf{sk}_B \cdot p) = \mathsf{sk}_B \cdot (\mathsf{sk}_A \cdot p) = F(\mathsf{sk}_B, \mathsf{pk}_A)$$

for all choices of secret keys $\mathsf{sk}_A, \mathsf{sk}_B \in \mathcal{P}$.

However, the scheme is not secure at all. For instance, given a public key pk_A, one can easily compute the corresponding secret key sk_A as $\mathsf{sk}_A = \mathsf{pk}_A/p$. Note that the hardness of the scheme is not *based on the factorization problem which is assumed to be computationally hard.*

The description of cryptographic one-way functions lies at the heart of this course. We will encounter the following conjectured one-way functions:

- Modular exponentiation (Section 2)
- Elliptic curve multiplication (Section 3)
- Isogenies (Section 4)

For all of these, we will construct a public key exchange protocol as in Figure 2, and analyze the security of the protocol. The conjectured one-way function relying on the hardness of factorization from Example 1.5 will not be studied explicitly in the following sections. However, we would like to highlight that it builds the basis of the RSA cryptosystem which was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977 [24], and is another important pillar of modern public key cryptography. It will appear on the problem sets accompanying these lectures notes.

2 The discrete logarithm problem and Diffie-Hellman key exchange

In this section, we study the first cryptographic one-way function of the course in detail: *Modular exponentiation*

$$\mathsf{exp}_g : \mathbb{Z} \to \mathbb{F}_p^*, \quad a \mapsto g^a,$$

for some prime field $\mathbb{F}_p$, and a base element $g \in \mathbb{F}_p^*$. Recall that modular exponentiation can be evaluated in polynomial time in the input using square and multiply techniques,[2] hence the first property from Definition 1.4 is satisfied. In order to be a cryptographic one-way function, we also require that the inversion of exp_g is hard. The inversion consists in computing the *discrete logarithm*:

Definition 2.1: Discrete Logarithm Problem (DLP)

Let g be a primitive root for $\mathbb{F}_p$, and let $A \in \mathbb{F}_p^*$. The *Discrete Logarithm Problem* (DLP) is the problem of finding an exponent a such that $\mathsf{exp}_g(a) = A$.
The number a is called the *discrete logarithm* of A to the base g, we denote $a = \mathsf{dlog}_g(A)$.

Recall that $\mathbb{F}_p^*$ is cyclic of order $p-1$. A *primitive root* of $\mathbb{F}_p$ is an element $g \in \mathbb{F}_p$ so that $\mathbb{F}_p^* = \langle g \rangle$. Choosing a primitive root g as the base, there exists a solution to the DLP for any $A \in \mathbb{F}_p^*$. Further note that $\mathsf{exp}_g(a) = \mathsf{exp}_g(a + k \cdot (p-1))$ for any $k \in \mathbb{Z}$. One may show that $\mathsf{dlog}_g : \mathbb{F}_p^* \to \mathbb{Z}/(p-1)\mathbb{Z}$ is a well-defined function.

2.1 Diffie-Hellman key exchange

If the discrete logarithm problem (Definition 2.1) is hard in $\mathbb{F}_p$, then modular exponentiation in $\mathbb{F}_p$ is a cryptographic one-way function. Plugging this one-way function into the general framework from Figure 2, we obtain the famous *Diffie-Hellman key exchange* protocol. This protocol was proposed in 1976 by Whitfield Diffie and Martin Hellman in their ground-breaking article "New directions in cryptography" [5] which represents the start of public key cryptography. The Diffie-Hellman key exchange is sketched in Figure 3.

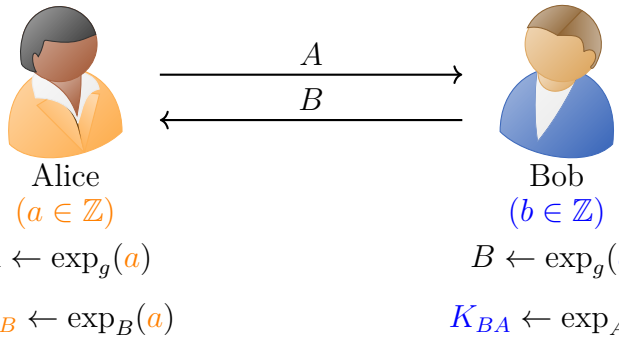


Figure 3: Diffie-Hellman key exchange for some public parameter $g \in \mathbb{F}_p^*$

Here the secret keys are integers $a, b \in \mathbb{Z}$, and the public keys are $A = \mathsf{exp}_g(a)$ and $B = \mathsf{exp}_g(b)$, respectively. Note that we omit including Eve in the sketches from now on. But one should have in mind that the public keys A, B are sent over a public channel and can be intercepted by an eavesdropper.

The correctness of the scheme follows from commutativity, more precisely

$$K_{AB} = B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b = K_{BA}$$

for all $a, b \in \mathbb{Z}$.

Remark 2.2. *If the public element $g \in \mathbb{F}_p^*$ is a primitive root, then its order is $p-1$. For security reasons, it is better to choose an element with (large) prime order (see Exercise 2.6). For instance, one*

[2] See Exercise 7, on Problem Set 0.

may choose to work with a prime of the form $p = 2q + 1$ where q is prime as well. Then, g is replaced by $h = g^2 \in \mathbb{F}_p^*$, and we have $ord(h) = q$.

Further note that in this setting, $h^a = h^{a+n\cdot q}$ for all integers n. Therefore, it makes sense to choose secret keys in the range $\{0, \dots, q-1\}$.

Example 2.3. *In order to establish a shared key, Alice and Bob use the following public parameters*

- $q = 113$ *and* $p = 2q + 1 = 227$,
- $g = 4 \in \mathbb{F}_p^*$.

Alice chooses $a = 73$ as her secret key and sends the public key $A = \mathsf{exp}_g(a) = 102$ to Bob.

Bob chooses $b = 89$ as his secret key and sends the public key $B = \mathsf{exp}_g(b) = 19$ to Alice.

Now Alice computes $K_{AB} = \mathsf{exp}_B(a) = 19^{73} = 113$. Similarly Bob computes $K_{BA} = \mathsf{exp}_A(b) = 102^{89} = 113$.

After this key exchange, Alice and Bob can now use a symmetric crypto system with their shared key $K_{AB} = K_{BA} = 113$.

In order to understand the security of the scheme, it is essential to study the hardness of the DLP. If the DLP is easy to solve, then an eavesdropper, can compute the shared session key established by Alice and Bob. The following exercise should give an idea on the effect of changing the size of the primes in the setup on the hardness of computing discrete logarithms. In Subsections 2.3 and 2.4, we will discuss algorithms to solve the DLP in more detail. For now, the reader may simply use the functions integrated in SageMath to solve the exercises.

Exercise 2.4. SAGE *Alice and Bob want to create a shared key. They use setups with varying security levels. In all of these, the public parameters are a prime $p = 2q + 1$, and the element $g = 4 \in \mathbb{F}_p^*$ with order q. You observe the following conversations. Can you find the shared keys?*

(a) $q = 4294967681 \approx 2^{32}$,
$A = 5104411285$, $B = 7620748646$

(b) $q = 18446744073709552109 \approx 2^{64}$,
$A = 17485644247020728566$, $B = 17485644247020728566$

(c) $q = 340282366920938463463374607431768219863 \approx 2^{128}$,
$A = 158556695861572453782110953476057063 05$, $B = 643791185530305885858740134964520672205$

In SageMath, you can use the `log` *function to compute discrete logarithms, e.g.* `a = A.log(g)` *(provided that g, A are defined as elements in $\mathbb{F}_p$). Further, you can use* `%time` *to time your results. How does the runtime evolve for increasing values of q?*

Let us now turn to the question, why we prefer to work in a large prime order subgroup of $\mathbb{F}_p^*$ (Remark 2.2). Essentially, the reason is that the DLP in $\mathbb{F}_p^*$ can be reduced to the DLP in its prime order subgroups. We illustrate this idea in Example 2.5. The generalization of these ideas to a general algorithm (the *Pohlig-Hellman algorithm*) is left as an exercise to the reader (Exercise 2.6).

Example 2.5. *Let $p = 443$ and $g = 2 \in \mathbb{F}_p^*$ be the setup for a Diffie-Hellman key exchange protocol. Assume that we have intercepted Alice's public key $A = 74$, and we want to compute her secret key $a = \mathsf{dlog}_2(A)$.*

We note that $g = 2$ is a primitive root of $\mathbb{F}_p^*$, i.e. $ord(g) = 442 = 2 \cdot 13 \cdot 17$. Our goal is to decompose the computation of $\mathsf{dlog}_2(74)$ into computations in the prime order subgroups of $\mathbb{F}_p^*$. To do so, we will repeatedly use the fact that

$$g^a = A \quad \Leftrightarrow \quad g^{ak} = A^k \quad \text{for all } k \in \mathbb{Z}. \tag{1}$$

- *The unique subgroup of order 2 in $\mathbb{F}_p^*$ is given by $G_2 = \langle g^{13\cdot 17} \rangle = \{1, 442\}$. It follows from Equation 1 that*

$$442^a = g^{a\cdot 13\cdot 17} = A^{13\cdot 17} = 442,$$

hence $a \equiv 1 \pmod 2$.

- *The unique subgroup of order* 13 *in* $\mathbb{F}_p^*$ *is given by* $G_{13} = \langle g^{2\cdot 17} = 35 \rangle$*, explicitly:*

$$G_{13} = \{1,\ 35,\ 35^2 = 339,\ 35^3 = 347,\ 35^4 = 184,\ 35^5 = 238,\ 35^6 = 356,\ 35^7 = 56,$$
$$35^8 = 188,\ 35^9 = 378,\ 35^{10} = 383,\ 35^{11} = 115,\ 35^{12} = 38\}.$$

  Now $A^{2\cdot 17} = 356$*, and it follows from Equation 1 that* $a \equiv 6 \pmod{13}$*.*

- *In a similar way, we find that* $a \equiv 4 \pmod{17}$*.*

Now, one applies the explicit Chinese remainder theorem to find a solution for $a \pmod{2 \cdot 13 \cdot 17}$*.*

$$\begin{cases} a \equiv 1 \pmod{2} \\ a \equiv 6 \pmod{13} \\ a \equiv 4 \pmod{17} \end{cases} \quad \Leftrightarrow \quad a \equiv 123 \pmod{442}.$$

Exercise 2.6. *(difficult) Let* p *be a prime and* $g \in \mathbb{F}_p^*$ *a primitive element. We denote* $p-1 = p_1^{e_1} \cdots p_n^{e_n}$ *for the prime factorization of* $p-1$*. The goal of this exercise is to show that solving the* DLP *in* $\mathbb{F}_p^*$ *is essentially as hard as solving the* DLP *in a subgroup* $G \subset \mathbb{F}_p^*$ *of prime order* $\#G = \max\{p_i \mid i \in \{1, \ldots, n\}\}$*. To make this more formal, let's say that the* DLP *in a subgroup* $G_i \subset \mathbb{F}_p$ *of order* p_i *can be solved in time* $O(S_i)$*.*

Let $g \in \mathbb{F}_p^*$ *be a primitive element and* $A \in \mathbb{F}_p^*$ *the challenge for which we want to solve the* DLP*, i.e. we want to find* $a \in \mathbb{Z}$ *with* $g^a = A$*.*

(a) Use the Chinese remainder theorem to translate the problem into solving n *smaller instances of the* DLP *in subgroups of order* $p_i^{e_i}$ *with* $i \in \{1, \ldots, n\}$*, respectively.*

(b) Fix $i \in \{1, \ldots, n\}$ *and say you want to solve one of the small* DLP *instances on input* $A_i, g_i \in \mathbb{F}_p^*$*, where* $ord(g_i) = p_i^{e_i}$*, i.e. you want to find* $a_i \in \mathbb{Z}$ *with*

$$g_i^{a_i} = A_i \pmod{p_i^{e_i}}.$$

Show that this can be done in time $O(e_i S_i)$*.*
hint: Use a p_i-ary representation $a_i = \alpha_0 + \alpha_1 p_i + \cdots + \alpha_{e_i - 1} p_i^{e_i - 1}$

(c) Combine the results of (a) *and* (b) *to show that the* DLP *can be solved in time*

$$O(\mathsf{polylog}(p) \max\{S_i\}).$$

So far, we only discussed the hardness of the DLP. We tried to attack the key exchange protocol by finding the secret key. *Are there any other strategies for an eavesdropper to compute the shared session key instead of solving the* DLP*?* To formalize this question, we first define a problem which mimics the information that the potential eavesdropper can use to compute the shared session key.

Definition 2.7: Computational Diffie-Hellman Problem (CDH)

Let $g \in \mathbb{F}_p^*$, and let $A = g^a$, $B = g^b$ for some $a, b \in \mathbb{Z}$. The *Computational Diffie-Hellman Problem* (CDH) is the problem of computing $C = g^{ab}$.

Until today, the best known methods to solve CDH rely on solving DLP. We will discuss these methods in Subsections 2.3 and 2.4. However, we would like to point out that it is an open problem, whether the two problems CDH and DLP are equivalent, or if the CDH problem is easier than DLP. Here are some pointers to interesting results in this directions:

- *Maurer reduction* [18]: In this reduction, it is shown that solving the DLP can be reduced to solving CDH in a different auxiliary group, more precisely in a particular elliptic curve group (see Section 3 for a definition of these groups). However the construction of the specific auxiliary group is in general not efficient (see [19] for some recent progress in this direction).

- *Algebraic group model* [8]: Roughly explained, in the algebraic group model, an adversary can only work with elements of $\mathbb{F}_p^*$ that are obtained through group operations from the public parameters and public keys. For instance, in order to solve a DLP instance (g, A), the adversary can only work with elements of the form g^x and A^x for some $x \in \mathbb{Z}$ (or combinations of these). In this model, it is shown that CDH and DLP are indeed equivalent.

2.2 ElGamal encryption

Modular exponentiation cannot only be used to construct a public key exchange protocol. In addition, one can directly construct a *public key encryption scheme*. Such a scheme was first suggested by Taher Elgamal in 1985 [6].

In contrast to a *symmetric* key encryption scheme as sketched in Figure 1, Alice and Bob do not need to share a key in advance. A public key is used for encryption, while decryption requires knowledge of the corresponding secret key.

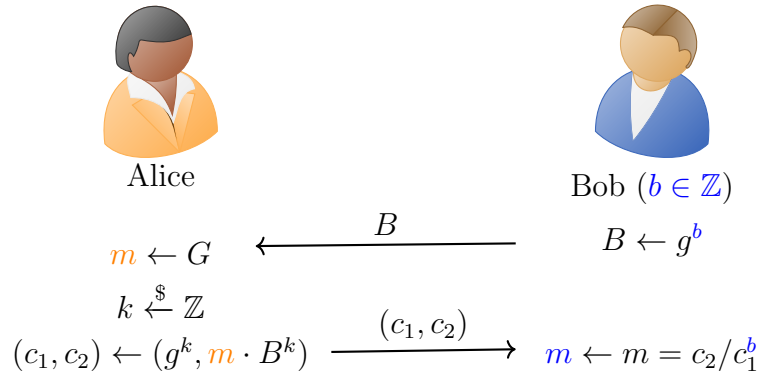


Figure 4: The ElGamal encryption scheme in a finite field $\mathbb{F}_p$ with public parameter $g \in \mathbb{F}_p^*$ and $G = \langle g \rangle \subset \mathbb{F}_p^*$.

The ElGamal encryption scheme is sketched in Figure 4. As in the Diffie-Hellman key exchange scheme, the public parameter is a (prime order) element $g \in \mathbb{F}_p^*$. We denote $G = \langle g \rangle \subset \mathbb{F}_p^*$. The key pairs $(\mathsf{sk}, \mathsf{pk})$ are of the same form as in the Diffie-Hellman key exchange:

$$(\mathsf{sk}, \mathsf{pk}) = (a, A = \mathsf{exp}_g(a)) \quad \text{for some } a \in \mathbb{Z}.$$

A difference is that here only one party (the receiver of the message) needs to create such a key pair. In Figure 4, Bob is the receiver and generates the key pair $(b, B = g^b)$. While Alice is the sender, she chooses a message $m \in G$ which she wants to transmit to Bob. The encryption function is given by

$$\begin{aligned} \mathsf{Enc} : G \times G &\to G^2 \\ (\mathsf{pk}, m) &\mapsto (c_1, c_2) = (g^k, m \cdot B^k) \quad \text{for some random } k \in \mathbb{Z}. \end{aligned}$$

Note that the encryption function is *not deterministic*. The outputted cipher text not only depends on the message m and the public key pk, but it also depends on an integer k which is chosen at random during encryption.

On the other hand, the decryption function is *deterministic*, it is given by

$$\begin{aligned} \mathsf{Dec} : \mathbb{Z} \times G^2 &\to G \\ (\mathsf{sk}, (c_1, c_2)) &\mapsto c_2/c_1^{\mathsf{sk}}. \end{aligned}$$

Proposition 2.8: ElGamal encryption (correctness)

The ElGamal encryption scheme as sketched above is correct, i.e. $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m$ for all messages $m \in G$, and key pairs $(\mathsf{sk}, \mathsf{pk})$.

Proof. Let $\mathsf{sk} \in \mathbb{Z}$ be a secret key, $\mathsf{pk} = \mathsf{exp}_g(\mathsf{sk})$ and $m \in G$. Then

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = \mathsf{Dec}_{\mathsf{sk}}(g^k, m \cdot \mathsf{pk}^k) = \frac{m \cdot \mathsf{pk}^k}{(g^k)^{\mathsf{sk}}} = m.$$

In the last equality, we used that $(g^k)^{\mathsf{sk}} = (g^{\mathsf{sk}})^k = \mathsf{pk}^k$ for every $k \in \mathbb{Z}$. □

Note that the scheme is insecure if DLP or CDH are easy to solve (try this as an exercise). On the other hand, showing that the encryption scheme is secure would require some more advanced cryptographic definitions. The important key words here are *chosen cipher text* and *chosen plain text* attacks. This is not in the scope of this course, and we refer to [14, Section 11.4.1] for more details.

However to better understand certain properties of the protocol, we include two exercises below that illustrate the following:

- It is important that the integer $k \in \mathbb{Z}$ is chosen at random for each encryption (in some sense, it can be viewed as Alice's secret key, but it should never be reused).
- The message has to be in $G \subset \mathbb{F}_p^*$ and cannot be an arbitrary element in $\mathbb{F}_p^*$.

Exercise 2.9. *Consider the following setup:*

$$p = 8589935363, \quad g = 4 \in \mathbb{F}_p^*,$$

and assume that Bob's public key is

$$B = 1865230978.$$

(For readability, we chose a small prime for which the dlog *can still be computed efficiently. For the sake of this exercise, assume however that you cannot compute* $b = \mathsf{dlog}_g(B)$*.)*

Bob is asking some yes/no questions to Alice. Alice encrypts her answers ($Y = 25 \in \mathbb{F}_p^*$ *for yes and* $N = 14 \in \mathbb{F}_p^*$ *for no) using Bob's public key and the ElGamal encryption scheme.*

1. *Eve intercepts Alice's answers:*

   Answer 1: $(2456530342, 8487632028)$,
   Answer 2: $(2456530342, 1660697205)$,
   Answer 3: $(2456530342, 1660697205)$,

   and immediately sees that Alice is reusing the random integer $k \in \mathbb{Z}$*.*

   (a) *Without doing any computations: What are the possible answers that Alice could have sent?*

   (b) *With some (computationally easy) computation: What are Alice's answers to Questions 1,2,3?*

2. *Alice notices her mistake and uses different random exponents for the next answers. However, she decides that it is easier to encode* $Y = 1 \in \mathbb{F}_p^*$ *and* $N = -1 \in \mathbb{F}_p^*$*. Now Eve intercepts the following messages:*

   Answer 4: $(6324669601, 8569725934)$,
   Answer 5: $(5864877653, 1038689194)$,
   Answer 6: $(1841857395, 573429127)$,

   Can you recover Alice's answers to Questions 4,5,6 as well?

While the specific scenarios in this exercise might look a bit artificial, such mistakes can lead to serious security leaks in real-world applications.

2.3 Exponential attacks on DLP

Let $g \in \mathbb{F}_p^*$ be an element of order q, and $A \in \langle g \rangle$. The "easiest" way to solve the corresponding DLP instance, i.e. find an integer $a \in \mathbb{Z}$ with $A = g^a$, is a *linear search*. One simply computes $g_i = g^i$ for all integers $0 \leq i \leq q-1$ until a match $A = g_i$ is found. The linear search algorithm has running time $O(q)$.[3]

Here, we discuss two different algorithms that both have running time $O(\sqrt{q})$, but different memory requirements. Assuming that g generates a large subgroup of $\mathbb{F}_p^*$, in particular $q \approx p$, the running times are exponential in the length of the input.

[3] We use the *big O notation* to describe the running time of algorithms. We write $f(n) = O(h(n))$ ($f(n)$ is big O of $h(n)$) if there exist a constant $c > 0$ and an integer $n_0 \in \mathbb{N}$ so that $|f(n)| \leq c|h(n)|$ for all $n > n_0$.

2.3.1 Baby-step giant-step algorithm

In 1971, David Shanks proposed the so-called *baby-step giant-step algorithm* to compute discrete logarithms [27]. The main underlying idea is to decompose the solution $a = \mathsf{dlog}_g(A)$ as

$$a = jm + i, \quad \text{with } m = \lfloor \sqrt{q} \rfloor + 1 \text{ and } i, j \in \{0, \ldots, m-1\}.$$

In order to find these integers i, j, one creates two different lists of size m: the *baby-steps*

$$g_0 = 1,\ g_1 = g,\ \ldots,\ g_{m-1} = g^{m-1}$$

and the *giant-steps*

$$A_0 = A,\ A_1 = g^{-m} \cdot A,\ \ldots,\ A_{m-1} = (g^{-m})^{m-1} \cdot A.$$

Then one searches a matching element in both lists, i.e. a match $g_i = g^i = (g^{-m})^j \cdot A = A_j$. In this case

$$g^{jm+i} = A$$

which provides us with the solution $\mathsf{dlog}_g(A) = jm + i$.

Algorithm 1 Shank's baby-step giant-step algorithm

Input: $g \in \mathbb{F}_p^*$ with $ord(g) = q$ and $A \in \langle g \rangle$
Output: $a = \mathsf{dlog}_g(A)$

1: $m \leftarrow \lfloor \sqrt{q} \rfloor + 1$
2: $g_0 \leftarrow 1$
/*baby steps*/
3: **for** $i = 1, \ldots, m-1$ **do**
4: $\quad g_i \leftarrow g \cdot g_{i-1}$ ▷ $g_i = g^i$
5: **end for**
6: $\tilde{A} \leftarrow A$
/*giant steps*/
7: **for** $j = 0, \ldots, m-1$ **do**
8: $\quad$ **if** $\tilde{A} = g_i$ for some i **then**
9: $\quad\quad$ **return** $jm + i \pmod q$
10: $\quad$ **else**
11: $\quad\quad \tilde{A} \leftarrow \tilde{A} \cdot g^{-m}$ ▷ $\tilde{A} = A \cdot g^{m(j+1)}$
12: $\quad$ **end if**
13: **end for**

The procedure is sketched in Algorithm 1. Note that in this version, only the first list $(g_0, \ldots, g_{m-1})$ is stored. Whereas for the second list $(A_0, \ldots, A_{m-1})$, one immediately checks after the computation of an element A_j whether there is a match with one of the g_i's. In particular, there is no need to store the elements of the second list.

Proposition 2.10

On input $g \in \mathbb{F}_p^*$ with $ord(g) = q$ and $A \in \langle g \rangle$, Algorithm 1 finds $a = \mathsf{dlog}_g(A)$. The number of $\mathbb{F}_p^*$ multiplications is $O(\sqrt{q})$, and the required memory is $O(\sqrt{q})$ as well.

Proof. Let us first look at the running time. Creating the first list (Lines 2 to 5) takes $m - 1 \approx \sqrt{q}$ multiplications. Each step in the second for-loop (Lines 7 to 13) requires one multiplication by g^{-m}, hence the number of multiplications from this part are at most $m - 1 \approx \sqrt{q}$ as well. Note that the check in Line 8 can be done efficiently assuming that the values for (g_i, i) are stored in a look-up table. The required memory storage is determined by the number of elements in the first list: $m \approx \sqrt{q}$.

It remains to show that the algorithm outputs the correct solution. To see this, note that for any element $a \in \{0, \ldots, q-1\}$, there exist $i^*, j^* \in \{0, \ldots, m-1\}$ so that

$$a = j^* \cdot m + i^*.$$

This solution leads to a collision of the two lists, and in particular the condition $\tilde{A} = g_{i^*}$ (Line 8) will be satisfied in iteration $j = j^*$. □

Example 2.11. *Let $g = 4 \in \mathbb{F}_{83}$. This generates a subgroup of order 41. We are given the challenge $A = 68$. Here, we illustrate the computation of $a = \mathsf{dlog}_g(A)$ using the baby-step giant step algorithm (Algorithm 1). A sketch is provided in Figure 5.*

First, we set

$$m = \lfloor \sqrt{41} \rfloor + 1 = 7.$$

Then the baby steps are computed:

$$g_0 = 1, g_1 = 4, g_2 = 16, g_3 = 64, g_4 = 7, g_5 = 28, g_6 = 29.$$

Now, we compute the giant steps until a collision is found

$$A_0 = 68, A_1 = 75, A_2 = 40, A_3 = 49, A_4 = 4.$$

The collision is given by $g_1 = A_4$ which allows us to deduce $\mathsf{dlog}_g(A)$:

$$g^1 = A \cdot g^{-4m} \quad \Leftrightarrow \quad A = g^{4m+1} = g^{29} \quad \Leftrightarrow \quad \mathsf{dlog}(A) = 29.$$

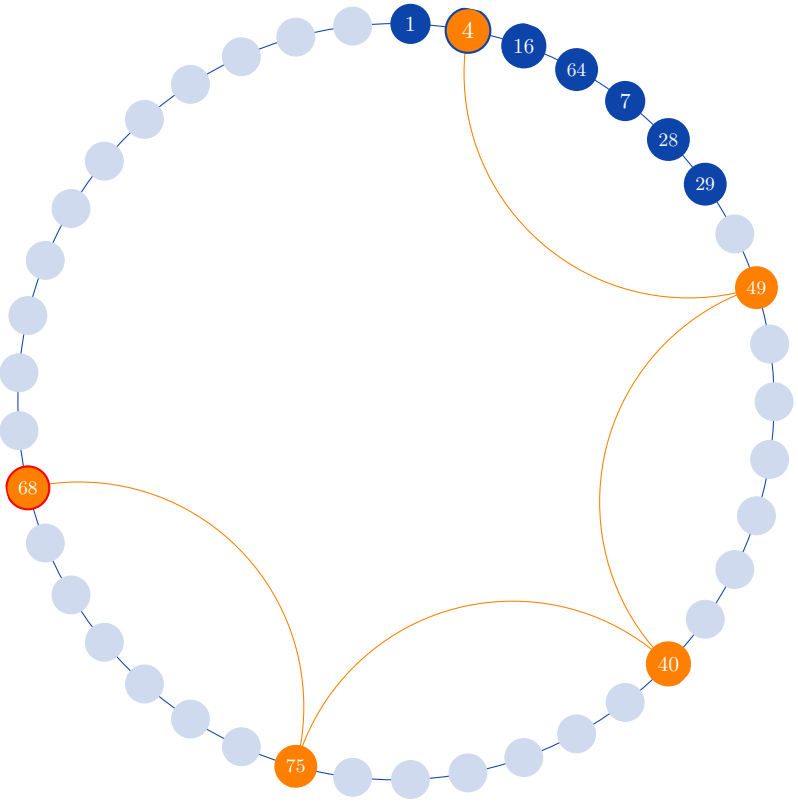


Figure 5: Illustration of the baby-step giant-step algorithm, Example 2.11

Remark 2.12. *As stated in Proposition 2.10 the memory complexity of the baby-step giant-step algorithm is $O(\sqrt{q})$. For a large order q, and in particular for cryptographic applications, this is unrealistic. In order to obtain a version of the baby-step giant-step algorithm with smaller memory requirements, one may choose a smaller value of $m \ll \sqrt{q}$, and find a representation of the form*

$$a = jm + i, \quad \text{with } i \in \{0, \ldots, m-1\} \text{ and } j \in \{0, \ldots, \lfloor q/m \rfloor\}.$$

However, note that this increases the running time of the algorithm. The number of $\mathbb{F}_p$-multiplications is now given by $O(q/m)$. Such time-memory trade-offs *are often considered in cryptanalysis.*

Exercise 2.13. *We have seen in Remark 2.12 that one may reduce the size of the required memory at the cost of increasing the overall runtime of the algorithm. In this exercise, the goal is to achieve the opposite: decreasing the runtime at the cost of increasing the required memory.*

(a) We have shown that Algorithm 1 requires $O(\sqrt{q})$ multiplications. More concretely, show that the average runtime is given by $T = 3/2\sqrt{q}$ (if the exponent a is chosen uniformly at random).

Now consider a variant of Algorithm 1, where the baby steps and giant steps are computed in parallel, and all values g_i, A_i for $0 \leq i \leq n \leq m$ are stored until a match is found for some n.[4]

(b) What is the average runtime of this variant of Algorithm 1? What is the required memory?
Hint: You can use (or prove if you are familiar with probability theory) that for two integers i, j uniformly chosen at random from $\{0, \ldots, m\}$, the expected value of $\max(i, j)$ is $\approx 2/3m$

Exercise 2.14. SAGE *Implement the babystep-giantstep algorithm and use it solve the* DLP *instances from Exercise 2.4. How does the running time compare to the* `log` *function in SageMath? Which algorithm is used in SageMath to solve the* DLP*?*

2.3.2 Pollard's rho algorithm

In 1978, John Pollard suggested the now called *Pollard's rho algorithm* to compute discrete logarithms [22]. This algorithm has the same asymptotic running time as the babystep-giantstep algorithm, its major advantage is that it only requires a constant amount of memory.

Pollard's rho algorithm is a *randomized* algorithm, and based on finding a collision. In some sense, the baby-step giant-step algorithm can also be viewed as a collision-finding algorithm. It is based on finding a collision between the baby-steps and the giant-steps. However, note that this is a *deterministic* approach: the definition of the baby steps and giant steps guarantees a specific collision. In contrast, Pollard's rho algorithm is a *Las Vegas algorithm*: It always produces a correct output, but the running time depends on the input.

Say we want to solve the DLP instance (g, A) with $g, A \in \mathbb{F}_p^*$. Then Pollard's algorithm uses a function $f : \mathbb{F}_p^* \to \mathbb{F}_p^*$ which sends elements of the form $g^k A^\ell$ to $g^{k'} A^{\ell'}$, always keeping track of the exponents. More explicitly, in the original paper [22], Pollard suggests to use

$$f : x = g^k A^\ell \quad \mapsto \quad x' = \begin{cases} g^k A^{\ell+1} & \text{if } 0 < x < p/3, \\ g^{2k} A^{2\ell} & \text{if } p/3 < x < 2p/3, \\ g^{k+1} A^\ell & \text{if } 2p/3 < x < p. \end{cases} \tag{2}$$

Using this function, one creates a sequence

$$x_0 = g^0 A^0,\ x_1 = f(x_0) = g^{k_1} A^{\ell_1},\ x_2 = f(x_1) = g^{k_2} A^{\ell_2},\ \ldots$$

Given that $\mathbb{F}_p^*$ is finite, this sequence is necessarily periodic. More precisely, it will enter into a loop. This is sketched in Figure 6, and explains the name of the algorithm. As in the sketch, we refer to *the tail length T* of a sequence, and the *loop length L*.

A collision in the sequence $(x_i)_{i \in \mathbb{N}}$, can (almost certainly) be used to solve the DLP instance. To see this, assume $x_i = x_j$ for some $i \neq j$. Then

$$g^{k_i} A^{\ell_i} = g^{k_j} A^{\ell_j} \Leftrightarrow g^{k_i - \ell_j} = A^{k_j - \ell_i}.$$

Now if $\ell_j - \ell_i$ is prime to the order of g, then

$$\mathsf{dlog}_g(A) = a \quad \text{with } a \equiv (k_i - k_j) \cdot (\ell_j - \ell_i)^{-1} \pmod{ord(g)}.$$

As we discussed earlier, typically $ord(g) = q$ is a prime, in which case $(\ell_j - \ell_i)$ is almost certainly invertible.

Example 2.15. *We go back to Example 2.11. Recall that we are given a* DLP *challenge with $g = 4 \in \mathbb{F}_{83}$ and $A = 68$. Let's compute the sequence $x_0, x_1, \ldots$ obtained with the function f in Equation 2. Note that more formally, this function should be viewed as a function with domain and codomain $\mathbb{F}_p^* \times \mathbb{Z}/q\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$, as we want to keep track of the exponents k and ℓ as well. In particular, we denote $x_{i+1}, k_{i+1}, \ell_{i+1} = f(x_i, k_i, \ell_i)$.*

[4] This algorithm is proposed in [23]. Apart from the analysis of different variants of algorithms to solve the DLP, this article also explains how some of the results can be used to improve your *monopoly* skills.

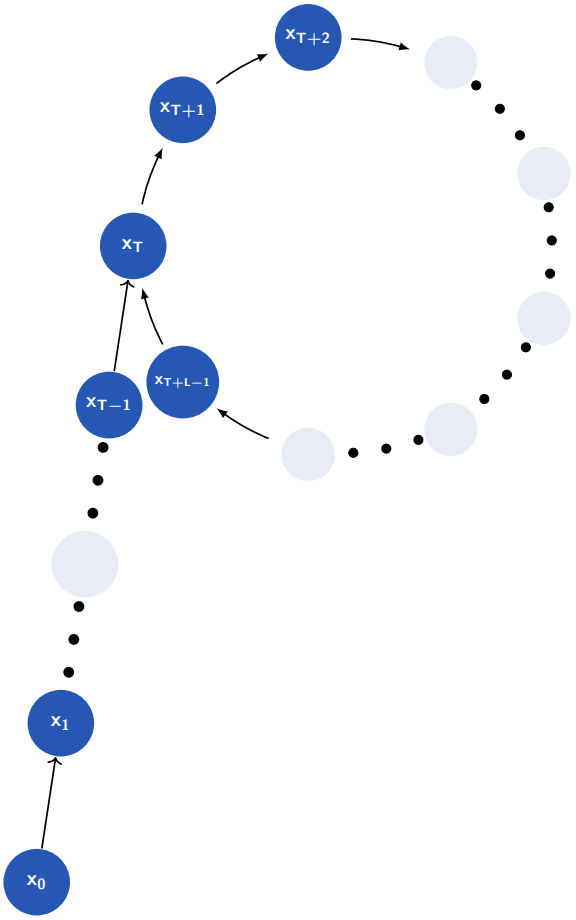


Figure 6: A typical Pollard's rho with tail length T and loop length L.

Starting with $(x_0, k_0, \ell_0) = (1, 0, 0)$*, we obtain the following sequence*

tail of length $T = 4$

$$\begin{cases} (x_0, k_0, \ell_0) & = (1, 0, 0) \\ (x_1, k_1, \ell_1) & = (68, 0, 1) \\ (x_2, k_2, \ell_2) & = (23, 1, 1) \\ (x_3, k_3, \ell_3) & = (70, 1, 2) \end{cases}$$

loop of length $L = 9$

$$\begin{cases} (x_4, k_4, \ell_4) & = (31, 2, 2) \\ (x_5, k_5, \ell_5) & = (48, 4, 4) \\ (x_6, k_6, \ell_6) & = (63, 8, 8) \\ (x_7, k_7, \ell_7) & = (3, 9, 8) \\ (x_8, k_8, \ell_8) & = (38, 9, 9) \\ (x_9, k_9, \ell_9) & = (33, 18, 18) \\ (x_{10}, k_{10}, \ell_{10}) & = (10, 36, 36) \\ (x_{11}, k_{11}, \ell_{11}) & = (16, 36, 37) \\ (x_{12}, k_{12}, \ell_{12}) & = (9, 36, 38) \end{cases}$$

$$(x_{13}, k_{13}, \ell_{13}) = (31, 36, 39)$$

$$(x_{14}, k_{14}, \ell_{14}) = (48, 31, 37)$$

$$\dots$$

The first collision occurs at $x_{13} = x_4$*. This collision provides us with the identity*

$$g^2 A^2 = 31 = g^{36} A^{39} \quad \Leftrightarrow \quad g^7 = g^{2-36} = A^{39-2} = A^{37}.$$

Note that 37 *is invertible modulo* $q = 41$ *with multiplicative inverse* $37^{-1} \equiv 10 \pmod{q}$*, hence we find* $\mathsf{dlog}_g(A) = 7 \cdot 10 \equiv 29 \pmod{q}$.

In Example 2.15, we followed a trivial approach to find the first collision which occurs for $x_T = x_{L+T}$. We just stored all computed values x_i until the collision is found. However, this approach requires a large amount of memory which we want to avoid. A smarter strategy is needed to detect collisions. A second open question concerns the length of the tail and the loop which determine the runtime of the algorithm. These questions will be answered in the following theorem.

Theorem 2.16: Abstract version of Pollard's ρ method as in [12, Theorem 4.47]

Let S be a finite set of cardinality N, and let $f : S \to S$ be a map. Consider the sequence

$$x_0,\ x_1 = f(x_0),\ x_2 = f(x_1),\ \ldots$$

for some initial value $x_0 \in S$.

(a) Suppose that the sequence $(x_i)_i \in \mathbb{N}$ has a tail of length T and a loop of length L. Then

$$x_{2i} = x_i \quad \text{for some } 1 \leq i \leq T + L.$$

(b) If the map f is sufficiently random, then the expected value of $T + L$ is

$$E(T + L) \approx 1.2533 \cdot \sqrt{N}$$

Proof. We will prove the first part, but only sketch the proof of the second part.

(a) To see this, it is easiest to look at Figure 6. We have that $x_{2i} = x_i$ for some i if and only if

$$i \geq T \quad \text{and} \quad i \equiv 2i \pmod{L}.$$

Note that the second condition is equivalent to $i \equiv 0 \pmod{L}$. There is precisely one value $T \leq i \leq T + L - 1$ satisfying these properties.

(b) Since the function f is assumed to be random, one may view the elements $x_0, x_1, \ldots$ as random elements chosen from S. We want to estimate the number of elements that have to be chosen until a collision occurs. Note that this is related to the *birthday paradox*. And roughly, it tells us that it is enough to choose around $\sqrt{N}$ elements to obtain a collision with some positive probability.
The details of the proof for this part involve techniques from probability theory that are not a prerequisite for this course. The reader is kindly referred to [12, Theorem 4.47].

□

Using the results from Theorem 2.16, we can now formulate a version of Pollard's rho algorithm that has running time $O(\sqrt{N})$ and only requires a constant amount of memory. This is made explicit in Algorithm 2. A subtlety is the definition of a *sufficiently random* function f. Here, we use the function from Eq. 2 which was suggested by Pollard. But we note that follow-up works by Teske suggest the use of different functions with better mixing properties [33],[34]. To ease notation, we view f as a function $f : \mathbb{F}_p^* \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{F}_p^* \times \mathbb{Z} \times \mathbb{Z}$, with $f(x, k, \ell) = (x', k', \ell')$ as defined in Equation 2.

Algorithm 2 Pollard's rho algorithm

Input: $g \in \mathbb{F}_p^*$ with $ord(g) = q$ and $A \in \langle g \rangle$
Output: $\mathsf{dlog}_g(A)$

1: $(x, k, \ell) = (1, 0, 0)$ ▷ initial value x_0
2: $(x', k', \ell') = (x, k, \ell)$
3: **while** True **do**
4: $(x, k, \ell) \leftarrow f(x, k, \ell)$ ▷ sequence element $x = x_i$
5: $(x', k', \ell') \leftarrow f(x', k', \ell')$
6: $(x', k', \ell') \leftarrow f(x', k', \ell')$ ▷ sequence element $x' = x_{2i}$
7: **if** $x' = x$ **then**
8: **if** $\gcd(\ell' - \ell, q) = 1$ **then**
9: **return** $\mathsf{dlog}_g(A) = (k - k')(\ell' - \ell)^{-1} \pmod q$
10: **else**
11: go back to Line 1 and start with a different initial value.
12: **end if**
13: **end if**
14: **end while**

Example 2.17. *We go back to Example 2.15, where we computed the structure of* Pollard's rho *for the* DLP *challenge with* $g = 4 \in \mathbb{F}_{83}^*$ *and* $A = 68$. *Applying Pollard's rho algorithm (Algorithm 2), we find a collision after the 9th step, more precisely*

$$(x_9, k_9, \ell_9) = (33, 18, 18), \quad (x_{18}, k_{18}, \ell_{18}) = (33, 3, 27).$$

This provides us with the solution

$$\mathsf{dlog}_g(A) \equiv (18 - 3) \cdot (27 - 9)^{-1} \equiv 29 \pmod q.$$

Note that as claimed in the proof of Theorem 2.16, the first collision occurs for the minimal value i *with* $i > T = 4$ *and* $i \equiv 0 \pmod{L = 9}$.

2.4 Index-calculus

The algorithms discussed in Subsection 2.3 all have exponential running time. Another method, *index calculus*, for computing discrete logarithms in finite fields was already mentioned in 1922 in a book about number theory by Maurice Kraitchik [17]. Here, the word *index* is just another word for *discrete logarithms*.

In contrast to the previously discussed methods, the index calculus algorithm has subexponential running time. Subexponential means that for any base $b > 1$, the running time $T(x)$ for an instance of size x is bounded by $T(x) < b^x$ for all sufficiently large values of x. More precisely, the running time of the index calculus algorithm to compute discrete logarithms in the finite field $\mathbb{F}_p$ is given by $O(2^{c\sqrt{\log p \log\log p}})$ for some constant c.

One of the main ingredients for index calculus is the possibility to lift elements from the finite field $\mathbb{F}_p$ to the integers $\mathbb{Z}$. In particular, one may lift an element $x \in \mathbb{F}_p$ to $\mathbb{Z}$, perform some operations in $\mathbb{Z}$ and reduce modulo p to obtain an element in $\mathbb{F}_p$ again. This procedure is compatible with finite field operations, in the sense that performing the same operations in $\mathbb{F}_p$ yields the same result.

$$\begin{array}{ccc} x \in \mathbb{F}_p & \xrightarrow{lift} & \hat{x} \in \mathbb{Z} \\ \vdots & & \downarrow f \\ y \in \mathbb{F}_p & \xleftarrow[reduce]{} & \hat{y} \in \mathbb{Z} \end{array}$$

A priori, computations in $\mathbb{Z}$ are more expensive than in $\mathbb{F}_p$. The advantage of the idea is that one may exploit properties of the integers that are not present in finite fields. In the case of index calculus, one uses the fact that $\mathbb{Z}$ is a unique factorization domain.

Algorithm 3 provides a brief description of the method (following [32, Algorithm 10.1]). As in the previous section, we want to solve a DLP instance with $g, A \in \mathbb{F}_p^*$.[5]

Algorithm 3 Index calculus algorithm

Input: $g \in \mathbb{F}_p^*$ and $A \in \langle g \rangle$
Output: $\mathsf{dlog}_g(A)$

1: Choose a smoothness bound B and let $\mathcal{P}_B = \{p_1, \ldots, p_b\}$ be the set of primes less than or equal to B.
/*Phase 1: Relation creation in $(\mathbb{Z}/(p-1)\mathbb{Z})[x_1, \ldots, x_{b+1}]$*/
2: $\mathcal{R} \leftarrow \{\}$
3: **while** $\#\mathcal{R} < b+1$ **do**
4: $\quad e \xleftarrow{\$} \{1, \ldots, p-1\}$
5: $\quad$ **if** $g^e/A = \prod_{i=1}^{b} p_i^{e_i}$ **then**
6: $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{e = \sum_{i=1}^{b} e_i x_i + x_{b+1}\}$
7: $\quad$ **end if**
8: **end while**
/*Phase 2: Linear algebra*/
9: Solve the linear system $\mathcal{R}$ for $x_{b+1} \in \mathbb{Z}/(p-1)\mathbb{Z}$.
10: **if** Solution for x_{b+1} in Line 9 is unique **then**
11: $\quad$ **return** x_{b+1}
12: **else**
13: $\quad$ Go back to Line 2
14: **end if**

In the setup for the algorithm, a factor bases $\mathcal{P}_B$ containing all primes less than or equal to B is chosen. The first phase consists of finding linear relations between $\mathsf{dlog}_g(A)$ and $\mathsf{dlog}_g(p_i)$ with $p_i \in \mathcal{P}_B$. To this end, one selects an integer $e \in \{1, \ldots, p-1\}$ at random and checks whether the lift of g^e/A to $\mathbb{Z}$ is B-*smooth*, i.e. whether the factorization is of the form

$$\widehat{g^e/A} = \prod_{i=1}^{b} p_i^{e_i} \in \mathbb{Z}, \quad \text{for some exponents } e_i \in \mathbb{Z}.$$

Taking logarithms, this provides us with the linear relation

$$e - \mathsf{dlog}_g(A) = \sum_{i=1}^{b} e_i \mathsf{dlog}_g(p_i).$$

Note that we do not know $\mathsf{dlog}_g(p_i)$ either. However, after finding enough linear relations of this form, we obtain an overdetermined system which can then be solved using linear algebra methods. This is the second phase of the algorithm.

Remark 2.18. *The above discussion explains that the index calculus method described in Algorithm 3 works correctly. However, several details are missing in order to analyze the runtime. An important question concerns the choice of the smoothness bound B. On the one hand, choosing B large makes it more likely that g^e/A is B-smooth for some random integer e. On the other hand, checking that an integer is B-smooth will be more costly. The running time of the entire algorithm will crucially depend on this choice. More precisely, the running time depends on the following values:*

- *The number of required linear relations. This number is $b+1$ which is roughly $b \approx B/\log B$.*
- *The (expected) number of iterations that are needed in the* `while` *loop to find $b+1$ relations: This number can be estimated using the Dickman–de Bruijn function which describes the asymptotic proportion of smooth numbers.*
- *The time to check if a number is B-smooth (Line 5): Using trial division, one would need to perform b divisions.*

[5] In contrast to the algorithms from Subsection 2.3, here the order of q is not important for the running time of the algorithm.

- *The cost to solve the linear system (Line 9)*

One can show that in this setting, a choice of

$$B \approx e^{1/2\sqrt{\log p \log\log p}}$$

leads to a running time in

$$O\left(e^{2\sqrt{\log p \log\log p}}\right).$$

As claimed in the beginning of the subsection, this running time is subexponential. Note that the square root in the exponent is crucial here; without it, the expression would be exponential.

For a detailed discussion, and further improvements, we refer to [32, §10.3]

Example 2.19. *Let's look at our running example from the previous section (Examples 2.11, 2.17), where we want to compute* $a = \mathsf{dlog}_g(A)$ *with* $g = 4, A = 68 \in \mathbb{F}_{83}^*$. *As a factor base, we choose* $\mathcal{P}_B = \{2, 3, 5\}$.

To simulate the first phase of the algorithm, we choose random exponents $e \in \{1, \dots, 82\}$ *(using the SageMath function* `ZZ.random(1,82)`*), and check whether* g^e/A *is 5-smooth until we have four different linear relations. The results are summarized in the following table. The first column contains the random exponent* e*, and the second column contains the factorization of* g^e/A *if it is* B*-smooth.*

exponent e	g^e/A		*exponent* e	g^e/A
15	5^2		31	2^4
59	×		33	×
60	3^2		17	×
7	×		(72)	(2^4)
40	×		43	$2 \cdot 5$

Note that the relation $g^{72}/A = 2^4$ *does not provide any new information, since* $ord(g) = 41$ *and so* $g^{72} = g^{31}$. *The latter exponent was already in the list.*

Slightly deviating from the notation in Algorithm 3, we denote

$$x_2 = \mathsf{dlog}_4(2), \quad x_3 = \mathsf{dlog}_4(3), \quad x_5 = \mathsf{dlog}_4(5), \quad \text{and} \quad x_A = \mathsf{dlog}_4(68).$$

Then the results from the table induce the following system of equations over $\mathbb{Z}/82\mathbb{Z}$

$$\begin{cases} 15 = & & 2x_5 + x_A \\ 60 = & 2x_3 & + x_A \\ 31 = & 4x_2 & + x_A \\ 43 = & x_2 + x_5 & + x_A \end{cases}$$

Solving for x_A*, we find the solution* $x_A = 29 \equiv 70 \pmod{41}$.

Note that the system of equations in the example is very sparse which makes it easier to solve by hand. Indeed, this is typically the case for large parameter sets as well. In particular, one may exploit the sparseness of the set of equations to implement the linear algebra step in Algorithm 3 more efficiently.

Exercise 2.20. *Using the system of equations from Example 2.19, can you also solve for* x_2, x_3 *and* x_5*? Explain your observations.*

2.5 Diffie-Hellman for generic groups

Many of the algorithms and cryptographic constructions that we have seen are not specific to working in a finite field, but one could formulate them for any group. In this section, let $\mathbb{G}$ be a group equipped with the group operation $\circ$. The definition of the DLP (Definition 2.1) generalizes to the group setting as follows:

Definition 2.21: Group Discrete Logarithm Problem (Group-DLP)

Let $g \in \mathbb{G}$ and $A \in \langle g \rangle$. The *Group Discrete Logarithm Problem* (Group-DLP) is the problem of finding an integer a such that

$$A = \underbrace{g \circ \cdots \circ g}_{a \text{ times}}.$$

The number a is called the *discrete logarithm* of A to the base g, we denote $a = \mathsf{dlog}_g(A)$.

In analogy with the finite field setting, we use the notation

$$\mathsf{exp}_g(a) = \underbrace{g \circ \cdots \circ g}_{a \text{ times}}.$$

Furthermore, if the context is clear, we simply write DLP instead of Group-DLP.

One can define the Diffie-Hellman protocol for any general group $\mathbb{G}$. With the notation that we just introduced, one obtains precisely the same description as in the finite field setting (see Figure 3).

For cryptographic applications, we require that exp_g is a cryptographic one-way function (Definition 1.4). This brings us to the question how hard it is to solve DLP in a general group $\mathbb{G}$. Algorithms that can be used for any group $\mathbb{G}$ are called *generic*. Both, the baby-step giant-step algorithm and Pollard's rho algorithm from Subsection 2.3, are generic. They do not use any properties specific to finite fields. In contrast to that, the index-calculus approach uses specific properties of finite fields and the integers, and cannot be applied to a generic group.

Moreover, we note that in an idealized model known as the *generic group model*, one can even show that Pollard's rho algorithm is optimal for generic groups [29]. Given that the index-calculus approach solves the DLP much faster in finite fields, this motivates the study of alternative groups in cryptography which may be closer to being *generic*. A widely used alternative are *elliptic curve groups* which will be studied in Section 3.

3 Elliptic curves and cryptography

Elliptic curves are objects coming from algebraic geometry. They play a central role in Wiles' proof of Fermat's last theorem, but also find applications in computational number theory, for instance in factoring and for primality testing. In 1985, it was first suggested by Neil Koblitz [16] and Victor S. Miller [21] to use elliptic curves as a replacement for finite fields in public key cryptography. Today, these ideas are commonly used in modern cryptographic protocols.

The first part of this section provides a brief introduction to the rich theory of elliptic curves. The focus is on elliptic curves defined over finite fields, since these are the objects used in cryptography. In the second part, we discuss cryptographic applications.

3.1 Introduction to elliptic curves

There are different equivalent ways to define an elliptic curve. In algebraic geometry it is usually defined as a *smooth projective curve of genus one with a specified base point*. We will work with the following more concrete definition.

Definition 3.1: Elliptic Curve

Let K be a field of characteristic $p \neq 2, 3$. Then an *elliptic curve* E defined over K consists of a point at infinity and points (x, y) in the plane satisfying an equation of the form

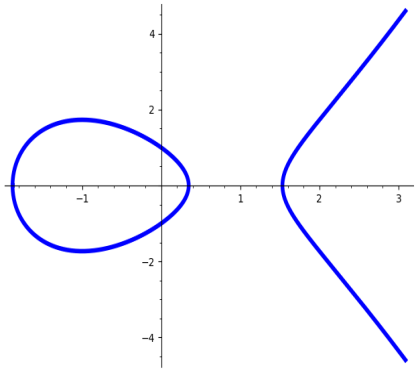
$$y^2 = x^3 + ax + b \quad \text{with } a, b \in K \text{ and } 4a^3 + 27b^2 \neq 0.$$

An equation of the form $y^2 = x^3 + ax + b$ as in Definition 3.1 is called *(short) Weierstrass equation*. If one wants to work over a field K with $\mathrm{char}(K) \in \{2, 3\}$, then it is necessary to work with different types

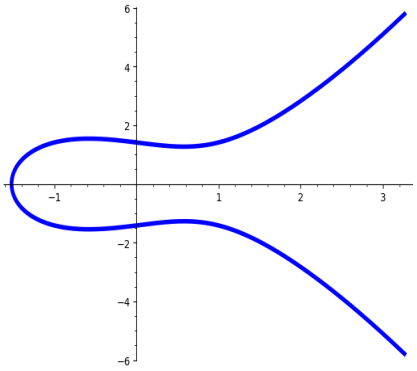
of equations, see for example [30, Appendix A]. However, such curves will not appear in these lecture notes.

The condition $4a^3 + 27b^2 \neq 0$ ensures that the corresponding elliptic curve is smooth. Associated to this, one defines the element $\Delta = -16(4a^3 + 27b^2)$ called the *discriminant of* E. Here the factor -16 is necessary in order to be compatible with the definitions over finite fields of small characteristic.

Two sketches of elliptic curves defined over the real numbers $\mathbb{R}$ are provided in Figure 7.



(a) $E_1/\mathbb{R}$ defined by $y^2 = x^3 - 3x + 1$.



(b) $E_2/\mathbb{R}$ defined by $y^2 = x^3 - x + 2$.

Figure 7: Two typical sketches of elliptic curves

Example 3.2. *Consider the curve* $C : y^2 = x^3 - 3x + 2$. *Here,* $4a^3 + 27b^2 = 0$, *hence the equation does not define an elliptic curve. One can verify that the curve has a singularity at* $P = (1, 0)$. *Broadly speaking, here this means that there are two different tangent directions in the same point. This is illustrated in Figure 8.*

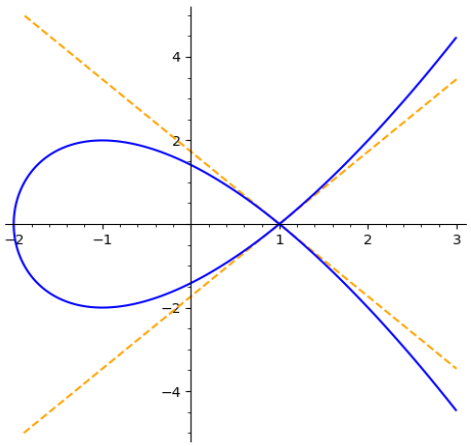


Figure 8: The curve $C : y^2 = x^3 - 3x + 2$ (solid blue) and the tangent lines at the singular point $P = (1, 0)$ (dashed orange)

Exercise 3.3. *Let* $a, b \in K$ *and consider the (affine plane)*[6] *curve* C *defined by* $y^2 = x^3 + ax + b$.

(a) Show that $4a^3 + 27b^2 = 0$ *if and only if the polynomial* $f = x^3 + ax + b$ *has a repeated root.*

(b) A point $P = (x_0, y_0)$ on an affine plane curve is a singularity if and only if both partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ vanish at P; otherwise P is called a smooth point
Use this definition and part (a) to show that all points P *on* C *are smooth if and only if* $4a^3 + 27b^2 \neq 0$.

[6] We do not use the word *elliptic* curve here, since it is not required that $4a^3 + 27b^2 \neq 0$ in this exercise.

3.1.1 Points on elliptic curves

Let E be an elliptic curve defined over K. Depending on the applications, we consider only the points with coordinates in the base field K, or points with coordinates in the algebraic closure $\bar{K}$, or points living in some intermediate field $K \subset L \subset \bar{K}$.

Notation 3.4: Points on elliptic curves

Let $E : y^2 = x^3 + ax + b$ with $a, b \in K$ be an elliptic curve. For any field extension L/K, the set

$$E(L) = \{\infty\} \cup \underbrace{\{(u, v) \subset L^2 \mid v^2 = u^3 + au + b\}}_{\text{affine points}}$$

is called the set of *L-rational points* of E. We denote by $\#E(L)$ the number of L-rational points of E, and we say that $\#E(K)$ is the *order of E*.

The structure of the points of an elliptic curve very much depends on the underlying base field.

Example 3.5. *(a) First consider the curve $E_1 : y^2 = x^3 - 3x + 1$ from Figure 7a. Over $\mathbb{R}$, there are infinitely many points. For example:*

$$\infty,\ P_1 = (0, 1),\ P_2 = (-1, \sqrt{3}),\ P_3 = (2, \sqrt{3}),\ P_4 = (3, \sqrt{19}), \dots$$

Indeed, we may just evaluate the right hand side, $f(x) = x^3 - 3x + 1$, at an arbitrary value x_0, and as long as $f(x_0)$ is positive, there is a solution $y_0^2 = f(x_0)$ in $\mathbb{R}$.
Note that we typically sketch elliptic curves over $\mathbb{R}$, even if we are more interested in different base fields for number theoretic or cryptographic applications.

(b) Now consider the same equation defined over the rationals, i.e. $E_1 : y^2 = x^3 - 3x + 1$ over $\mathbb{Q}$. The first two points of the above list are also defined over $\mathbb{Q}$, and we can easily find a third one: $(0, -1) \in E(\mathbb{Q})$.
Are there more $\mathbb{Q}$-rational points? *The answer is yes, there are even infinitely many $\mathbb{Q}$-rational points. Here are some more examples (that become increasingly more difficult to find by simple guessing):*

$$\left(\frac{9}{4}, \frac{19}{8}\right), \quad \left(\frac{-152}{81}, \frac{107}{729}\right), \quad \left(\frac{12033}{5776}, \frac{-854783}{438976}\right), \quad \dots$$

Not any elliptic curve has infinitely many $\mathbb{Q}$-rational points. For instance consider $E_2 : y^2 = x^3 - x + 2$ from Figure 7b. Over $\mathbb{R}$ there are still infinitely many points, but there is only one $\mathbb{Q}$-rational point: $E_2(\mathbb{Q}) = \{\infty\}$.
In this course, we cannot prove the statements made in this part of the example. The reader interested in learning more about elliptic curves over $\mathbb{Q}$ is referred to [30, Chapter VIII]

(c) Let us now turn to finite fields. We consider the elliptic curve $E_1 : y^2 = x^3 - 3x + 1$ defined over the finite field $\mathbb{F}_{13}$. In this case, there are certainly only finitely many points in $E_1(\mathbb{F}_{13})$. By trying all possible x-coordinates and solving for y, one can check that there are exactly 19 different points, i.e. $\#E_1(\mathbb{F}_{13}) = 19$.

$$E_1(\mathbb{F}_{13}) = \{\infty, (0, 1), (0, -1), (1, 5), (1, -5), (2, 4), \dots\}$$

A plot of these points is provided in Figure 9.

Remark 3.6. Where is the point at infinity coming from and why do we need it? *Roughly speaking, this is a way to "compactify" the elliptic curve. In order to answer this question more rigorously, some algebraic geometry is required. An elliptic curve is a* planar, projective *curve, i.e. it is defined by a* homogeneous polynomial F *in the* projective plane $\mathbb{P}^2$.

- *The set of points of the projective plane, $\mathbb{P}^2(K)$, consists of all non-zero elements $(X, Y, Z) \in K^3 \setminus \{(0, 0, 0)\}$ modulo the equivalence relation $(X, Y, Z) \sim (\lambda X, \lambda Y, \lambda Z)$ for any $\lambda \in K$. The equivalence class of (X, Y, Z), i.e. a point of $\mathbb{P}^2(K)$, is denoted by $(X : Y : Z)$.*
- *A plane projective curve C is defined by a homogeneous polynomial $F \in K[X, Y, Z]$. Homogeneous means that all monomials have the same degree. Points with coordinates $(x : y : 1) \in C(K)$ are called* affine points, *and points with coordinates $(x : y : 0) \in C(K)$ are called* points at infinity.

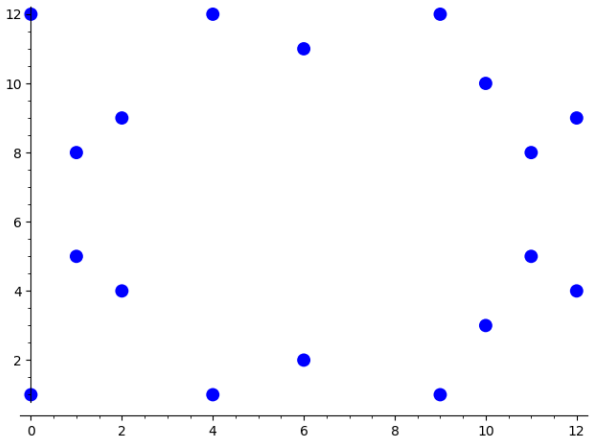


Figure 9: Elliptic curve $E_1 : y^2 = x^3 - 3x + 1$ over $\mathbb{F}_{13}$ from Example 3.5

In our setting ($char(K) \neq 2, 3$), an elliptic curve is then defined by the equation

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3 \quad \text{in } \mathbb{P}^2.$$

The affine points $(x : y : 1)$ just correspond to the points satisfying the (affine) equation $y^2 = x^3 + ax + b$ as in Definition 3.1. Further, one can check that there is exactly one point of the form $(x : y : 0)$; the point at infinity:

$$\infty = (0 : 1 : 0).$$

For more details, we refer to [35, Section 2.3] which provides a very accessible explanation of this topic.

3.1.2 The group law

The points on an elliptic curve form a group. This is something very special, and indeed elliptic curves are the only algebraic curves that enjoy this property. The idea behind the group law is natural from an algebraic geometry point of view. It follows from the theory of *divisors* which will not be discussed in this course. Instead, we work with a more explicit description of the group law.

Theorem 3.7: Group law for elliptic curves

Let $E : y^2 = x^3 + ax + b$ be an elliptic curve defined over K with $\text{char}(K) \neq 2, 3$.
For $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in E(\bar{K})$, define $P_1 + P_2 = P_3 = (x_3, y_3)$ as follows.
(a) If $x_1 \neq x_2$, then $(x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1)$ with $m = \frac{y_2 - y_1}{x_2 - x_1}$.
(b) If $x_1 = x_2$ and $y_1 = y_2 \neq 0$, then $(x_3, y_3) = (m^2 - 2x_1, m(x_1 - x_3) - y_1)$ with $m = \frac{3x_1^2 + a}{2y_1}$.
(c) If $x_1 = x_2$ and $y_1 \neq y_2$ or $y_1 = y_2 = 0$, then $P_1 + P_2 = \infty$.
Moreover, we define $P + \infty = P$ for any point $P \in E(\bar{K})$. Then $(E(\bar{K}), +)$ is an abelian group with identity element ∞.

Proof. First note that swapping the points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in the definition of "+" above does not change the formulas, hence the law is commutative. To see that it is indeed a group law, we need to show the existence of a neutral element, inverses and associativity.

- *neutral element*: By definition, we have that $P + \infty = P$ for any element $P \in E(\bar{K})$, hence the point at infinity is the neutral element.
- *existence of inverses*: Let $P_1 = (x_1, y_1) \in E(\bar{K})$, then its inverse is given by $-P_1 = (x_1, -y_1)$. To see this, first note that $(x_1, -y_1)$ is a point on E as well. To compute $(x_1, y_1) + (x_1, -y_1)$, one then applies the law in (c) which yields ∞.
- *associativity*: Showing that the group law is associative can be done in a rather tedious computation involving several case distinctions. One may use a computer algebra system to verify the associativity symbolically. Explicitly, this strategy is implemented and explained in a SageMath

worksheet in [32, Lecture 2]. The proof becomes much more elegant if one were to use the theory of divisors. However this is out of the scope for this course, and we refer to [30, Section III.3.4] for a proof using these techniques.

□

We defined the group law for points in the algebraic closure $\bar{K}$. Note that adding two K-rational points $P_1, P_2 \in E(K)$, we have $P_1 + P_2 \in E(K)$ as well. This means that $E(K)$ is a subgroup of $E(\bar{K})$, and in particular a group itself. Similarly, for any field extension L/K, the L-rational points $E(L)$ form a group.

Example 3.8. *Consider the curve $E_1 : y^2 = x^3 - 3x + 1$ over $\mathbb{Q}$ from Example 3.5 (b). We have seen that*

$$P_1 = (0, 1), \quad P_2 = \left(\frac{9}{4}, \frac{19}{8}\right)$$

are points in $E(\mathbb{Q})$. To compute $P_1 + P_2$, we apply the law from Theorem 3.7. Here, the first rule applies since $x_1 = 0 \neq 9/4 = x_2$. And we find

$$\begin{aligned} m &= \frac{\frac{19}{8} - 1}{\frac{9}{4} - 0} = \frac{11}{18} \\ x_3 &= \left(\frac{11}{18}\right)^2 - 0 - \frac{9}{4} = \frac{-152}{81} \\ y_3 &= \frac{11}{18}\left(0 - \frac{-152}{18}\right) - 1 = \frac{107}{729}, \end{aligned}$$

hence

$$P_1 + P_2 = \left(\frac{-152}{81}, \frac{107}{729}\right).$$

Note that this is one of the other points from the example. Moreover, you can check that in this example $P_1 + P_1 = P_2$ applying the second rule of the law in Theorem 3.7. And indeed, the way we found the different points in Example 3.5(b), was by repeatedly adding (or subtracting) the point P_1 to the known points. That is to say all the points in the example are multiples of P_1.

Remark 3.9. *The group law has a nice geometric interpretation. To add two points $P \neq Q$ on an elliptic curve $E : y^2 = x^3 + ax + b$, one proceeds as follows:*

1. *Draw a line L through $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.*
2. *The line L intersects the elliptic curve E in a third point, R.*
3. *Then $P + Q = -R$.*

We discuss the different steps in detail for the generic case and show how it relates to the formulas from Theorem 3.7. For the remaining special cases, we provide a brief description as well.

In the first step, since $P \neq Q$, there is a unique line though the two points. This line is defined by the equation

$$L : x(y_2 - y_1) + y(x_1 - x_2) = x_1y_2 - x_2y_1.$$

Generic case: $x_1 \neq x_2$ (case a). In this case, we can write the equation for L as

$$L_{(a)} : y = m \cdot x + c, \quad \text{with } m = \frac{y_1 - y_2}{x_1 - x_2},\ c = \frac{x_1y_2 - x_2y_1}{x_1 - x_2}.$$

In particular, the slope m coincides with the value for m in the theorem. In the second step, we compute the intersection $E \cap L$. For the generic case (a), we find

$$\begin{aligned} & R = (x_3, y_3) \in E \cap L_{(a)} \\ \Leftrightarrow\ & y_3^2 = x_3^3 + ax_3 + b \text{ and } y_3 = mx_3 + c \\ \Leftrightarrow\ & (mx_3 + c)^2 = x_3^3 + ax_3 + b \text{ and } y_3 = mx_3 + c. \end{aligned}$$

It remains to solve the equation $(mx+c)^2 = x^3 + ax + b$ *for* x*. Here, we can use that we already know two solutions:* x_1, x_2*, hence*

$$x^3 - m^2x^2 + (a - 2mc)x + (b - c^2) = (x - x_1)(x - x_2)(x - x_3).$$

Comparing coefficients (here the coefficient of x^2*), we find* $x_3 = m^2 - x_1 - x_2$*. And substituting this into the equation defining* $L_{(a)}$*, we obtain*

$$y_3 = mx_3 + c = mx_3 + (y_1 - mx_1) = m(x_3 - x_1) + y_1.$$

The last step then outputs $P + Q = (x_3, -y_3)$ *which coincides with the statement in the theorem.*

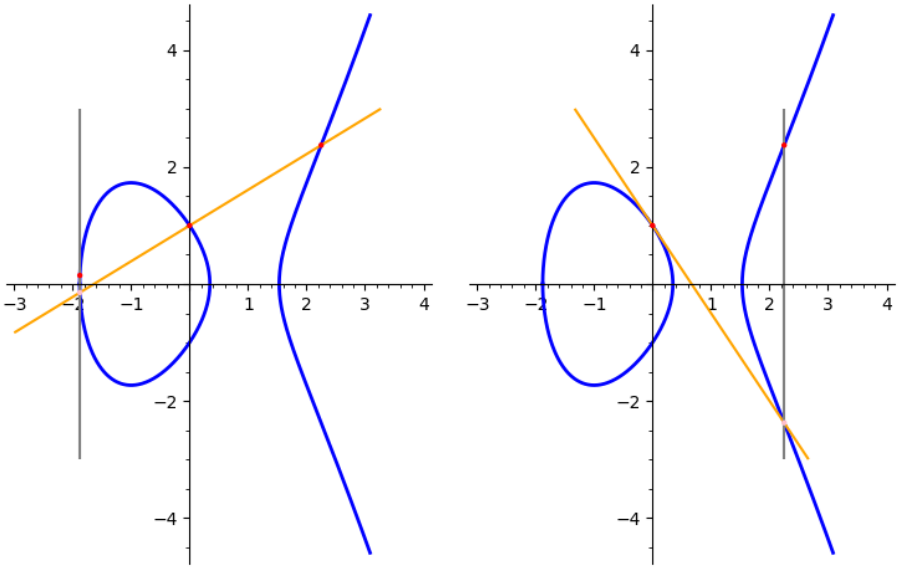
Case (c) If $x_1 = x_2$ *(case c), then the equation for the line* L *simplifies to*

$$L_{(c)} : x = \frac{x_1y_2 - x_2y_1}{y_2 - y_1}.$$

Recall that we assumed $P \neq Q$*, hence* $y_2 \neq y_1$ *here. One can now check that the only two* affine *intersection points are given by* P_1 *and* P_2*. However there is a third intersection point at infinity. To see this, one needs to work with the equations in projective space as in Remark 3.6. This explains that in this case* $P_1 + P_2 = \infty$*.*

Case (b) It remains to cover the case where $P = Q = (x_1, y_1)$*. In this case the first step (drawing a line through* P *and* Q*) has to be understood as drawing the tangent to* E *at* P*. The slope of this tangent line is given by* $m = \frac{3x_1^2+a}{2y_1}$*. Now the remaining steps can be performed in a similar way as in the generic case, and one obtains the formulas from Theorem 3.7.*

An illustration of the geometric interpretation of the group law by means of Example 3.8 is provided in Figure 10.



(a) Addition $P_1 + P_2$ (b) Doubling $P_1 + P_1$

Figure 10: Geometric interpretation of the group law as in Remark 3.9 for the elliptic curve $E_1 : y^2 = x^3 - 3x + 1$ with $P_1 = (0, 1)$, $P_2 = (9/4, 19/8)$ from Example 3.8

3.1.3 Scalar multiplication and torsion

Given that we can add points on an elliptic curve, we can also multiply them by a scalar $N \in \mathbb{Z}$. Implicitly, this property was already used in Example 3.8.

Notation 3.10

Let E be an elliptic curve over K, and $N \in \mathbb{Z}$ an integer. We denote

$$[N] : E(K) \to E(K), \quad P \mapsto \underbrace{P + \cdots + P}_{N \text{ times}},$$

for the scalar multiplication by N.

Exercise 3.11. *Consider the elliptic curve $E : y^2 = x^3 - 3x + 1$ defined over $\mathbb{F}_{13}$ from Example 3.5(c), and let*

$$P_1 = (0, 1) \in E(\mathbb{F}_{13}).$$

(a) Compute $[2] \cdot P_1$. Is there any relation to the point P_2 from Example 3.8?

(b) Compute $[12] \cdot P_1$. Try to use as few elliptic curve additions as possible.

Exercise 3.12. *Given an elliptic curve E over K, a point $P \in E(K)$ and an integer N. Show that Algorithm 4 computes $[N] \cdot P$ using at most $2 \cdot \log_2(N)$ elliptic curve additions (a doubling $[2]P$ is counted as one addition $P + P$).*

Algorithm 4 Double-and-add algorithm

Input: $P \in E(K), N \in \mathbb{N}$,
Output: $[N]P$

1: $(e_0 e_1 \dots e_n)_2 \leftarrow N$, ▷ $N = e_0 + e_1 \cdot 2 + \cdots + e_n \cdot 2^n$, $e_n = 1$
2: $Q \leftarrow P$
3: $R \leftarrow \infty$
4: **for** $i = 0$ **to** $n - 1$ **do**
5: **if** $e_i = 1$ **then**
6: $R \leftarrow R + Q$
7: **end if**
8: $Q \leftarrow [2]Q$
9: **end for**
10: $R \leftarrow R + Q$
11: **return** R

Points on an elliptic curve can have finite order. More precisely, when multiplying a non-zero element $P \in E$ with an integer N, it can happen that $[N]P = \infty$.

Definition 3.13

Let E be an elliptic curve over K, $N \geq 1$ and integer. The group of points of order N is denoted by

$$E[N] = \{P \in E(\bar{K}) \mid [N]P = \infty\}.$$

We say that $E[N]$ is the *N-torsion group* of E.

First, note that $E[N]$ is indeed a group. This follows from the commutativity of the group law on E. Second, we want to point out that the notation $E[N]$ refers to the torsion points over the algebraic closure of the base field. If one wants to only consider the K-rational N-torsion points, the notation $E(K)[N] \subset E[N]$ is used.

Proposition 3.14

Let E be an elliptic curve over K and $N \geq 1$ an integer.

1. If $\mathrm{char}(K) = 0$ or $\mathrm{char}(K) = p$ with $p \nmid N$. Then
$$E[N] \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}.$$
2. If $\mathrm{char}(K) = p > 0$, then one of the following is true:
   (i) $E[p^k] \cong \{\infty\}$ for all $k \geq 1$.
   (ii) $E[p^k] \cong \mathbb{Z}/p^k\mathbb{Z}$ for all $k \geq 1$.

Proof. We refer to [30, Corollary III.6.4] for a proof of this proposition. □

Example 3.15. *Let $E : y^2 = x^3 + ax + b$ be an elliptic curve defined over K, and denote*
$$x^3 + ax + b = (x - \alpha_1)(x - \alpha_2)(x - \alpha_3)$$
with $\alpha_1, \alpha_2, \alpha_3 \in \bar{K}$. Note that the α_i are pairwise distinct as per Exercise 3.3. The points
$$P_i = (\alpha_i, 0) \in E(\bar{K}) \quad \text{for } i = 1, 2, 3,$$
are 2-torsion points (Case (c) of the addition law from Theorem 3.7). It follows from Proposition 3.14 that
$$E[2] = \{\infty, P_1, P_2, P_3\} = \langle P_1, P_2 \rangle.$$
Note that you can also verify explicitly that $P_1 + P_2 = P_3$ using the group law.

3.1.4 Elliptic curves over finite fields

In cryptography, we work with elliptic curves defined over a finite field. In this context, p will always denote a prime number and $\mathbb{F}_p$ is a prime field, whereas q denotes a prime power, i.e. $q = p^k$ for some positive integer k, and $\mathbb{F}_q$ is the finite field with q elements.[7]

An elliptic curve defined over a finite field, only has a finite number of points. Even the (affine) plane over a finite field $\mathbb{F}_q$ only has q^2 points, and the affine points on an elliptic curve are a subset of these. This provides us with some *very* rough bounds on the number of points on an elliptic curve $E/\mathbb{F}_q$
$$1 \leq \#E(\mathbb{F}_q) \leq 1 + q^2.$$
We can make this more precise by noting that for a given x-coordinate $x_0 \in \mathbb{F}_q$, the equation $y^2 = x_0^3 + ax_0 + b$ can have at most two solutions y_0 and $-y_0$ in $\mathbb{F}_q^*$, hence
$$1 \leq \#E(\mathbb{F}_q) \leq 1 + 2q.$$
This is made much more precise in a theorem by Helmut Hasse from 1936 [11].

Theorem 3.16: Hasse's theorem

Let E be an elliptic curve over a finite field $\mathbb{F}_q$. Then
$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}.$$

The proof of Hasse's theorems requires some more advanced knowledge about elliptic curves, and we will not discuss it here.

For some elliptic curves, it is even possible to provide an explicit formula for the number of points depending on the prime p. An example for such a formula is provided in Lemma 3.17. We note that there are no such formulas for arbitrary elliptic curves, but there exists an efficient algorithm, the Schoof–Elkies–Atkin (SEA) point counting algorithm, which can be used to count the number of points of an elliptic curve over a finite fields.

[7] We highlight this here, because this is completely unrelated to the notation $p = 2q + 1$ that was used in Section 2.

Lemma 3.17

Let $p \equiv 2 \pmod 3$ be a prime , and consider the elliptic curve $E : y^2 = x^3 + 1$. Then

$$\#E(\mathbb{F}_p) = p + 1.$$

Proof. In order to find all points on an elliptic curve, our strategy was to try different x-coordinates $x_0 \in \mathbb{F}_p$ and then check whether $x_0^3 + ax_0 + b$ is a square, which then yields a point $(x_0, y_0) \in E(\mathbb{F}_p)$ (Example 3.5(c)). In this proof, we will instead consider different y-coordinates y_0, and check whether $x_0^3 = y_0^2 - 1$ has a solution, i.e. we check whether $y_0^2 - 1$ is a cube.

Which elements in $\mathbb{F}_p$ are cubes? Since, $p \equiv 2 \pmod 3$, all elements are cubes! To see this, first note that, $0 = 0^3$ is a cube. Now let $g \in \mathbb{F}_p^*$, then

$$h = g^{(2p-1)/3} \in \mathbb{F}_{p^*}$$

is a cube root of g. This is well-defined: $2p - 1 \equiv 0 \pmod 3$, hence we can divide by 3. And one can check that

$$h^3 = \left(g^{(2p-1)/3}\right)^3 = g^{2p-1} = g^{2(p-1)+1} = g.$$

Further note that every element in $\mathbb{F}_p$ has a *unique* cube root, since the map $x \mapsto x^3$ is a homomorphism with domain and codomain $\mathbb{F}_p^*$, and we have just shown that it is surjective.

In conclusion, for each $y_0 \in \mathbb{F}_p$, there exists a unique $x_0 \in \mathbb{F}_p$ so that $(x_0, y_0) \in E(\mathbb{F}_p)$. This means that E has p affine points. Adding the point at infinity, we find

$$\#E(\mathbb{F}_p) = p + 1.$$

□

Since elliptic curves over finite fields are finite groups, all points are torsion points (Definition 3.13). We can use Proposition 3.14 to analyze possible group structures.

Proposition 3.18

Let E be an elliptic curve over $\mathbb{F}_q$. Then

$$E(\mathbb{F}_q) \cong \mathbb{Z}/N_1\mathbb{Z} \times \mathbb{Z}/N_2\mathbb{Z}$$

for some integers $N_1, N_2 \geq 1$ and $N_2 \mid N_1$.

Proof. The group $E(\mathbb{F}_q)$ is finite. Let $N = \#E(\mathbb{F}_q)$. Then it holds that $[N] \cdot P = \infty$ for every element $P \in E(\mathbb{F}_q)$, in particular

$$E(\mathbb{F}_q) \subset E[N].$$

Recall from Proposition 3.14 that $E[N]$ is of rank at most 2. More precisely, if $\gcd(N, q) = 1$, then $E[N] \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$. Any subgroup of the latter is of the form $\mathbb{Z}/N_1\mathbb{Z} \times \mathbb{Z}/N_2\mathbb{Z}$ with $N_1, N_2 \mid N$, and $N_2 \mid N_1$.

Now if $\gcd(N, q) = p^k$ for some $k \geq 1$, write $N_0 = N/p^k$. Then

$$E[N] \cong \begin{cases} \mathbb{Z}/N_0\mathbb{Z} \times \mathbb{Z}/N_0\mathbb{Z}, \\ \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N_0\mathbb{Z}. \end{cases}$$

In both cases, any subgroups are of the form $\mathbb{Z}/N_1\mathbb{Z} \times \mathbb{Z}/N_2\mathbb{Z}$ for some $N_1, N_2 \mid N$, and $N_2 \mid N_1$. □

Example 3.19. *Consider $E/\mathbb{F}_5$ defined by $y^2 = x^3 - x$. In the following, we prove that*

$$E(\mathbb{F}_5) = \langle (3, 2), (1, 0) \rangle \cong \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$$

Let us first show that $ord((3,2)) = 4$. *To do this, we compute* $[i] \cdot (3,2)$ *for* $[i] \in \mathbb{N}$ *until we obtain* ∞. *First* $[1] \cdot (3,2) = (3,2) \neq \infty$. *Then we compute* $[2] \cdot (3,2)$ *as* $(3,2) + (3,2)$ *(this is the second case in Theorem 3.7). We find*

$$m = \frac{3 \cdot 3^2 - 1}{2 \cdot 2} = -1, \quad x_3 = (-1)^2 - 2 \cdot 3 = 0, \quad y_3 = -1(3-0) - 2 = 0,$$

i.e. $[2] \cdot (3,2) = (0,0)$. *Next* $[3] \cdot (3,2)$ *is computed as* $(0,0) + (3,2)$. *This is the generic case, and we find*

$$m = \frac{2-0}{3-0} = 4, \quad x_3 = 4^2 - 0 - 3 = 3, \quad y_3 = 4(0-3) - 0 = 3,$$

i.e. $[3] \cdot (3,2) = (3,3)$. *Finally, we compute* $[4] \cdot (3,2)$ *as* $(3,3) + (3,2)$. *This is the third case in Theorem 3.7, hence* $(3,3) + (3,2) = \infty$, *and we have shown that* $ord((3,2)) = 4$.

If immediately follows from the discussion in Example 3.15 that $ord((1,0)) = 2$ *(alternatively, apply the group law). Moreover* $(1,0) \notin \langle (3,2) \rangle = \{(3,2), (0,0), (3,3), \infty\}$. *It follows that*

$$G = \langle (3,2), (1,0) \rangle \subset E(\mathbb{F}_5)$$

is a subgroup of order 8. *On the other hand, Hasse's theorem (Theorem 3.16) implies that* $2 \leq E(\mathbb{F}_5) \leq 10$, *hence* $G = E(\mathbb{F}_5)$.

Remark 3.20. *Determining the group structure for elliptic curves over* $\mathbb{Q}$ *is more complicated, and we will not answer this question in this course. For completeness, we want to mention that for the curves in Example 3.5 (part b), it holds that*

$$E_1(\mathbb{Q}) \cong \mathbb{Z}, \quad E_2(\mathbb{Q}) \cong \{0\}.$$

In general, one can show that $E(\mathbb{Q})$ *is finitely generated (Mordell-Weil Theorem), and there exist precise statements about possible group structures (Mazur's theorem), see [30, Chapter VIII].*

Exercise 3.21. *Let* $p > 3$ *be a prime, and consider the two elliptic curves*

$$E : y^2 = x^3 + ax + b, \qquad \tilde{E} : y^2 = x^3 + ax - b$$

defined over $\mathbb{F}_p$.

(a) Assume that $p \equiv 1 \pmod 4$. *Show that* $\#E(\mathbb{F}_p) = \#\tilde{E}(\mathbb{F}_p)$.

(b) Assume that $p \equiv 3 \pmod 4$. *Show that*

$$\#E(\mathbb{F}_p) + \#\tilde{E}(\mathbb{F}_p) = 2p + 2.$$

Here are some hints/questions that will help find the solution:

- *Is* -1 *a square in* $\mathbb{F}_p$*?*
- *Let* $P = (x_0, y_0) \in E(\mathbb{F}_p)$. *Is there a point* $\tilde{P} = (x_0, \star) \in \tilde{E}(\mathbb{F}_p)$*? What about* $\tilde{P} = (-x_0, \star)$*?*

3.2 Elliptic curve cryptography

Elliptic curve cryptography started with the idea to work in an elliptic curve group $E(\mathbb{F}_q)$ instead of the multiplicative group of a finite field $\mathbb{F}_p^*$, the hope being that scalar multiplication on elliptic curves is a better one-way function than modular exponentiation. We will see that:

- On the one hand, this is not true. There are elliptic curves (of large prime order) for which scalar multiplication is easy to invert.
- On the other hand, this is very true. In the sense that for well-chosen elliptic curves, the best-known methods to invert scalar multiplications are not better than generic methods that work in any group. In particular, there are no known subexponential algorithms (such as index calculus, Subsection 2.4) that work on any elliptic curve.

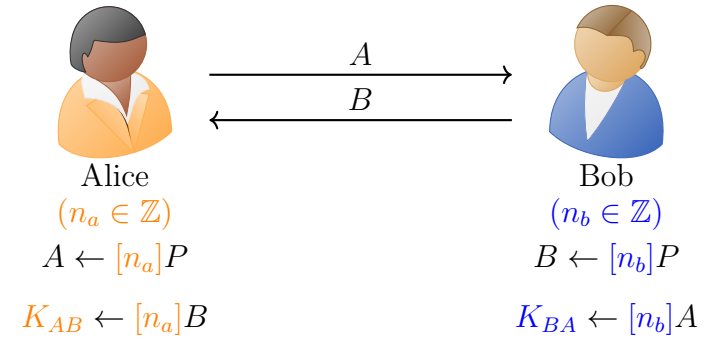


Figure 11: Elliptic Curve Diffie-Hellman key exchange for some public parameter $P \in E(\mathbb{F}_q)$

Concretely, to obtain a key exchange protocol on elliptic curves, we may simply replace all exponentiations in the Diffie-Hellman protocol (Figure 3) by elliptic curve scalar multiplications. We call the resulting protocol the *Elliptic Curve Diffie-Hellman key exchange* (ECDH). This is sketched in Figure 11.

Example 3.22. *Alice and Bob want to create a shared session key using* ECDH. *As a setup they use the curve* $E : y^2 = x^3 - 3x + 1$ *over* $\mathbb{F}_{13}$ *(see Example 3.5(c)) and the point* $P = (0, 1) \in E(\mathbb{F}_{13})$. *Recall that* $E(\mathbb{F}_{13}) = 19$.

Alice chooses $n_a = 5$ *as her secret key and sends the public key* $A = [5]P = (1, 8)$ *to Bob.*

Bob chooses $n_b = 12$ *as his secret key and sends the public key* $B = [12]P = (4, 1)$ *to Alice.*

Now Alice computes $K_{AB} = [n_a]B = [5](4, 1) = (10, 3)$. *Similarly Bob computes* $K_{BA} = [n_b]A = [12](1, 8) = (10, 3)$.

After this key exchange, Alice and Bob can now use a symmetric crypto system with their shared key $K_{AB} = K_{BA} = (10, 3)$.

In real-world applications, it is required to work with elliptic curves over large fields, and in particular, it is require that they contain a large prime order subgroup. To give an idea of the size of parameters, we provide an example below.

Example 3.23. *The elliptic curve* `Secp256k1` *is currently used in Bitcoin.*[8]. *It is defined by an equation of the form*

$$E : y^2 = x^3 + 7, \quad \text{over } \mathbb{F}_p,$$

where p *is a prime of bit size* 256, *and one can check that the order of* $E(\mathbb{F}_p)$ *is prime. The explicit parameters are provided below.*

```
sage: p = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
sage: Fp = GF(p)
sage: a = Fp(0)
sage: b = Fp(7)
sage: E = EllipticCurve(Fp, (a, b))
sage: P = E(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,  0
   x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)
sage: E.set_order(0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141 * 0x1)
```

The elliptic curve Diffie-Hellman protocol serves as our main motivation in this lecture to study (the conjectured) cryptographic one-way function given by scalar multiplication on elliptic curves. But we remark that the field of Ellitpic Curve Cryptography offers further interesting protocols. For example, the ElGamal encryption scheme (Figure 4) may be instantiated with elliptic curves as well. Moroever, there exists an important signature scheme known as ECDSA. And last but not least, there is *pairing-based cryptography* which provides us with further important cryptographic constructions such as tripartite Diffie-Hellman key exchange, ID-based cryptography, and signatures, see for example [12, 5.10].

[8]Bitcoin does not rely on the elliptic curve Diffie-Hellman protocol, but on `ECDSA`, the Elliptic curve digital signature algorithm.

3.2.1 The elliptic curve discrete logarithm problem (ECDLP)

For our cryptographic applications, we want to use scalar multiplication on elliptic curves as a one-way function, replacing the modular exponentiation from the previous section. We have already seen that scalar multiplication can be evaluated efficiently (Exercise 3.12) using a double and add strategy. For this to be a cryptographic one-way function, we further require that the inversion of scalar multiplication is hard. This is formalized below.

Definition 3.24: Elliptic Curve Discrete Logarithm Problem (ECDLP)

Let $P \in E(K)$ be a point on an elliptic curve E, and let $Q \in \langle P \rangle$. The *Elliptic Curve Discrete Logarithm Problem* (ECDLP) is the problem of finding an integer $n_a \in \mathbb{Z}$ such that $[n_a]P = Q$. The number n_a is called the *discrete logarithm* of Q to the base P, we denote $n_a = \mathsf{dlog}_P(Q)$.

The expression *discrete logarithm* in used in analogy to the definitions of DLP and Group-DLP, even though the group operation on elliptic curves is usually denoted additively.

The important question to study now is the hardness of the ECDLP for different choices of elliptic curves E and points $P \in E$.

Generic methods to solve ECDLP

Clearly, the hardness of the ECDLP depends on the order $N = ord(P)$. Moreover, the Pohlig-Hellman algorithm (Exercise 2.6) may be applied to this setting as well. Therefore, we prefer to work with a point P of prime order.

As we already mentioned in Subsection 2.5, the baby-step giant-step algorithm and Pollard's rho algorithm are generic algorithms and can be used to solve the Group-DLP in any group $\mathbb{G}$. In particular, they can be used to solve the ECDLP in time $O(\sqrt{N})$, where $N = ord(P)$.

Let's sketch the adaption of Pollard's rho algorithm (Algorithm 2) to the setting of elliptic curves. The main ingredient is a map

$$f : \langle P \rangle \times \mathbb{Z} \times \mathbb{Z} \to \langle P \rangle \times \mathbb{Z} \times \mathbb{Z}$$

which satisfies the properties of Theorem 2.16. To do this, one may choose a splitting $X_1 \cup X_2 \cup X_3 = \langle P \rangle \subset E(\mathbb{F}_q)$ into sets of approximately equal size. For instance, one can achieve this by imposing conditions on the x-coordinate of a point. Then the function f is defined as

$$f : R = [k]P + [\ell]Q \to R' = \begin{cases} [k]P + [\ell+1]A & \text{if } R \in X_1, \\ [2k]P + [2\ell]A & \text{if } R \in X_2, \\ [k+1]P + [\ell]A & \text{if } R \in X_3. \end{cases}$$

To apply Theorem 2.16, and show that the resulting algorithm solves ECDLP in time $O(\sqrt{N})$ (using only a constant amount of memory), we need to rely on the heuristic assumption that f behaves like a random function.

On the other hand, the index calculus method (Subsection 2.4) is specific to solving the DLP in finite fields, and does not apply to the ECDLP.

3.2.2 Pairings of elliptic curves

An important tool in elliptic curve cryptography are *pairings*. In full generality, a pairing e takes as input a pair of points from two groups G_1, G_2 and outputs an element in a third group H:

$$e : G_1 \times G_2 \to H, \quad (g_1, g_2) \mapsto h = e(g_1, g_2).$$

The map e is required to be bilinear and non-degenerate. In this subsection, we discuss pairings where $G_1 = G_2 = E[N]$ for some elliptic curve E and integer N, and H is the group of N-th roots of unity, $\boldsymbol{\mu_N}$ contained in some finite field.

Pairings are a tool that can be used to translate computational problems from the groups G_1, G_2 to problems in the group H. Specifically, in our setting, it allows us to translate the ECDLP to DLP in a

finite field. Depending on the parameters, this can lead to significantly better algorithms to solve ECDLP than the generic ones. We will discuss this in Subsection 3.2.3.

The main purpose of this subsection is to introduce the *Weil pairing*, and discuss important properties. We essentially follow the notation from [30], and refer to the latter for proofs and more details.

Determinant pairing

As a warm-up, let us first consider a different kind of pairing, the *determinant pairing*.

Definition 3.25

Let E over $\mathbb{F}_q$ be an elliptic curve and $N \in \mathbb{N}$ a positive integer which is not divisible by $p = \mathrm{char}(\mathbb{F}_q)$. We write

$$E[N] = \langle T_1, T_2 \rangle \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z},$$

and define

$$\det : E[N] \times E[N] \to \mathbb{Z}/N\mathbb{Z}, \quad (aT_1 + bT_2, cT_1 + dT_2) \mapsto ad - bc.$$

We say that det is the *determinant pairing associated to the basis* (T_1, T_2).

While it is quite simple, the determinant pairing already has many of the properties that we need for our applications.

Lemma 3.26

The determinant pairing from Definition 3.25 is

(a) bilinear:

$$\begin{aligned} \det(P_1 + P_2, Q) &= \det(P_1, Q) + \det(P_2, Q), \\ \det(P, Q_1 + Q_2) &= \det(P, Q_1) + \det(P, Q_2), \end{aligned}$$

for all $P_1, P_2, Q_1, Q_2 \in E[N]$

(b) alternating:

$$\det(P, P) = 0 \qquad \text{for all } P \in E[N].$$

(c) nondegenerate:

$$\text{If } \det(P, Q) = 0 \text{ for all } P \in E[N], \text{ then } Q = 0.$$

Proof. The properties of the determinant pairing follow from the properties of the determinant (here of 2×2 matrices). We make this explicit for the first property (linearity in the first component).

Let P_1, P_2, Q be arbitrary points in $E[N]$. We denote

$$P_1 = \alpha_1 T_1 + \beta_1 T_2, \quad P_2 = \alpha_2 T_1 + \beta_2 T_2, \quad Q = \gamma T_1 + \delta T_2.$$

Then

$$\det(P_1 + P_2, Q) = (\alpha_1 + \alpha_2)\delta - (\beta_1 + \beta_2)\gamma = (\alpha_1\delta - \beta_1\gamma) + (\alpha_2\delta - \beta_2\gamma) = \det(P_1, Q) + \det(P_2, Q)$$

The proofs of the remaining properties are left as an exercise to the reader.

□

An important consequence of Lemma 3.26 is the following:

Corollary 3.27

Let det be the determinant pairing on $E[N]$ with respect to some basis (T_1, T_2). Further, let $P \in E[N]$ and $Q \in \langle P \rangle$. Then

$$\det(Q, T) = \mathsf{dlog}_P(Q) \cdot \det(P, T) \qquad (\text{in } \mathbb{Z}/N\mathbb{Z})$$

for all points $T \in E[N]$.

This corollary allows us to reduce the ECDLP to a Group-DLP in the (additive!) group $\mathbb{Z}/N\mathbb{Z}$, provided that we have an efficient way of evaluating det. It would suffice to compute $\det(Q, T)$ and $\det(P, T)$ for some point T with $\det(P, T) \in (\mathbb{Z}/N\mathbb{Z})^*$ (convince yourself that such a T exists, and that this allows you to efficiently compute $\mathsf{dlog}_P(Q)$).

Unfortunately (or fortunately for elliptic curve cryptography), we do not know how to evaluate the determinant pairing efficiently on arbitrary input. The naive strategy would be to apply the definition. However this requires finding a representation in terms of the basis elements for an arbitrary point P, i.e. a representation $P = aT_1 + bT_2$. If we knew how to find such representations efficiently, then we could also solve the ECDLP. So we have not gained anything so far.

Weil pairing

We now turn our attention to the Weil pairing. This pairing was introduced by André Weil in 1940 [36] for the more general setting of Jacobians of algebraic curves - elliptic curves are a special case. It shares similar properties with the determinant pairing, but it is more intrinsic (there is no need to fix a basis). Most importantly, for cryptography, there exists an algorithm to compute it efficiently.

Notation 3.28: Weil pairing

Let E over $\mathbb{F}_q$ be an elliptic curve, and $N \in \mathbb{Z}$ a positive integer coprime to $\mathrm{char}(K) = p$. Further let $\boldsymbol{\mu_N} \subset \bar{\mathbb{F}}_q$ be the group of N-th roots of unity. We denote by

$$e_N : E[N] \times E[N] \to \boldsymbol{\mu_N}$$

the *Weil pairing*.

The formal definition of the Weil pairing already requires a good amount of algebraic geometry. We will not provide this definition here, instead, we first describe some properties of the pairing, and then provide an Algorithm to compute it (Algorithm 6), which could be used as a definition of the pairing. Formal definitions can for instance be found in [12, Section 5.8] or [30, Section III.8].

Proposition 3.29: [30, Proposition III.8.1]

The Weil pairing

$$e_N : E[N] \times E[N] \to \boldsymbol{\mu_N}$$

satisfies the following properties

(a) bilinear:

$$e_N(P_1 + P_2, Q) = e_N(P_1, Q)e_N(P_2, Q),$$
$$e_N(P, Q_1 + Q_2) = e_N(P, Q_1)e_N(P, Q_2).$$

(b) alternating:

$$e_N(P, P) = 1.$$

(c) nondegenerate:

$$\text{If } e_N(P, Q) = 1 \text{ for all } P \in E[N], \text{ then } Q = 0.$$

We note that [30, Proposition III.8.1] describes more properties of the Weil pairing which are not needed for our applications.

Exercise 3.30. *Let $e_N : E[N] \times E[N] \to \boldsymbol{\mu_N}$ be a pairing satisfying the properties from Proposition 3.29. Show that*

$$e_N(P,Q) = e_N(Q,P)^{-1},$$

for all $P, Q \in E[N]$.

Recall that the determinant pairing takes values in the additive group $\mathbb{Z}/N\mathbb{Z}$ whereas the Weil pairing takes values in the multiplicative group $\boldsymbol{\mu_N}$. The relation between the two pairings can be made very explicit:

Exercise 3.31. *Let $e_N : E[N] \times E[N] \to \boldsymbol{\mu_N}$ be the Weil pairing, and $\det : E[N] \times E[N] \to \mathbb{Z}/N\mathbb{Z}$ be the determinant pairing with respect to some basis (T_1, T_2). Further denote*

$$\mu = e_N(T_1, T_2) \in \boldsymbol{\mu_N}.$$

Show that

$$e_N(P,Q) = \mu^{\det(P,Q)} \quad \text{for all } P, Q \in E[N].$$

Hint: Use the properties of the Weil pairing from Proposition 3.29.

For small N, we can write down the Weil pairing directly. Below, we describe the case $N = 2$.

Example 3.32. *Let $E : y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbb{F}_q$, and denote*

$$x^3 + ax + b = (x - \alpha_1)(x - \alpha_2)(x - \alpha_3) \quad \text{over } \bar{\mathbb{F}}_q.$$

Recall from Example 3.15 that the 2-torsion is given by

$$E[2] = \langle (\alpha_1, 0), (\alpha_2, 0) \rangle.$$

We denote

$$P_0 = \infty, \quad P_1 = (\alpha_1, 0), \quad P_2 = (\alpha_2, 0), \quad P_3 = (\alpha_3, 0).$$

Then the pairing

$$e_2 : E[2] \times E[2] \to \{\pm 1\},$$
$$(P_i, P_j) \mapsto \begin{cases} 1 & \text{if } i = j \text{ or } 0 \in \{i, j\} \\ -1 & \text{otherwise} \end{cases}$$

satisfies the properties from Proposition 3.29, i.e. it is bilinear, alternating and nondegenerate.

Evaluating the Weil pairing

This part is dedicated to the description of an algorithm to compute the Weil pairing. For completeness, we will sometimes mention divisors in this part of the lecture notes. This is meant as additional information for students who have some familiarity with algebraic geometry, and should help to provide some intuition for the definition of the algorithm. But we recall that this is not a prerequisite, and one may simply ignore these additional comments. Furthermore, if one is willing to accept that there is a way to efficiently evaluate the Weil pairing, one may also skip this rather technical part of the lecture notes, and continue reading with the part *Embedding degree*.

The main ingredient of the evaluation algorithm is Miller's algorithm (Algorithm 5). Before stating this algorithm, we introduce the following notation.

Notation 3.33

Let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ be affine points on some elliptic curve E, and let m be the slope of the line connecting P, Q, i.e. $m = \frac{y_Q - y_P}{x_Q - x_P}$ if $x_P \neq x_Q$; $m = \frac{3x_P^2 + a}{2y_P}$ if $P = Q$ with $y_P \neq 0$; and $m = \infty$ otherwise.
We set

$$h_{P,Q} = \begin{cases} \frac{y - y_P - m(x - x_P)}{x + x_P + x_Q - m^2} & \text{if } m \neq \infty, \\ x - x_P & \text{if } m = \infty. \end{cases}$$

Note that $h_{P,Q}$ is a function in two variables x, y, and formally, it should be viewed as an element in the function field of the elliptic curve E (see for example [30, §I.1]). In particular, we are only interested in evaluations of the function $h_{P,Q}$ in points on the elliptic curve. Remark 3.34 provides some observations about the evaluation of $h_{P,Q}$ at P, Q and $P+Q$ that motivate its definition. A more precise characterization (using some terminology from algebraic geometry) is then provided in Remark 3.35.

Remark 3.34. *Let P, Q be affine points on some elliptic curve E. In this remark, we want to explain the following two properties of $h_{P,Q}$ in an elementary way (i.e. without using tools from algebraic geometry).*

$$\begin{array}{l} \textit{(i)}\ \ h_{P,Q}(P) = h_{P,Q}(Q) = 0 \\ \textit{(ii)}\ \ h_{P,Q}(P+Q) = \infty. \end{array}$$

In other words, $h_{P,Q}$ vanishes at P and Q and has a pole at $P+Q$. We show this under the simplifying assumption: $P+Q \notin \{-P, -Q, -P-Q\}$.

First assume that $m \neq \infty$, and denote $P+Q = (x_{P+Q}, y_{P+Q})$. Note that $x_{P+Q} = m^2 - x_P - x_Q$ as per Theorem 3.7. Using this information, we find

$$h_{P,Q}(P) = \frac{0}{x_P - x_{P+Q}} = 0, \quad h_{P,Q}(Q) = \frac{y_Q - y_P - m(x_Q - x_P)}{x_Q - x_{P+Q}} = \frac{0}{x_Q - x_{P+Q}} = 0.$$

Here, we used that $x_P \neq x_{P+Q}$ and $x_Q \neq x_{P+Q}$ since $P+Q \notin \{-P, -Q\}$ by assumption, and $P+Q \notin \{P, Q\}$ since P and Q are affine points.

Furthermore,

$$h_{P,Q}(P+Q) = \frac{y_{P+Q} - y_P - m(x_{P+Q} - x_P)}{0}$$

We need to check that the numerator does not vanish. For this, we again use that $x_P \neq x_{P+Q}$ and write the numerator as

$$\left(\frac{y_{P+Q} - y_P}{x_{P+Q} - x_P} - m\right)(x_{P+Q} - x_P).$$

Note that $m' = \frac{y_{P+Q} - y_P}{x_{P+Q} - x_P}$ is the slope of the line through P and $P+Q$. The two slopes m and m' are equal if and only if P, Q and $P+Q$ lie on the same line. However, we know that P, Q and $-(P+Q)$ lie on on the same line (cf. Remark 3.9), hence the only possibility is that $P+Q = -(P+Q)$ which is excluded by the assumptions.

Now assume $m = \infty$. Then $x_P = x_Q$, and it follows that

$$h_{P,Q}(P) = x_P - x_P = 0, \quad h_{P,Q}(Q) = x_Q - x_P = 0.$$

Moreover, $P+Q = \infty$ (case (c) in Theorem 3.7). Intuitively, it seems clear that in this case $x_{P+Q} = \infty$ and $h_{P,Q}(P+Q) = \infty - x_P = \infty$. To make this rigorous, it is necessary to use more formal arguments as in Remark 3.35.

Remark 3.35. *The statements from Remark 3.34 can be made more precise by saying that $h_{P,Q}$ vanishes to order one at P and Q (if $P = Q$, the order of vanishing is two), it has poles of order one at $P+Q$ and ∞ (again if $P+Q = \infty$, this pole has order two), and these are the only zeros and poles of the function.*

Formally, one denotes

$$div(h_{P,Q}) = (P) + (Q) - (P+Q) - (\infty).$$

This is a formal sum of points *on the elliptic curve. It is called the* divisor associated to $h_{P,Q}$*, where the zeros of the function appear with a positive sign, $+$, and the poles with a negative sign, $-$.*

We refer to [30, Theorem XI.8.1(a)] for a proof of this statement, and to [30, Section II.3] for the definition of divisors.

We are now ready to describe Miller's algorithm.

Algorithm 5 Miller's algorithm (optionally evaluation at a point)

Input: $P \in E(K)[N]$, (optional: $R \in E(K)$)
Output: f_P, $(f_P(R))$

1: $(e_0 e_1 \dots e_n)_2 \leftarrow N$, ▷ $N = e_0 + e_1 \cdot 2 + \cdots + e_n \cdot 2^n$, $e_n = 1$
2: $T \leftarrow P$
3: $f \leftarrow 1$
4: **for** $i = n - 1$ **down to** 0 **do**
5: $\quad f \leftarrow f^2 \cdot h_{T,T}$, $(f^2 \cdot h_{T,T}(R))$
6: $\quad T \leftarrow [2]T$
7: $\quad$ **if** $e_i = 1$ **then**
8: $\quad\quad f \leftarrow f \cdot h_{T,P}$, $(f \cdot h_{T,P}(R))$
9: $\quad\quad T \leftarrow T + P$
10: $\quad$ **end if**
11: **end for**
12: **return** f

For the first look at Algorithm 5 , we ignore the optional parameter $R \in E(K)$ (i.e. ignore the orange text in parentheses everywhere). Then the output of Miller's algorithm is a function f_P with divisor

$$div(f_P) = N \cdot (P) - N \cdot (\infty).$$

This means f_P vanishes to order N at P, and has a pole of order N at infinity. Again, it is best to show this using the theory of divisors. The key ingredient is the description of $h_{P,Q}$ from Remark 3.35. A proof can be found in [30, Theorem XI.8.1(b)].

We note that the outline of the algorithm is similar to the double-and-add algorithm for efficient scalar multiplication on elliptic curves (see Exercise 3.12). In particular, the function f_P is computed in $n \approx \log_2(N)$ steps.

In applications, we usually don't require to know the function f_P explicitly, instead, we are only interested in the evaluation of the function at some specific points. It is faster to perform this evaluation at each step of Miller's algorithm, instead of computing the function, and evaluating it in the end. This variant of the algorithm requires modifications in Lines 5 and 8. The modifications are provided in the text in orange in Algorithm 5.

Finally, we can describe an algorithm to compute the Weil pairing which requires four calls to Miller's algorithm (it can be implemented more efficiently in practice).

Algorithm 6 Computing the Weil pairing with Miller's algorithm

Input: $P, Q \in E(K)[N]$
Output: $e_N(P, Q) \in \boldsymbol{\mu_N}$

1: $S \xleftarrow{\$} E(K) \setminus E[N]$ ▷ If there is no such point, it is required to extend the base field.
2: $a \leftarrow f_P(Q + S)$ ▷ Miller's algorithm
3: $b \leftarrow f_P(S)$ ▷ Miller's algorithm
4: $c \leftarrow f_Q(P - S)$ ▷ Miller's algorithm
5: $d \leftarrow f_Q(-S)$ ▷ Miller's algorithm
6: **return** $\frac{a}{b} \cdot \frac{d}{c}$

As mentioned before, it is beyond the scope of this course, to show that Algorithm 6 really computes a pairing. It is not even easy to see that given two N-torsion points P, Q, the output is an N-th root of unity. A proof is given in [30, Section XI.8; Exercise 3.16]. Finally, we remark that the formula underlying Algorithm 6 can be used to provide a hands-on definition for the Weil pairing. This idea is followed in [12, Section 5.8.3].

Embedding degree

So far, we have established that there is a pairing $E[N] \times E[N] \to \boldsymbol{\mu_N}$ that can be computed *efficiently*, more precisely in $O(\log_2(N))$ steps. An important question that we have not discussed so far concerns the embedding of the N-th roots of unity into $\bar{\mathbb{F}}_q$. *What is the smallest field $\mathbb{F}_{q^d}$ containing $\boldsymbol{\mu_N}$?*

Recall that $\mathbb{F}_{q^d}^*$ is a cyclic group of order $q^d - 1$, hence it contains an element of order N if and only if $N \mid q^d - 1$.

Definition 3.36

Let N be a positive integer, and $\mathbb{F}_q$ a finite field. The smallest value d with $N \mid q^d - 1$ is called the *embedding degree* of N in $\mathbb{F}_q$.

If $\gcd(N, q-1) = 1$, then one can even show that all N-torsion points are defined over $\mathbb{F}_{q^d}$, where d is the embedding degree of N. In other words, in this case we have $E[N] \subset E(\mathbb{F}_{q^d})$ (see [30, Lemma XI.6.2]).

When evaluating the Weil pairing, we need to compute over $\mathbb{F}_{q^d}$. It is necessary to take this into account when talking about the efficiency of the evaluation of the Weil pairing.

Example 3.37. *We look at the embedding degrees of the two examples from the beginning of the subsection.*

(a) Example 3.22: Here $E(\mathbb{F}_{13}) = 19$, and the smallest integer d with $19 \mid 13^d - 1$ is $d = 18$.

(b) Example 3.23: Here $E(\mathbb{F}_p) = N$ is a prime of size $N \approx 2^{256}$. And one can check[9] that for the explicit parameters the embedding degree is $d = 1929868153955269923726183083478131797547292737984581739710086052358636024905 6$, in particular $d \approx 2^{253}$.

Exercise 3.38. *Let $E : y^2 = x^3 + 1$ over $\mathbb{F}_p$ for some $p \equiv 2 \pmod 3$. Consider some point $P \in E(\mathbb{F}_p)$ of order $ord(P) = N$. Determine the embedding degree of N in $\mathbb{F}_p$.*

Hint: Use Lemma 3.17.

3.2.3 The MOV Attack

In 1991, Alfred Menezes, Scott Vanstone and Tatsuaki Okamoto showed that the Weil pairing can be used to translate the ECDLP on E to a DLP problem in $\mathbb{F}_{q^d}$, where d is the embedding degree of N in $\mathbb{F}_q$. This attack on the ECDLP is known as the *MOV attack* (Algorithm 7).

Algorithm 7 The Menezes-Okamoto-Vanstone (MOV) algorithm

Input: $P \in E(\mathbb{F}_q)$ with $ord(P) = N$, $Q \in \langle P \rangle$, and $\gcd(q-1, N) = 1$.
Output: $g, A \in \mathbb{F}_{q^d}^*$ with $\mathsf{dlog}_g(A) = \mathsf{dlog}_P(Q)$ and $ord(g) = N$.

```
d ← embedding degree of N in F_q
Determine the group structure E(F_{q^d}) ≅ Z/N_1Z × Z/N_2Z        ▷ Proposition 3.18
while True do
    T ←$ E(F_{q^d})
    T ← [N_1/N] · T
    g ← e_N(P, T) ∈ μ_N ⊂ F*_{q^d}                                ▷ Algorithm 6
    if ord(g) = N then
        break
    end if
end while
A ← e_N(Q, T) ∈ μ_N ⊂ F*_{q^d}                                    ▷ Algorithm 6
return (g, A)
```

[9] See for example `https://neuromancer.sk/std/secg/secp256k1`

In the following, we prove correctness of the algorithm. The concrete running time depends on the embedding degree d (see Remark 3.41).

Theorem 3.39

Let $E/\mathbb{F}_q$ be an elliptic curve, N with $\gcd(q-1, N) = 1$, $P \in E(\mathbb{F}_q)[N]$ and $Q \in \langle P \rangle$, and let d be the embedding degree of N in $\mathbb{F}_q$. Then Algorithm 7 reduces the ECDLP for P and Q to a DLP in $\mathbb{F}_{q^d}^*$.

Proof. We first make an observation similar to Corollary 3.27: For any $T \in E[N]$ it holds that

$$e_N(Q, T) = e_N(P, T)^{\mathsf{dlog}_P(Q)}.$$

This follows from the bilinearity of the Weil pairing. Setting $g = e_N(P, T)$ and $A = e_N(Q, T)$, this shows that

$$\mathsf{dlog}_P(Q) = \mathsf{dlog}_g(A) \pmod{ord(g)} \tag{3}$$

However note that this could be an empty statement, if for instance $g = e_N(P, T) = 1$. Therefore we require that T is a point for which $e_N(P, T)$ is a primitive N-th root of unity in order to obtain a unique solution modulo N.

The *while-loop* (Lines 3 to 10) ensures that we choose a point $T \in E[N]$ so that $e_N(P, T)$ is a primitive N-th root of unity. First a random point $T \in E(\mathbb{F}_{q^d})$ is chosen which is then multiplied by $[N_1/N]$ so that the resulting point is an N-torsion point. The probability that $g = e_N(P, T)$ is a primitive N-th root for a random point $T \in E[N]$ is quite high (especially if N is prime), see Exercise 3.40. So we will find a valid point T after a few iterations.

The correctness of the output now follows from Equation 3. □

In the following exercise, we analyze the "failure probability" in the while-loop of Algorithm 7. Essentially, it follows from the results of the exercise that with high probability only one iteration is required if N is prime.

Exercise 3.40. *Let $P \in E[N]$ a point of order N on an elliptic curve over a finite field $\mathbb{F}_q$, where $q = p^k$ and $p \nmid N$.*

(a) First assume that N is prime. Show that

$$\#\{T \in E[N] \mid ord(g) = N \text{ where } g = e_N(P, T)\} = N(N-1),$$

and conclude that the probability that $e_N(P, T)$ is a primitive root of unity for a point $T \in E[N]$ chosen uniformly at random is $(N-1)/N$.

(b) Now let $N \in \mathbb{Z}$ be an arbitrary positive integer. Compute the proportion of points $T \in E[N]$ so that $e_N(P, T)$ is a primitive N-th root of unity.

Remark 3.41. *The efficiency of the MOV attack crucially depends on the embedding degree of N in $\mathbb{F}_q$. On the bright side (for elliptic curve cryptography), for a randomly chosen elliptic curve, the embedding degree is so high that it is harder to solve DLP in $\mathbb{F}_{q^d}$ than to directly solve the original ECDLP (the distribution of embedding degrees has been analyzed by Ramachandran Balasubramanian and Neal Koblitz [1]).*

However, there exist classes of elliptic curves that have a very low embedding degree. In particular so-called supersingular elliptic curves *which always have embedding degree $d = 2$ (for any $N \mid \#E(\mathbb{F}_p)$). In this case, the MOV algorithm reduces the ECDLP on $E/\mathbb{F}_p$ to a DLP in $\mathbb{F}_{p^2}$. The latter problem can be solved in subexponential time using index calculus (Subsection 2.4). Therefore such curves need to be avoided in the elliptic curve Diffie-Hellman protocol. Curiously, they play an important role in other cryptographic constructions, such as pairing-based cryptography and isogeny-based cryptography (Section 4).*

For a secure implementation one chooses elliptic curves with high embedding degree (see for instance the curve `Secp256k1` in Example 3.37). To illustrate the MOV attack, we choose an example below, where the embedding degree is small.

Example 3.42. *Let $E : y^2 = x^3 + x$ over $\mathbb{F}_{67}$. One can check that $E(\mathbb{F}_p) = 68 = 4 \cdot 17$. Let $P = (64, 29)$, this is a point of order 17. Say we are given the* ECDLP *challenge $Q = (6, 50)$, i.e. we want to determine $n_a \in \mathbb{Z}$ with $[n_a]P = Q$.*

To apply the MOV attack, we first observe that the embedding degree is $d = 2$, since $67^2 - 1 = 4488 \equiv 0 \pmod{17}$. So we need to consider E defined over $\mathbb{F}_{67^2} = \mathbb{F}_{67}(i)$ with $i^2 = -1$.

We compute $n = E(\mathbb{F}_{67^2}) = 2^4 17^2$, and it turns out that the group structure is given by $E(\mathbb{F}_{67^2}) \cong \mathbb{Z}/(4 \cdot 17)\mathbb{Z} \times \mathbb{Z}/(4 \cdot 17)\mathbb{Z}$. In order to sample a random point of order 17, we first sample a random point:

- $T \leftarrow (6i + 9, 26i + 24) \in E(\mathbb{F}_{67^2})$,

and then multiply it by the cofactor $N_1/N = 4$:

- $T \leftarrow 4 \cdot T = (11i + 37, 22i + 42)$.

Now, we compute the Weil pairing $g = e_{17}(P, T) = 39i + 50 \in \mathbb{F}_{67^2}$, which is a primitive 17th root of unity.

Finally, we compute $A = e_{17}(Q, T) = 46i + 30$, and solve the DLP *instance in $\mathbb{F}_{67^2}$:*

$$n_a = \mathsf{dlog}_g(A) = \mathsf{dlog}_{39i+42}(46i + 30) = 14.$$

And one can check that indeed $Q = [14]P$.

3.2.4 Weak parameters for the ECDLP

Generic attacks, like Pollard's rho method are asymptotically the best known attacks to solve the ECDLP on an arbitrary elliptic curve. However, several better algorithms are known that only apply to specific parameters. An important example is the MOV attack that we described in Subsection 3.2.3. Here, we provide a non-exhaustive list of choices that should be *avoided* (or at least require a larger group order), and provide a reference to the known attacks that apply in these cases. For this purpose, denote $P \in E(\mathbb{F}_q)$ for some elliptic curve E, some prime power $q = p^k$, and $ord(P) = N$.

1. N is smooth: In this case, the Pohlig-Hellman algorithm can be applied. The difficulty of solving ECDLP only depends on the largest prime factor of N (see Subsection 3.2.1).
2. $q^d \equiv 1 \pmod{N}$ for some small d: The minimal integer d with $N \mid q^d - 1$ is called the embedding degree of N. The ECDLP for $E/\mathbb{F}_q$ can be reduced to the DLP in $\mathbb{F}_{q^d}^*$. Recall that the latter can be solved in subexponential time in the size of the field $\mathbb{F}_{q^d}$ (Subsection 2.4). If d is small, then this provides us with a subexponential algorithm to solve ECDLP (see Subsection 3.2.3).
3. $N = p$: An elliptic curve with $E(\mathbb{F}_p) = p$ is called anomalous. In this case, the ECDLP can be reduced to computing the Group-DLP in the additive (!) group $(\mathbb{F}_p, +)$ which can be solved in polynomial time. This attack was presented in concurrent articles by Takakazu Satoh and Kiyomichi Araki [25], Igor Semaev [26] and Nigel Smart [31].
   Interestingly, the attack relies on some abstract concepts in the theory of elliptic curves: One needs to consider a lift of the elliptic curve to the p-adics $\mathbb{Q}_p$, and perform computations in the formal group. The mathematical prerequisites for understanding this attack can be found in [30, Chapter VII].
4. $q = p^k$ for $k > 2$: There are several attack strategies to solve the ECDLP faster than with a generic algorithm when the finite field $\mathbb{F}_q$ is not a prime field. The first one, was introduced by Gerhard Frey [7] who had the idea to use *Weil descent* in order translate the discrete logarithm problem for E over $\mathbb{F}_q$ to a Group-DLP on a higher dimensional *abelian variety* over $\mathbb{F}_q$.
   In a similar gist, Pierrick Gaudry developed a variant of the index-calculus attack with factor base the set of points $\{P = (x_i, y_i) \mid x_i \in \mathbb{F}_p\}$, [10]. For fixed k, the complexity of the resulting algorithm is $\tilde{O}(p^{2-2/k})$ which is better than Pollard's rho approach when $k > 2$. There are many interesting follow-up works of these two papers.

We would like to highlight that the mathematics underlying the attacks of the ECDLP problem are significantly more advanced than the mathematical prerequisites necessary to construct the elliptic-curve Diffie-Hellman scheme. It is fascinating that some fairly abstract concepts from algebraic geometry find such concrete and practical applications in the analysis of modern cryptographic protocols.

3.2.5 Invalid curve attack

The *invalid curve attack* is a type of attack on the elliptic-curve Diffie-Hellman protocol that cannot be avoided by a good choice of parameters. Instead, it requires either a (small) change in the protocol, or a more clever implementation. This type of attack was first proposed by Ingrid Biehl, Bernd Meyer and Volker Müller in the year 2000 [3].

Assume that

$$E : y^2 = x^3 + ax + b \quad \text{over } \mathbb{F}_q$$

and $P \in \mathbb{F}_q$ with $ord(P) = N$ are well chosen parameters for an ECDH key exchange. Further let

$$E' : y^2 = x^3 + ax + b' \quad \text{over } \mathbb{F}_q$$

be another elliptic curve which has a point $P' = (x', y')$ of order m with $1 < m \ll N$. Consider the scenario sketched in Figure 12.

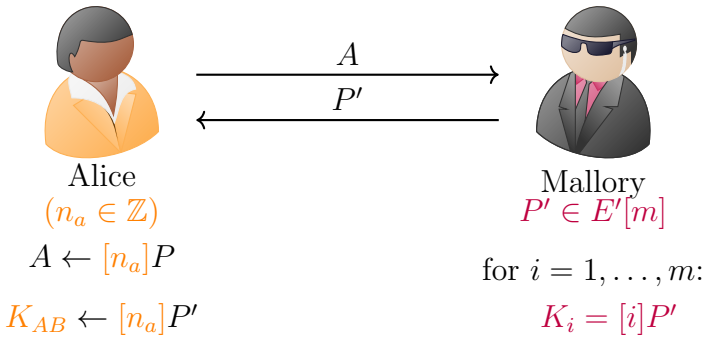


Figure 12: Invalid curve attack with public parameters $P \in E(\mathbb{F}_q)$

Alice thinks she is communicating with Bob, and performs all computations as usual. On the other side, there is a malicious member (Mallory) who instead of sending an honestly generated public key, sends the point P' to Alice. Alice does not notice that $P' \notin E(\mathbb{F}_q)$. Note that the addition and doubling formulas are the same on the curves E and E'. They only depend on the coefficient a, and not on the constant term b (cf. Theorem 3.7). This means that Alice computes the value

$$K_{AB} = [n_a]P' \in \langle P' \rangle \subset E'[m].$$

Since the order $ord(P') = m$ is small by assumption, Mallory can compute all possible shared keys $K_1 = [1]P', \ldots, K_m = [m]P'$ and simply test which one is correct (for example when Alice uses the key to encrypt a message). This reveals Alice's secret key modulo m. Repeating the attack modulo enough small values m_i, Mallory can recover the entire secret key $n_a \pmod N$.

Exercise 3.43. SAGE *In this exercise, you are Mallory and your goal is to recover Alice's key using the invalid curve attack.*[10] *You are using the elliptic curve*

$$E : y^2 = x^3 + x + 25, \quad \textit{over } \mathbb{F}_p$$

with parameters $p = 18446744073709551629$ *and* $P = (100, 6701093164194334038) \in E(\mathbb{F}_p)$ *with prime order* $ord(P) = 18446744070571455341$.

Alice's public key is $A = (10301126922579099648, 17157096027455143833)$

(a) You (Mallory) send the point `P1 = (18165116349323561130, 6150811377566577555)`. *Check that* $P_1 \notin E(\mathbb{F}_p)$. *Find the (unique) parameter* b_1 *so that* P_1 *is on the curve* $E_1 : y^2 = x^3 + x + b_1$. *What is the order of* P_1*? Compute all possible values for* $[a]P_1$ *(without computing the secret key* a*).*

(b) By checking all possible values for $[a]P_1$*, you find that Alice computed the "shared" session key* `KAB1 = (18446744073709551626, 5458368549901343073)`. *What can you conclude about the value of* a*?*

[10] The parameters chosen in the example are still smaller than for real-world applications (cf. Example 3.23), but sage will already struggle to compute the discrete logarithm from scratch.

(c) *You continue sending more maliciously generated public keys. The points P_i sent by you and the value of Alice's corresponding session keys $K_{AB,i}$ are collected in the table below. What is Alice's secret key?*

Point P_i	*shared key* $K_{AB,i}$
$(18165116349323561130, 6150811377566577555)$	$(18446744073709551626, 5458368549901343073)$
$(16395352116619970353, 6034018393034262788)$	$(16718172481871216672, 1183835131033830123)$
$(12524092530016578390, 5123067425181934705)$	$(14160835454605074121, 3060569204740460707)$
$(4516937973540258973, 7005509288484349242)$	$(8040943336447228867, 13169014645599232942)$
$(15975665384073761733, 11032318707512935771)$	$(13311443695356568982, 15843145926225201761)$
$(7142461303424024564, 6616795770963544980)$	$(15087812134455913873, 4833814421951071352)$
$(15087812134455913873, 4833814421951071352)$	$(5030474179534288684, 4948558071821812509)$
$(9450845281388796607, 5731912410853213485)$	$(7134676168471120217, 6089059990139022202)$
$(5131031356309480317, 13835549974890579026)$	$(13275501062262900275, 10650377260320625285)$

Clearly, for a secure key exchange, it is necessary to thwart such an attack. We describe two potential counter measures:

- **Public key validation**: An easy way to avoid these kind of attacks is to check if the public key is a valid point on the curve. Explicitly, in the scenario in Figure 12, upon receiving the point $P' = (x', y')$ from Mallory, Alice would check whether P' is on the curve E or not. In the sketched scenario, she would find that

$$y'^2 = x'^3 + ax' + b' \neq x'^3 + ax' + b,$$

  since $b \neq b'$, and abort the key exchange. At the expense of a few additional computations, this completely prevents the invalid curve attack.

- **x-only arithmetic** Another solution is that Alice and Bob represent their public key point $B = (x_B, y_B)$ and $A = (x_A, y_A)$ only by their x-coordinates, x_A and x_B respectively. Note that the x-coordinates defines the elliptic curve point up to a sign. This means that using only the x-coordinate, Alice and Bob can compute $K_{AB} = K_{BA} = (x_{AB}, y_{AB})$ up to a sign, in other words, they would use the x-coordinate x_{AB} as the shared key. *How can they perform these computations?*

  - Say Alice receives Bob's public key x_B. She can then compute the possible y-coordinates by solving $\pm y_B = \sqrt{x_B^3 + ax_B + b}$. She has no way of knowing which of y_B and $-y_B$ is the correct solution, but she may just proceed with any of the two solutions to compute $K'_{AB} = (x_{AB}, y'_{AB})$ with the usual formulas. The resulting x_{AB} does not depend on the choice of $\pm y_B$.
    Note that taking square-roots is a relatively expensive computations, and for an efficient implementation, it is not a good idea to use this approach.

  - The x-coordinate x_{AB} can be computed more efficiently. Phrased more abstractly, we just saw that scalar multiplication is well-defined on the level of x-coordinates. One can show that there exist explicit formulas that given the x-coordinate $x(P)$ of a point P and an integer m, output the x-coordinate $x([m]P)$ (see also Example 3.44 and Exercise 3.46). This concept is known under the name *x-only arithmetic*. Indeed, it turns out that in practice using x-only arithmetic is often faster than using the "normal" addition formulas. Note that the latter is independent of the invalid curve attack and thwarting this attack is just an additional benefit of using x-only arithmetic.

Even though this attack was already discovered in the year 2000, there were still some more recent implementations that did not use any of the above mentioned countermeasures which led to serious security vulnerabilities in real-world applications, see for example [13].

Remark 3.44. *Let $E : y^2 = x^3 + ax + b$ be an elliptic curve over some field K, and let $P \in E(K) \setminus E[2]$. Then*

$$x([2]P) = \frac{(3x(P)^2 + a)^2}{4(x(P)^3 + ax(P) + b)} - 2x(P).$$

As before, $x(P)$ denotes the x-coordinate of P, and $x([2]P)$ is the x-coordinate of the point $[2]P$. Similarly, we also denote $y(P)$ for the y-coordinate.

The formula is deduced from Theorem 3.7 (case (b)). Note that we replaced $y(P)^2$ in the denominator of m^2 by $x(P)^3 + ax(P) + b$.

Remark 3.45. *While scalar multiplication is well-defined on the level of x-coordinates, this is* not *true for point addition. For instance assume you are given $x_P = x(P)$, $x_Q = x(Q)$ for some points $P, Q \in E(\mathbb{F}_q) \setminus E[2]$ with $P \neq Q$. There are two possible points with each given x-coordinate. Denote*

$$P^+ = (x_P, +y_P), \quad P^- = (x_P, -y_P), \quad Q^+ = (x_Q, +y_Q), \quad Q^- = (x_Q, -y_Q).$$

Using the formulas from Theorem 3.7, we observe that

$$x(P^+ + Q^+) = x(P^- + Q^-) \neq x(P^+ + Q^-) = x(P^- + Q^+).$$

There is no way of knowing which of the results is correct.

Exercise 3.46. *Let $E : y^2 = x^3 + ax + b$ be an elliptic curve. Find a formula that computes $x([3]P)$ from $x(P)$ and the curve constants a, b.*

4 Isogeny-based cryptography

Isogeny-based cryptography is a relatively new and very active research topic in modern cryptography.

References

[1] Ramachandran Balasubramanian and Neal Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the menezes—okamoto—vanstone algorithm. *Journal of cryptology*, 11(2):141–145, 1998.

[2] Daniel J. Bernstein and Tanja Lange. Safecurves: choosing safe curves for elliptic-curve cryptography. URL: `https://safecurves.cr.yp.to`.

[3] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Annual international cryptology conference*, pages 131–146. Springer, 2000.

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[5] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[6] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[7] Gerhard Frey. How to disguise an elliptic curve. Talk at ECC 1998, 1998.

[8] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Annual International Cryptology Conference*, pages 33–62. Springer, 2018.

[9] Steven D Galbraith. Mathematics of public key cryptography. version 2.0. 2018.

[10] Pierrick Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *Journal of Symbolic computation*, 44(12):1690–1702, 2009.

[11] Helmut Hasse. Zur theorie der abstrakten elliptischen funktionenkörper. 1936.

[12] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. An introduction to cryptography. In *An Introduction to Mathematical Cryptography*, pages 1–59. Springer, 2014.

[13] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. Practical invalid curve attacks on tls-ecdh. In *European Symposium on research in computer security*, pages 407–425. Springer, 2015.

[14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[15] Auguste Kerckhoff. La cryptographie militaire. IX:5–38, 1883.

[16] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[17] Maurice Kraitchik. *Théorie des nombres: Théorie des nombres*, volume 1. Gauthier-Villars, 1922.

[18] Ueli M Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In *Annual international cryptology conference*, pages 271–281. Springer, 1994.

[19] Alexander May and Carl Richard Theodor Schneider. Dlog is practically as hard (or easy) as dh–solving dlogs via dh oracles on ec standards. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):146–166, 2023.

[20] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, 1991.

[21] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

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

[23] John M Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of cryptology*, 13(4):437–447, 2000.

[24] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[25] Takakazu Satoh and Kiyomichi Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Mathematici Universitatis Sancti Pauli*, 47(1):81–92, 1998.

[26] Igor Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of computation*, 67(221):353–356, 1998.

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

[28] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[29] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–266. Springer, 1997.

[30] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[31] Nigel P Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of cryptology*, 12(3):193–196, 1999.

[32] Andrew Sutherland. Lecture notes on elliptic curves. `https://math.mit.edu/classes/18.783/2023/`, 2023.

[33] Edlyn Teske. Speeding up Pollard's rho method for computing discrete logarithms. In *International Algorithmic Number Theory Symposium*, pages 541–554. Springer, 1998.

[34] Edlyn Teske. Square-root algorithms for the discrete logarithm problem. In *Public-Key Cryptography and Computational Number Theory*, pages 283–301, 2001.

[35] Lawrence C Washington. *Elliptic curves: number theory and cryptography*. Chapman and Hall/CRC, 2008.

[36] André Weil. Sur les fonctions algébrique à corps de constantes fini. *CR Acad. Sci. Paris*, 210(592-594):149, 1940.