PART II. CRYPTOGRAPHY

Contents

Some useful references

[1] I. Blake, G. Seroussi and N. Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Note Series **265**, Cambridge University Press (1999)

[2] H. Cohen and G. Frey (eds.), *Handbook of elliptic and hyperelliptic curve cryptography*, Discrete Mathematics and its Applications **34**, Chapman & Hall/CRC (2005)

[3] N. Koblitz, *Algebraic aspects of cryptography*, Algorithms and Computation in Mathematics **3**, Springer-Verlag (1997)

[4] P. Nguyen, *Public-key cryptanalysis*, a chapter in I. Luengo, *Recent trends in cryptography*, Contemporary Mathematics series **477**, AMS-RSME (2009)

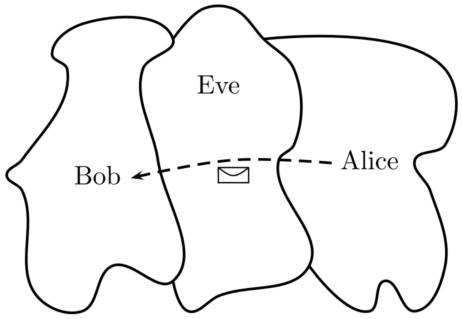
[5] J. Silverman, *The arithmetic of elliptic curves*, Graduate Texts in Mathematics **106**, Springer-Verlag (1986)

The most down-to-earth text is [3]. The standard reference on elliptic curves is [5], for which some experience with algebraic geometry is recommended (the first two chapters give a good introduction). The most up-to-date account is the extensive [2], of which a second edition will appear in the near future. Reference [4] is mainly about RSA instead of elliptic curve cryptography.

1. Introduction to cryptography

1.1. **A medieval tale.** We are writing 914 AD. The mighty king John the Great has come to pass away. The empire is subdivided into three regions, according to the number of heirs. John's oldest daughter Alice becomes the beloved queen of the Eastern Part. His son Bob becomes the benign king of the Western Part. John's youngest, bewitched daughter Eve crowns herself to become the first emperess of the Middle Part.

But greedy Eve wants more, and soon after her father's death she begins preparing a cowardly attack on the Western Part. However, her plans leak out, and Alice gets informed about them. She decides to warn her brother and commands her best messenger to bring a hand-written message across Eve's cursed empire. This is risky business: suppose revengeful Eve intercepts the message, then she will probably change plans and attack Alice's Eastern Part instead.



Now instead of fully relying on the agility of her messenger, Alice could take some security measures herself. She could for instance put the document in an unbreakable box equipped with an unforceable lock. Then the enemy intercepting the message would not be able to access it. Of course, this requires that Bob has a key to unlock the box. Therefore we call such an approach *symmetric*: Alice and Bob should possess a copy of the same key. This only seems to replace the problem by another one: how can both siblings obtain a copy of the same key? Using a messenger? Carrying a box that is locked using another key? Which should in its turn be sent by a messenger? One apparently runs into a vicious circle.

1.1.1. *The three-pass protocol.* Here is a solution to Alice's problem. She locks the box using one of her own keys, and sends it to Bob. Bob properly receives it, but is unable to open it. Instead, he attaches an additional lock, and sends the box back to Alice. Now Alice removes her own lock, and commands her exhausted messenger to take the box to Bob once more. The latter is now able to open the box and read the secret message. At no point in the process, the enemy will be able to access the message.

1.1.2. *A public key protocol.* Another approach for Alice is that Bob possesses one of those fancy padlocks, which can be locked by just 'clicking', without using a key. Then Alice sends out her messenger to go get Bob's open padlock. She then attaches it to her unbreakable box, and locks it. Now Bob is the only person that is able to open the box, and it is safe to let the messenger take it across the enemy's territory. So a way to think about the fancy padlock is as if it were a publicly available key, that is free-to-use to 'lock' messages.

1.2. **Modern setting.** Enough about kings, queens and evil empires. From now on, Alice and Bob are computer users, and Eve is a bad girl eavesdropping the internet or any other unsecured channel connecting their computers.

1.2.1. *Alice's message.* In practice, Alice's message M is a sequence of `0`'s and `1`'s. One usually limits the length of the sequence to some fixed, publicly known number r, i.e. $M \in \{\mathtt{0}, \mathtt{1}\}^r$. If the message does not meet this bound, it is split into blocks. The set $\mathcal{M} = \{\mathtt{0}, \mathtt{1}\}^r$ is called the *message space* or the *plaintext space.*

1.2.2. *Bob's secret key.* Bob possesses a secret key k_B, which allows him to 'unlock' or *decrypt* Alice's message. The key is a sequence of `0`'s and `1`'s of some publicly known length m. Note that the *key space* $\mathcal{K} = \{\mathtt{0}, \mathtt{1}\}^m$ is finite, so in principle evil Eve can exhaustively search for the key that unlocks Alice's message.

Exercise 1.1. Suppose $m = 80$ and Eve's computer can verify up to $4 \cdot 10^9$ candidate-keys per second, then how long is this expected to take her?

Thus in practice, this is not a threat. We will comment more on key lengths in Section 1.5.3.

1.2.3. *Symmetric keys.* In the symmetric setting, Alice uses a copy of Bob's key to 'lock' or *encrypt* her message. In this case, we put $k_A = k_B$. As explained in our medieval tale, obtaining such a copy is a priori non-trivial.

1.2.4. *Public keys.* Miraculously enough, there exist mathematical analogues of the fancy padlock described above: everyone (including Eve) can encrypt messages, but only Bob can decrypt them. This is based on well-chosen *one-way functions*

$$f : \mathcal{K} \dashrightarrow \mathcal{K},$$

which are publicly known functions that are easy to compute, but where computing an inverse image of a given binary string is (believed to be) very hard. The dashed arrow indicates that f doesn't need to be defined everywhere. The role of the fancy padlock is then played by $k_A = f(k_B)$, which can be made public without revealing k_B. The most famous instance of a public key scheme is RSA. There, the underlying one-way function is the map sending two prime numbers of at most $\lfloor m/2 \rfloor$ binary digits to their product. See Section 1.4 for a description of RSA.

1.2.5. *Encryption and decryption.* The role of the unbreakable box is played by a publicly known pair of efficiently computable functions requiring double input. The *encryption function*

$$E : \mathcal{M} \times \mathcal{K} \dashrightarrow \mathcal{C}$$

takes as input Alice's message M, along with the key k_A to 'lock' it. It takes values in the *ciphertext space* $\mathcal{C} = \{\mathtt{0}, \mathtt{1}\}^s$. The *decryption function*

$$D : \mathcal{C} \times \mathcal{K} \dashrightarrow \mathcal{M}$$

takes as input an encrypted message, along with Bob's key k_B to 'unlock' it:

$$D(E(M, k_A), k_B) = M.$$

Bob's key k_B is the big secret ingredient: if Eve gets to know it, she can apply $D(\cdot, \cdot)$ and recover the message M. 'Unbreakable' means that for an adversary that doesn't know k_B, it should be extremely hard to recover M from $E(M, k_A)$. The situation cryptographers strive for is where Eve's only approach is to exhaustively try all candidate-keys $k_B \in \mathcal{K}$; see Exercise 1.1 above.

1.3. **Symmetric examples.**

1.3.1. *Substitution ciphers.* In a sense, Julius Caesar is the founder of modern cryptography. To communicate with his generals, he substituted every letter of his message with the third subsequent letter of the Roman alphabet. Forcing this into the above formalism (though not forcing the non-electronic Roman empire in a binary world), we can write

$$\begin{array}{ccccccl} E: & \mathbb{Z}/(26) & \times & \mathbb{Z}/(26) & \to & \mathbb{Z}/(26): & (m,k_B)\mapsto m+k_B, \\ & \shortparallel & & \shortparallel & & \shortparallel & \\ & \mathcal{M} & & \mathcal{K} & & \mathcal{C} & \end{array}$$

$$\begin{array}{ccccccl} D: & \mathbb{Z}/(26) & \times & \mathbb{Z}/(26) & \to & \mathbb{Z}/(26): & (m,k_B)\mapsto m-k_B, \\ & \shortparallel & & \shortparallel & & \shortparallel & \\ & \mathcal{C} & & \mathcal{K} & & \mathcal{M} & \end{array}$$

where Caesar used $k_B = 3$. This is a very weak method due to the small key space: an adversary trying to decipher some sequence of letters $C \in \mathcal{C}$ can simply try all 26 possibilities for $k_B \in \mathcal{K}$ until the corresponding sequence of letters $D(C, k_B) \in \mathcal{M}$ makes sense[1].

The size of the key space can be dramatically increased by allowing arbitrary permutations of the alphabet:

$$\begin{array}{ccccccl} E: & \mathbb{Z}/(26) & \times & \mathcal{S}_{26} & \to & \mathbb{Z}/(26): & (m,\sigma)\mapsto \sigma(m). \\ & \shortparallel & & \shortparallel & & \shortparallel & \\ & \mathcal{M} & & \mathcal{K} & & \mathcal{C} & \end{array}$$

The set $\mathcal{S}_{26}$ now contains $26! \approx 2^{88}$ elements, which is certainly big enough to exclude exhaustive search as a possible attack, see Exercise 1.1. Nevertheless, these more general substitution ciphers are again easily broken. This can be done using *frequency analysis.* E.g., in a comparatively large English text, about 12.7% of all letters will be 'E', about 9.5% will be 'T', about 8.2% will be 'A', and so on. So by computing frequencies one can make a good guess for the secret permutation $\sigma \in \mathcal{S}_{26}$, and some additional puzzling will reveal Alice's message. This works very well in practice.

1.3.2. *Towards AES/Rijndael.* Throughout history, mankind has produced a long list of cryptographic schemes and attacks against these, many of which have the above statistical flavor (although in a less obvious manner). A famous example is the 1917 decryption of the Zimmermann telegram, which was an attempt of the German minister of foreign affairs to set up a war between Mexico and the USA. The main objective was to keep the USA away from Europe. But as said, the telegram got decrypted, causing the opposite effect: the USA declared the war against the German alliance, which brought World War I in a decisive stadium.

[1]Of course, this conclusion is not very honest. At the time of usage, the adversary was not familiar with the phenomenon of encryption, let alone that the functions E and D were publicly known. So before coming to the point of simply trying all 26 keys, he should have come to the harder point of understanding the mechanism of Caesar's cipher. From this perspective, the method definitely becomes somewhat safer. However, we put emphasis on 'somewhat'. In practice, it almost never turns out to be a good idea to treat E and D as a part of the secret. See Section 1.5.1.

WESTERN UNION TELEGRAM

via Galveston

GERMAN LEGATION

MEXICO CITY

[illegible]

Charge German Embassy

the Zimmermann telegram

Another threat for symmetric cryptosystems is *differential cryptanalysis*: if two closely related messages are encrypted using the same key, then linear properties of the cryptosystem (which should not be there!) leak information. This played an important role in the cracking of the Enigma and Lorenz ciphers in World War II. At some point in 1941, due to a mistake, a German clerk was forced to send a certain message twice. But he got lazy and started using abbreviations. By analyzing the difference of the two closely related encrypted messages, the allied were able to break the code.

Cryptography became serious academic business as of 1976. One reason is the discovery of public key cryptography by Diffie and Hellman, see Sections 1.4 and 2 below. Another reason is the introduction of a standardized symmetric cryptosystem called DES (Data Encryption Standard) by the US Government. The academic world was sceptical, because DES uses keys of 56 bits only (see Section 1.5.3 for some comments on key lengths). People feared that the government had foreseen some back door in the system. As a consequence, the system was subject to a thorough analysis, and ironically enough DES has meant a lot for the development of modern cryptanalysis. Nowadays, it is fully broken: in 1999, a DES key was recovered in less than a day.

In 2002, the lessons from DES and other (semi-)failures culminated in a new encryption standard, that was introduced through a 5-year competition: Rijndael or AES (Advanced Encryption Standard). This is now used worldwide, and considered very secure. For instance, the US Government itself uses it for encoding top secret information – see Section 4.4.

1.4. **A public key example: RSA.**

The principles of public key cryptography were discovered[2] by Diffie and Hellman in 1976. They proposed a scheme for key exchange, that will be described in Section 2. In 1978, Rivest, Shamir and Adleman published the first practical public key encryption method: RSA.

1.4.1. *A quick sketch.*

In Section 1.2.4, we already mentioned that the function sending two prime numbers to their product is considered one-way: factoring a product of two large prime numbers is believed to be very hard. It can be used to construct a public key scheme as follows. (We won't put effort in forcing our description in the formalism of Section 1.2.)

[2]Or better: rediscovered. In 1997, it was made public that Diffie-Hellman key exchange was already discovered in the early 1970's by members of a British intelligence agency.

Bob chooses a pair of prime numbers $k_B = (p, q)$ and computes their product n. He also chooses an *encryption exponent* e, which is an integer coprime to $(p-1)(q-1)$. The pair $k_A = (n, e)$ is the *public key* of the cryptosystem. Alice can freely use it to encode a message M as follows. Read M as an integer, and suppose it is $< n$ (otherwise the message is split into blocks as in Section 1.2.1). Then the encrypted message is

$$C = M^e \bmod n.$$

To decrypt, Bob first computes a multiplicative inverse d of e in the ring $\mathbb{Z}/((p-1)(q-1))$ using Euclid's algorithm. He then recovers the message as

$$C^d \bmod n = M^{ed} \bmod n = M,$$

where the latter equality holds by Euler's congruence (which might fail if M is not coprime to n, but in practice this never happens).

Clearly, if Eve knows about an efficient factoring algorithm, she can reveal Bob's secret key and decrypt Alice's messages. So the security of RSA highly depends on the hardness of factoring.

Big open question. Is factoring integers in fact *equivalent* to cracking RSA? I.e. would an attack on RSA result in an efficient algorithm for factoring integers?

For a long time, people believed that the answer would be 'yes'. However, nowadays the cryptographic community doesn't seem so convinced any more (Boneh-Venkatesan, 1998).

1.4.2. *An elliptic curve factorization method.* This course seems a good place for mentioning a surprising application of elliptic curves: an algorithm for factoring n into p and q that works considerably better than the naive methods (although not good enough to crack RSA). It is an ingenious extension of Pollard's $p-1$ method:

Exercise 1.2 (Pollard's $p-1$ method). Suppose that $p-1$ is a divisor of $\prod_{r=2}^{\beta} r$, for some small bound β. Explain why the following routine is likely to factor $n = pq$.

```
(1) Take a random $a \in \{2, \ldots, n-1\}$.
(2) For $r \in \{2, \ldots, \beta\}$ do
    (a) $a \leftarrow a^r$,
    (b) compute $\gcd(a, n)$, if a nontrivial factor is found then stop.
(3) If no nontrivial factor was found, then go back to (1).
```

Of course the routine also works if $q-1$ is a divisor of $\prod_{r=2}^{\beta} r$. What if both $p-1$ and $q-1$ are divisors?

The idea of elliptic curve factorization (Lenstra, 1987) is to replace the multiplicative groups $\mathbb{F}_p^\times$, $\mathbb{F}_q^\times$, which always have $p-1$ resp. $q-1$ elements, by randomly chosen elliptic curve groups $E_p(\mathbb{F}_p)$ and $E_q(\mathbb{F}_q)$, whose number of elements take values all over $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ resp. $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$. Whereas the concrete numbers $p-1$ or $q-1$ will rarely split into small prime factors, the above intervals are likely to contain an integer that *does* split into small primes.

One can exploit this as follows.

(1) Choose a small bound $\beta \in \mathbb{Z}_{\geq 0}$ and let $k = \prod_{r=2}^{\beta} r$.
(2) Take random $A, x_0, y_0 \in \mathbb{Z}/(n)$ and consider

$$f(x, y) = y^2 - x^3 - Ax - B \quad \in \mathbb{Z}/(n)[x, y],$$

where $B = y_0^2 - x_0^3 - Ax_0$. Verify that $\gcd(4A^3 + 27B^2, n) = 1$. If not, either output the non-trivial factor and stop, either try new A, x_0, y_0.

(3) Naively using the formulas from Exercise I.1.5, let $\mathcal{P} = (x_0, y_0)$ and iteratively compute $k \cdot (x_0, y_0)$ as in Step (2) of Exercise 1.2. If (at some point) division fails, we have found a denominator d for which $\gcd(n, d) \neq 1$. Then either output the non-trivial factor and stop, either try new A, x_0, y_0.

So the idea behind the algorithm is that at each step, $y^2 - x^3 - Ax - B$ defines two 'random' elliptic curves, one curve E_p over $\mathbb{F}_p$ and one curve E_q over $\mathbb{F}_q$. Likewise, the tuple $\mathcal{P} = (x_0, y_0)$ defines two points: $\mathcal{P}_p \in E_p(\mathbb{F}_p)$ and $\mathcal{P}_q \in E_q(\mathbb{F}_q)$. Now the orders of $E_p(\mathbb{F}_p)$ and $E_q(\mathbb{F}_q)$ are 'random' integers in the intervals

$$[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]\,, \quad [q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}]\,,$$

respectively (see Section 3.1.1 for some comments on this). With some probability, $\#E_p(\mathbb{F}_p)$ or $\#E_q(\mathbb{F}_q)$ will be a divisor of β. Suppose this is the case for $\#E_p(\mathbb{F}_p)$. Then $k\mathcal{P}_p = \mathcal{O}$, which will cause a division failure in Step (3). This is likely to produce a nontrivial factor.

We remark that nowadays there exist other methods (e.g. the number field sieve algorithm) that do better than the elliptic curve method, at least for the size of primes that are typically used in RSA.

1.5. **Some non-mathematical issues.**

1.5.1. *Kerckhoffs' principle.* Already in 1880, Kerckhoffs formulated the principle that the security of a cryptographic system should depend on the secret key only. In other words: the encryption and decryption functions $E(\cdot, \cdot)$ and $D(\cdot, \cdot)$ should be publicly known.

For designers of symmetric cryptosystems, it might be tempting to treat E and/or D as part of the secret: this certainly makes it harder for an adversary to crack the system. However,

(1) a good system does not need this,
(2) using techniques from reverse engineering, it is often possible to recover the encryption and decryption method anyway,
(3) transparency allows a bigger community to verify the security of the system.

History has provided many examples of cryptosystems that were unsuccessfully kept secret. For instance, using reverse engineering, in World War II the British allied were able to emulate a Lorenz machine without ever having seen one!

Despite all this, Kerckhoffs' principle is still being violated. A recent example is KeeLoq, the system used in the majority of remotely controlled car locks. The method was kept secret, but it leaked out in 2006 and one year later a team of Belgian and Israeli cryptographers showed that about an hour of communication with the key suffices to steal the corresponding car. Note that KeeLoq is still being used.

1.5.2. *Man-in-the-middle attacks and trusted third parties.* Let us reconsider the three-pass protocol, see Section 1.1.1. Suppose that Eve intercepts Alice's message. She is unable to access it, since Alice locked the box. But she has a mean plan: she attaches one of her own locks to the box. The (corrupted) messenger takes it back to Alice, who thinks the lock is Bob's. Naive Alice removes her own lock and sends the message back to Bob, uh... Eve. Eve removes her lock and reads the message.

This is called a *man-in-the-middle attack*:

$$\begin{tikzcd}\text{Alice} \arrow[r, leftrightarrow] & \underset{\substack{\text{pretends}\\ \text{she's Bob}}}{\text{Eve}} \arrow[r, dash, dashed] & \text{Bob}\end{tikzcd}$$

As explained, the three-pass protocol is very vulnerable to this attack. Man-in-the-middle attacks apply as well to public key schemes such as RSA: Eve can create her own public key and pretend it is Bob's. Also *phishing*, which is a plague on the internet, is a kind of man-in-the-middle attack.

Problems of this type can be addressed using instances providing authentication certificates. Such instances are sometimes called trusted third parties (TTP's). An authentication certificate is essentially a phrase 'I am Bob' that is *digitally signed* by the TTP. Alice can then verify the signature using the TTP's public key. We will not go into further detail on the concrete set-up of such a *public key infrastructure*. But in Section 3.2, we will describe an elliptic curve based example of a digital signature protocol.

1.5.3. *Security levels.* Consider the following engineery definition.

Definition 1.3. *A cryptosystem is said to provide* n-bit security *if Eve's best approach is equivalent to an exhaustive search in a set of size* 2^n.

We say 'engineery' for two reasons:

(1) The definition is modulo future breakthroughs: we only concern attacks that we currently know about. In practice, it turns out to be very hard to *prove* that a system is n-bit secure.
(2) The actual security of an n-bit secure system also depends on other factors (e.g. the time Eve needs to verify a key).

Nevertheless, it is useful to have some standards.

In Exercise 1.1, we studied 80-bit security. In contrast with the outcome of the exercise, this is becoming an old standard. First, instead of just a bad girl, Eve could be a criminal organization, or an intelligence agency. Then the exhaustive search can be spread among thousands of computers. Second, computers become faster and faster, and for a message that is intended to be kept secret on the long term, the clock speed mentioned in the exercise is an underestimation. USA's National Institute for Standards and Security (NIST) suggests that 80-bit secure systems should no longer be used as of 2010. The new standard has become 128-bit security. For 'highly sensitive data', one even imposes 192-bit or 256-bit security.

AES provides 128-bit security for 128-bit keys and 256-bit security for 256-bit keys, which is of course the optimal situation. 3DES, the update of DES that is used in most cash withdrawal machines, offers 112-bit security for a 168-bit key.

As for RSA, one recommends 3072-bit keys to obtain 128-bit security! This is due to the above-mentioned factorization methods: although they do not suffice to crack RSA, they at least force cryptographers to dramatically increase the key size. Note however that the margin is very safe. a record-breaking RSA key of 663 bits, published by RSA Laboratories as a challenge, was factored in 2005 (it took about five months on a cluster of computers).

2. Key exchange

Symmetric schemes typically allow much faster encryption and decryption than their public key counterparts. Therefore, an attractive option is the following combination: Bob sends his secret key k_B to Alice using a public key scheme. Then both siblings proceed using a symmetric system.

Instead of using a public key scheme, one usually prefers using one of the specialized *key exchange protocols*. These are quite related to key communication through public key encryption, but now the key is 'commonly generated' instead of just sent as a plain message. The most notorious example is the Diffie-Hellman protocol, which was published in 1976, thereby founding public key cryptography. The underlying principle is that for certain groups $G, \cdot$ and $g \in G$, the map

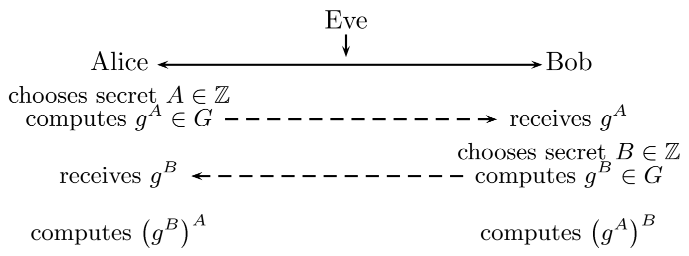
$$\mathbb{Z} \to G : g \mapsto g^A$$

is (believed to be) one-way: it is very hard to compute *logarithms* with base g.

Exercise 2.1. Show that the base g does not matter, in the following sense. Suppose g_1, g_2 generate the same subgroup of G. If computing logarithms with base g_1 is easy, then so is computing logarithms with base g_2.

2.1. **The Diffie-Hellman protocol (DHP).**

The Diffie-Hellman protocol can be formulated for any publicly known pair g, G where $G, \cdot$ is a group and $g \in G$. Here is how Alice and Bob can agree upon an element of G.


Eve
Alice
Bob
chooses secret $A \in \mathbb{Z}$
computes $g^A \in G$
receives g^A
chooses secret $B \in \mathbb{Z}$
receives g^B
computes $g^B \in G$
computes $\left(g^B\right)^A$
computes $\left(g^A\right)^B$

Thus, Alice and Bob have agreed upon $g^{AB} \in G$. Eavesdropping Eve collected the following information: g^A, g^B. If she wants to recover the key of Alice and Bob, she must solve the following problem:

Definition 2.2 (Diffie-Hellman problem). *The* Diffie-Hellman problem *is about finding an efficient method to compute g^{AB} from any triplet g, g^A and g^B.*

Closely related to this is

Definition 2.3 (discrete logarithm problem). *The* discrete logarithm problem *is about finding an efficient method to compute $A \pmod{\operatorname{ord}(g)}$ from any pair g, g^A.*

For well-chosen groups G, the discrete logarithm problem is believed to be very hard. Two popular choices are $G = \mathbb{F}_q^\times, \cdot$ for some large finite field $\mathbb{F}_q$, and $G = E(\mathbb{F}_q), \oplus$ for some elliptic curve E over some large finite field $\mathbb{F}_q$ (see Section 3).

Exercise 2.4. Show that the discrete logarithm problem is easily solved in the groups $\mathbb{Q}^\times, \cdot$ and $\mathbb{F}_q, +$.

Exercise 2.5. Show that a solution to the discrete logarithm problem implies a solution to the Diffie-Hellman problem.

An open problem is the converse of the above statement, i.e. does a solution to the Diffie-Hellman problem imply a solution to the discrete logarithm problem? Recall that a similar question appeared in RSA: is factoring integers as hard as cracking RSA? Whereas people have no clue in the RSA case, the answer in the Diffie-Hellman case is probably yes, for any practical choice of G. This was illustrated by Maurer and Wolf. Their argument uses... elliptic curves! The details can be found in U. Maurer and S. Wolf, *The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms*, SIAM J. Comput. **28**(5), pp. 1689-1721 (1999).

Exercise 2.6. Although key exchange is the more natural discrete logarithm based cryptographic application, one can modify the above to obtain a protocol for pure public key encryption (à la RSA). This is due to Elgamal (1985). Bob's private key is now an integer $k_B \in \mathbb{Z}$ and the corresponding public key is $k_A = g^{k_B}$. To send a message M (considered as an element of G), Alice chooses a random $y \in \mathbb{Z}$ and computes

(1) $C_1 = g^y$,
(2) $C_2 = Mk_A^y$.

She then sends the pair (C_1, C_2) to Bob. How can Bob recover M?

Remark 2.7. The security of the Elgamal encryption scheme can be shown to depend on the a priori weaker *decisional Diffie-Hellman problem*, which is about finding an efficient method to decide whether $h = g^{AB}$, from any given quartet g, g^A, g^B, h.

Exercise 2.8. Give a computer-implementable version of the three-pass protocol that was explained in Section 1.1.1. You are likely to come up with the Massey-Omura cryptosystem. Look it up and compare.

2.2. **Generic attacks.** From now on, we will assume that G is finite and that its group order $n = \#G$ is known, which will be the case in all practical situations. Suppose as well that we have the factorization of n:

$$n = \prod_{i=1}^{k} p_i^{r_i}.$$

Exercise 2.9. Using the fact that there exist fast primality tests, show that we can efficiently obtain this factorization if at most one p_i exceeds a certain given small bound $\beta \in \mathbb{Z}_{\geq 1}$.

Let $p = \max\{p_1, \ldots, p_k\}$. We will show that any discrete logarithm in G with respect to any base g can be computed in $O(\sqrt{p} \cdot \log n)$ steps. The attack consists of two parts: Silver-Pohlig-Hellman reduction and Pollard's ρ method.

2.2.1. *Silver-Pohlig-Hellman reduction.* The Silver-Pohlig-Hellman reduction reduces the discrete logarithm problem in G to at most r_1 discrete logarithm problems in groups of size p_1, plus at most r_2 discrete logarithm problems in groups of size p_2, and so on.

Let $g \in G$ and $h = g^A$ be given. We want to compute $A \bmod \operatorname{ord}(g)$.

(1) *Computing* $\operatorname{ord}(g)$. If

$$m = \operatorname{ord}(g) = \prod_{i=1}^{k} p_i^{s_i} \quad \text{with } s_i \leq r_i,$$

then for $i = 1, \ldots, k$ we can compute s_i by trying whether

$$g^{n/p_i^j} = 1$$

holds for increasing values of $j = 1, \ldots, r_i$. If equality fails for some j, then $s_i = r_i - j + 1$. If equality holds for all j, then $s_i = 0$.

(2) *Reducing the discrete logarithm problem to groups of size $p_i^{s_i}$.* Since $h = g^A$,

$$h^{m/p_i^{s_i}} = \left(g^{m/p_i^{s_i}}\right)^A.$$

Note that

$$\operatorname{ord}\left(g^{m/p_i^{s_i}}\right) = p_i^{s_i},$$

so the above equation determines A mod $p_i^{s_i}$. Solving this equation means computing a discrete logarithm problem in a group of size $p_i^{s_i}$. If we do this for all $p_i^{s_i}$, we can use the Chinese Remainder Theorem to recover A mod m.

(3) *Reducing the discrete logarithm problem to groups of size p_i.* Let $g \in G$ be an element of prime power order p^s and let $h = g^A$. Let $A_0 = A \bmod p$, then we have

$$h^{p^{s-1}} = \left(g^{p^{s-1}}\right)^{A_0},$$

so computing A_0 boils down to computing a discrete logarithm in a group of size p. Once we know A_0, then we are left with

$$hg^{-A_0} = (g^p)^{\frac{A-A_0}{p}},$$

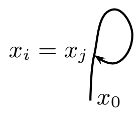
which is a discrete logarithm problem with basis g^p. This is an element of order p^{s-1}. Thus we can repeat the argument: $A_1 = (A - A_0)/p \bmod p$ can be obtained by computing a discrete logarithm in a group of size p. Repeating this s times, we can read off A mod p^s, since we have actually computed the expansion

$$A \bmod p^s = A_0 + A_1 p + A_2 p^2 + \cdots + A_{s-1} p^{s-1}.$$

2.2.2. *Pollard's ρ method.* Let S be any finite set of n elements and let $f : S \to S$ be a random function. Let $x_0 \in S$ be a random element and consider the infinite *random walk*

$$x_0, x_1 = f(x_0), x_2 = f(x_1), x_3 = f(x_2), \ldots$$

Since S is finite, for some smallest j we must have that x_j was already in the list, say $x_j = x_i$ for some $i \leq j$. But then also $x_{j+1} = x_{i+1}$, $x_{j+2} = x_{i+2}$, …: the sequence becomes periodic. The random walk can be pictured as follows:



hence the name Pollard ρ (rho)! By the birthday paradox, such a collision is expected to happen after

$$j = \sqrt{\pi n/2}$$

steps (a formula due to Ramanujan).

How can this be used to compute discrete logarithms? Suppose we are given $g \in G$ and $h = g^A$ as before. Without loss of generality, suppose that G is cyclic and that g is a generator, thus $\#G = \operatorname{ord}(g) = n$. Partition G into three 'random' (but easy to describe) sets G_1, G_2, G_3, such that $1 \neq G_2$. Define the following function

$$f : G \to G : x \mapsto \begin{cases} xh & \text{if } x \in G_1 \\ x^2 & \text{if } x \in G_2 \\ xg & \text{if } x \in G_3, \end{cases}$$

and consider the corresponding 'random' walk starting from $x_0 = 1$. Along the way, keep track of the expansion $x_i = g^{a_i}h^{b_i}$ (starting from $a_0 = b_0 = 0$), where the a_i, b_i only need to be remembered modulo n. Now suppose that at some point we have a collision

$$x_j = x_i \quad \Leftrightarrow \quad g^{a_j}h^{b_j} = g^{a_i}h^{b_i} \quad \Leftrightarrow \quad g^{a_j - a_i} = h^{b_i - b_j}.$$

Then $A \bmod n$ can be recovered as $(a_j - a_i)(b_i - b_j)^{-1}$ (in $\mathbb{Z}/(n)$). This is unless we have bad luck and division fails. In that case, we try again with different G_1, G_2, G_3.

As mentioned, we expect to find a collision after $\sqrt{\pi n/2}$ steps. However, this method has heavy memory requirements, since we need to remember all x_i to check whether we have a collision or not. Here is a trick to get around this, due to Floyd. Instead of computing $x_0, x_1, x_2, \ldots$ and storing all results, we compute the sequence of pairs $(x_1, x_2), (x_2, x_4), (x_3, x_6), (x_4, x_8), \ldots$ and forget about all previous outcomes.

$$(x_{i+1}, x_{2i+2}) = (f(x_i), f(f(x_{2i}))).$$

Exercise 2.10. Prove that we expect to have $x_i = x_{2i}$ for some $i \leq \sqrt{2\pi n}$.

Thus we can still find collisions in $O(\sqrt{n})$ time, using virtually no memory.

2.2.3. *Consequences.* Here are two important conclusions:

(1) *Silver-Pohlig-Hellman:* if the order of G splits into small prime factors, then the Diffie-Hellman protocol is not secure. In any case, each non-trivial factor reduces the security. The ideal situation is where $\#G$ is a large prime number (and $g \neq 1$).
(2) *Pollard ρ:* in order to provide 128-bit security, we have to work in a group of size at least 2^{256}.

3. Elliptic curve cryptography (ECC)

The Diffie-Hellman protocol was initially formulated for multiplicative groups of finite fields $\mathbb{F}_q^\times$. Nowadays, the security of such systems is highly comparable to that of RSA. In particular, to provide 128-bit security, one recommends the usage of 3072-bit keys. The most famous attack is index calculus, see [2, Chapter 20].

In the mid 80's, Miller and Koblitz independently proposed the usage of elliptic curves over finite fields as the underlying group in the Diffie-Hellman protocol. Elliptic curve cryptography was born. Gradually it convinced the cryptographic community of its value, due to

(1) the large amount of effort that was put in improving its performance,
(2) the fact that nobody proved able to come up with an attack that works substantially faster than Pollard's ρ method, meaning that 128-bit security is obtained using 256-bit keys,
(3) the discovery of certain cryptographic protocols that have no non-elliptic alternatives (pairing-based cryptography).

Property (2) is particularly attractive for small devices such as cell phones, bank cards, ID cards, ...

Note that, because we denote the group structure on an elliptic curve additively, the discrete logarithm problem reads

$$\text{given } \mathcal{P} \in E(\mathbb{F}_q) \text{ and } \mathcal{Q} = k \cdot \mathcal{P}, \text{ find } k \bmod \operatorname{ord}(\mathcal{P}).$$

3.1. **Constructive aspects of ECC.**

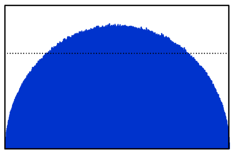
3.1.1. *Finding a good elliptic curve.* By the conclusions in Section 2.2.3, a first aim is to find an elliptic curve E over a finite field $\mathbb{F}_q$ of size about $q \approx 2^{256}$, whose number of elements is prime[3]. This can be done by trying random elliptic curves, each time applying the SEA algorithm to compute the number of points. By the following exercise, we can typically verify two curves at once.

Exercise 3.1. Let $E : y^2 = x^3 + Ax + B$ be an elliptic curve over $\mathbb{F}_q$, and suppose it has trace of Frobenius t. Let $d \in \mathbb{F}_q$ be non-square. Then show that the curve $E^T : y^2 = x^3 + Ad^2x + Bd^3$ has trace of Frobenius $-t$. Hint: first prove that the number of affine points on the latter curve equals the number of solutions to $dy^2 = x^3 + Ax + B$. The curve E^T is called a *quadratic twist* of E. Give a concrete example of a quadratic twist E^T that is equal to E.

The number of points is a 'random' integer in the Hasse interval $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$. By the prime number theorem, we expect to find a good curve after a couple of hundreds of steps. Although this seems to match with practice (in fact, practice does a little worse, see (4) below), it is only heuristical:

(1) It is not even known whether each Hasse interval contains at least one prime number.
(2) The above 'randomness' is not according to the uniform distribution. The trace of Frobenius t roughly follows the semicircular distribution

$$\frac{1}{2q\pi}\sqrt{4q-t^2}dt :$$



(3) $\#E(\mathbb{F}_q)$ strongly tends to avoid numbers that are congruent to 1 mod $p = \operatorname{char} \mathbb{F}_q$ (curves with such a number of points are supersingular – see Theorem 3.9).
(4) $\#E(\mathbb{F}_q)$ slightly favors small prime factors. E.g., the probability that $\#E(\mathbb{F}_q) \equiv 0 \bmod 2$ tends to $2/3$ as $q \to \infty$. Note that $\#E(\mathbb{F}_q) \equiv 1 \bmod 2$ if and only if $x^3 + Ax + B$ is irreducible. Now prove the following:

   Exercise 3.2. Show that the probability that a random monic cubic polynomial is irreducible tends to $1/3$ as $q \to \infty$. Hint: observe that any such polynomial is the minimal polynomial over $\mathbb{F}_q$ of three elements of $\mathbb{F}_{q^3}$.

   One can show that the same statistics apply to the less general family of polynomials $x^3 + Ax + B$.

[3]In practice, a small cofactor is tolerated.

Alternatively, there exist efficient methods that *construct* an elliptic curve with a given prime as number of rational points. However, for these methods, one cannot fix the field in advance (which one usually prefers to do for reasons of fast field arithmetic).

Further on, we will describe a number of additional obstructions for an elliptic curve to be suitable for cryptography. This section will be updated accordingly in Summary 3.10.

3.1.2. *Finding a point $\mathcal{P} \in E(\mathbb{F}_q)$.* Finding a nontrivial $\mathbb{F}_q$-rational point on E is done as you would expect: one takes a random $x \in \mathbb{F}_q$, computes $x^3 + Ax + B$ and verifies whether it is a square or not by computing the quadratic character (Legendre symbol if q is prime). If it is, compute a square root: there exist efficient methods for doing this. If $q \equiv 3 \bmod 4$, computing square roots is even almost trivial:

Exercise 3.3. If $q \equiv 3 \bmod 4$ and $z \in \mathbb{F}_q$ is a square, then prove that $z^{(q+1)/4}$ is a square root.

For the case $q \equiv 1 \bmod 4$, see [3, 6.1.8].

3.1.3. *Fast arithmetic.* During key generation, Alice and Bob have to compute several times a large multiple $k \cdot \mathcal{P}$ of a certain affine point $\mathcal{P} = (x_1, y_1) \in E(\mathbb{F}_q)$. The analog of the repeated squaring method is the *double-and-add method.* Let

$$\mathtt{1}\ \mathfrak{b}_1\ \mathfrak{b}_2\ \mathfrak{b}_3\ \ldots\ \mathfrak{b}_n$$

be the binary expansion of k. Then the routine

```
(1) $\mathcal{Q} := \mathcal{P}$
(2) for $j = 1, \ldots, n$ do
    (a) if $\mathfrak{b}_j = 0$ then $\mathcal{Q} := 2\mathcal{Q}$,
    (b) if $\mathfrak{b}_j = 1$ then $\mathcal{Q} := 2\mathcal{Q} \oplus \mathcal{P}$.
```

gives $\mathcal{Q} = k\mathcal{P}$ (verify this!) using roughly $\log_2 k$ doublings and, on average, $(\log_2 k)/2$ additions.

Since subtracting a point $\mathcal{P} = (x_1, y_1)$ means just adding $\mathcal{P} = (x_1, -y_1)$, it may be useful to allow `-1`'s in the binary expansion. For instance, it is better to write

$$31 = \mathtt{1\ 0\ 0\ 0\ 0\ \text{-}1} \quad \text{instead of} \quad 31 = \mathtt{1\ 1\ 1\ 1\ 1}.$$

This is called a NAF (non-adjacent form), which is non-unique. There exist quick methods to compute an optimal NAF, i.e. one with the maximal number of zeroes. On average, this reduces the number of additions to $(\log_2 k)/3$.

Another speed-up can be obtained using *projective coordinates.* Let us recall the formulas

$$(x, y) \mapsto \left(\left(\frac{3x^2 + A}{2y} \right)^2 - 2x, -y + \left(\frac{3x^2 + A}{2y} \right) (x - x_3) \right)$$

(where x_3 is the first component) for doubling, and

$$(x_1, y_1), (x_2, y_2) \mapsto \left(\left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2, -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) \right)$$

(where x_3 is the first component) for addition. Since in practice $\mathcal{P}$ will be a point of large prime order, we don't need to care about the exceptional case $x_1 = x_2, y_1 =$

$-y_2$. Now a problem is that repeatedly applying these formulas involves a lot of divisions in $\mathbb{F}_q$, which becomes time costly. To solve this problem, one uses projective coordinates. The above doubling formulas can be rewritten as follows (substitute $x \leftarrow x/z, y \leftarrow y/z, x_3 \leftarrow x_3/z_3$):

$$(1) \qquad (x,y,z) \mapsto \big(2yzT, -8y^4z^2 + (3x^2 + Az^2)(4xy^2z - T), 8y^3z^3\big)$$

with $T = (3x^4 + 6Ax^2z^2 + Az^4 - 8xy^2z)$.

Exercise 3.4. Do the same for the addition formulas.

The procedure for computing $k \cdot \mathcal{P}$ is then: start with $(x_1, y_1, 1)$, compute projective coordinates (x_k, y_k, z_k) of $k \cdot \mathcal{P}$ using the double-and-add method (or one of its variants), and output $(x_k/z_k, y_k/z_k)$. In this way $O(\log_2 k)$ field inversions are replaced by just 1 field inversion. This comes at the price of some additional field multiplications and additions, but these have a much lower time-cost.

We remark that both the double-and-add method and the use of other coordinate systems are still in full development. For a more up-to-date account, see [2]. See also Section 4.1.3.

3.2. **A digital signature algorithm.** One can reverse the idea of public key cryptography. Normally, Bob has a private key k_B and a corresponding public key k_A. Alice can then freely use k_A to encrypt a message. Decrypting however can only be done using k_B.

Conversely, suppose that Alice wants Bob to send her some contract. She wants Bob to *sign it*, so that he can never deny having read the contract. For that, Bob uses his private key k_B. Verifying the signature is then done using the public key k_A. Here is how it works to sign a message M in the elliptic curve setting. Both parties have publicly agreed on a curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = \ell$ prime and a non-trivial point $\mathcal{P} \in E(\mathbb{F}_q)$. Bob has a secret key $k_B \in \{2, \ldots, \ell - 1\}$, and a corresponding public key $k_A = \mathcal{Q} = k_B \cdot \mathcal{P}$.

First, the message M is *hashed* into $h(M) \in \mathbb{F}_\ell$.

Definition 3.5. *A* hash function *is a publicly known, efficiently computable map $h : S \to T$, where usually $\#S > \#T$, such that it is computationally infeasible to find two different $s_1, s_2 \in S$ for which $h(s_1) = h(s_2)$. It is allowed that $\#S = \infty$ (e.g. the set of all binary strings).*

The theory of hash functions is a branch on its own, we will just assume that such maps exist. Note that there exist weaker definitions. Famous hash functions are `md5sum`, `SHA-1` and `SHA-2`.

We also use a function $\phi : \mathbb{F}_q \to \mathbb{F}_\ell^\times$ that is 'almost' 1-1. Note that $\ell \approx q$ by Hasse's theorem. For instance, let $q = p^n$. An element of $x_0 \in \mathbb{F}_q$ can be seen as the expansion to base p of an integer in the range $\{0, \ldots, p^n - 1\}$. Then we could take $\phi(x_0) = 1 + (x_0 \bmod (\ell - 1))$.

Then the signing protocol is

(1) Bob chooses another random $k \in \mathbb{F}_\ell^\times$ and computes $k \cdot \mathcal{P}$. Denote the x-coordinate by x_0. He also computes k^{-1} in $\mathbb{F}_\ell^\times$.
(2) Bob computes $s = k^{-1}(h(M) + k_B\phi(x_0)) \in \mathbb{F}_\ell$. If $s = 0$, he goes back to Step (1).
(3) The signature for M is (x_0, s).

To verify the signature, Alice takes the following steps.

(1) Alice computes $w = s^{-1} \in \mathbb{F}_\ell^\times$.
(2) She computes $u_1 = wh(M)$ and $u_2 = w\phi(x_0)$.
(3) She computes $\mathcal{R} = u_1\mathcal{P} \oplus u_2\mathcal{Q}$.
(4) Alice verifies that the x-coordinate of $\mathcal{R}$ is x_0.

Exercise 3.6. Prove that, if everything went fine, the x-coordinate of $\mathcal{R}$ is indeed x_0.

Exercise 3.7. Use Definition 3.5 to show that neither Bob nor Alice can claim afterwards that *another* message M was signed.

One can prove that if Eve plans to forge a signature, then she should either crack the hash function, either solve a discrete logarithm problem. Thus under the assumption that these tasks are infeasible, the elliptic curve digital signature algorithm (ECDSA) is *provably* secure.

In a sense, the above instance of elliptic curve cryptography is even of greater relevance than key exchange. This is because, in the typical domains of applicability of digital signatures, short key lengths are highly desirable. For instance, as will become reality in the near future, people will be expected to sign documents using their personal ID card. Then ECDSA is the way to go.

3.3. **Constructive and destructive aspects of the Weil pairing.** Disclaimer: although related to the Weil pairing, there exist other pairings on elliptic curves (Tate pairing, ate pairing, eta pairing, ...) that are sometimes better suited for the applications we will explain below.

Let m be coprime to q. Then recall that the Weil pairing of level m is a map

$$e_m : E[m] \times E[m] \to \mu_m = \{m^{\text{th}} \text{ roots of unity in } \overline{\mathbb{F}}_q\}$$

satisfying

(1) *bilinearity*, meaning that

$$e_m(\mathcal{P}_1 \oplus \mathcal{P}_2, \mathcal{Q}) = e_m(\mathcal{P}_1, \mathcal{Q})e_m(\mathcal{P}_2, \mathcal{Q}) \quad \text{and} \quad e_m(\mathcal{P}, \mathcal{Q}_1 \oplus \mathcal{Q}_2) = e_m(\mathcal{P}, \mathcal{Q}_1)e_m(\mathcal{P}, \mathcal{Q}_2),$$

(2) *alternation*, in the sense that

$$e_m(\mathcal{P}, \mathcal{Q}) = e_m(\mathcal{Q}, \mathcal{P})^{-1}$$

(3) *nondegeneracy*: if $e_m(\mathcal{P}, \mathcal{Q}) = 1$ for all $\mathcal{P} \in E[m]$, then $\mathcal{Q} = \mathcal{O}$.

The Weil pairing is efficiently computable. This can be seen from its definition and the constructive proof of Theorem I.3.12. The resulting routine is called *Miller's algorithm*.

Note that the Weil pairing is a priori defined over $\overline{\mathbb{F}}_q$ only. But of course, since $E[m]$ is finite and e_m respects the field of definition (see the proof of Corollary I.3.17), the whole story happens over some finite extension $\mathbb{F}_{q^k}$.

Definition 3.8 (not standard!). *Let m be coprime to q. The* embedding degree *of an elliptic curve $E/\mathbb{F}_q$ with respect to m is the smallest k such that $E[m] \subset E(\mathbb{F}_{q^k})$.*

Now recall that if $E[m] \subset E(\mathbb{F}_{q^k})$, then $q^k \equiv 1 \bmod m$ by Corollary I.3.17. Thus the order of q in $\mathbb{Z}/(m)^\times$ is a lower bound for the embedding degree. A more standard definition of 'embedding degree' is simply the order of q in $\mathbb{Z}/(m)^\times$, because in fact:

(1) Both notions are typically equal. For instance, one can prove this to be true if m is coprime to $q - 1$.

(2) The similar notion for the Tate pairing always matches with this.

In any case, a lesson to be learned is that the embedding degree is usually at least about $\varphi(m)$.

3.3.1. *Menezes-Okamoto-Vanstone attack (MOV).* Now let $E/\mathbb{F}_q$ be an elliptic curve of prime order ℓ and suppose that ℓ is coprime to q. Let k be the embedding degree with respect to ℓ.

Let $\mathcal{P} \in E(\mathbb{F}_q)$ be the generator Alice and Bob commonly agreed upon. Let k_B be Bob's secret key, and let $k_A = \mathcal{Q} = k_B\mathcal{P}$ be the corresponding public key. Now evil Eve chooses a random point $\mathcal{E} \in E[\ell] \subset E(\mathbb{F}_{q^k})$ and she computes

$$e_\ell(\mathcal{P}, \mathcal{E}) = g \in \mu_\ell \subset \mathbb{F}_{q^k}.$$

If $\mathcal{E}$ was randomly chosen, then g is likely to be a generator of μ_ℓ. Eve also computes

$$e_\ell(\mathcal{Q}, \mathcal{E}) = h \in \mu_\ell \subset \mathbb{F}_{q^k}.$$

But Eve knows about the properties of the Weil pairing, so she realizes that

$$h = e_\ell(\mathcal{Q}, \mathcal{E}) = e_\ell(k_B\mathcal{P}, \mathcal{E}) = e_\ell(\mathcal{P}, \mathcal{E})^{k_B} = g^{k_B}.$$

Hence she can recover $k_B \bmod \ell$ by solving a discrete logarithm problem in $\mathbb{F}_{q^k}^\times, \cdot$, where better attacks exist (e.g. index calculus)!

However, as said, k is expected to be of size $\varphi(\ell) = \ell - 1$. So the field in which these better attacks are about to be applied are mostly extremely large. However, there exist elliptic curves with small embedding degree. For these curves, the above attack truly reduces the safety of the system. An important class of curves are the so-called *supersingular* curves. A definition was given below Theorem I.3.7; in our case it can be shown equivalent to $\ell \equiv 1 \bmod p$, where p is the field characteristic.

Theorem 3.9. *If $\ell \equiv 1 \bmod p$, then the embedding degree is at most 2.*

Recall from Section 3.1.1 that such curves are rare. We omit the proof of Theorem 3.9.

3.3.2. *One-round three-party key exchange.* Here is a constructive facet of Weil pairings. Again we work with curves having small embedding degree. As explained above, the discrete logarithm problem then transfers to the multiplicative group of a finite field, where better attacks apply. In particular, we lose the benefit of having short key lengths. But there come some advantages in place.

Suppose we have three people, Alice, Bob and Charlie, that want to agree upon a common key. One way to do this would be

(1) Alice and Bob agree upon a key $k_Ak_B\mathcal{P}$ using the Diffie-Hellman protocol.
(2) They interpret the x-coordinate as an element $k_{AB} \in \mathbb{F}_\ell$ (e.g. using the map ϕ from Section 3.2).
(3) Both now agree with Charlie upon a key $k_{AB}k_C\mathcal{P}$.

Using pairings, we can do this in *one round*, following an idea of Joux. Suppose we have two points $\mathcal{P}, \mathcal{Q}$ for which $e_\ell(\mathcal{P}, \mathcal{Q}) = g \neq 1$. Alice, Bob and Charlie respectively drop $(k_A\mathcal{P}, k_A\mathcal{Q})$, $(k_B\mathcal{P}, k_B\mathcal{Q})$ and $(k_C\mathcal{P}, k_C\mathcal{Q})$ in a public pool. Then all can compute

$$g^{ABC}$$

as

$$e_\ell(k_B\mathcal{P}, k_C\mathcal{Q})^{k_A}, \quad e_\ell(k_A\mathcal{P}, k_C\mathcal{Q})^{k_B} \quad \text{and} \quad e_\ell(k_A\mathcal{P}, k_B\mathcal{Q})^{k_C}$$

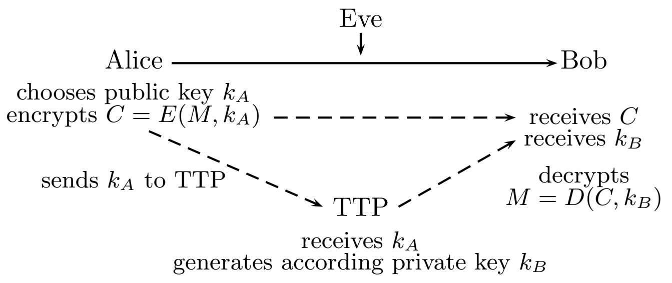
respectively. Please be aware that the description is somewhat artificial, it works better with other pairings. But the idea should be clear.

3.3.3. *Identity-based cryptography.* Let us have another look at public key encryption (RSA, Elgamal), from the following practical point of view. Suppose Alice wants to send a message to Bob, but Bob did not generate a public key yet. Then the following steps have to be undertaken:

(1) Alice publicly asks Bob: "Please give me a public key";
(2) Bob generates a private key k_B and an according public key k_A; he sends the public key k_A to Alice;
(3) Alice uses k_A to encrypt her message; she then sends it to Bob;
(4) Bob receives the message and decrypts it using k_B.

Thus, Alice has to wait for Bob to send her a public key, before she can start writing her message. This can be annoying: Bob could be on a holiday, or he could just be an irregular internet user.

For this and other reasons, an attractive alternative would be that Alice *chooses the public key k_A herself.* This seems a stupid idea: for Bob it would be infeasible to extract the corresponding private key k_B since the underlying function is one-way. But using a trusted third party (TTP), the following construction is in fact possible:



Recall that we need a TTP anyway, to provide authentication certificates (see Section 1.5.2 – we are not dealing with questions of this type here). The public key k_A could simply be Bob's e-mail address, or anything that identifies him. Hence the name *identity-based cryptography.*

The first practical realization of the above diagram was published in 2001 by Boneh and Franklin. Their construction makes use of the Weil pairing. In particular, they work on an elliptic curve $E/\mathbb{F}_q$ with a prime number of rational points $\#E(\mathbb{F}_q) = \ell$. Suppose the embedding degree k (with respect to ℓ) is small. Again, this means that we lose the benefit of having short key lengths. All parties publicly agree upon a random point $\mathcal{P} \in E[\ell] \subset E(\mathbb{F}_{q^k})$. The TTP chooses a private *master key* $k_{\mathrm{TTP}} \in \mathbb{Z}$ and publishes the corresponding public key $k_{\mathrm{TTP}} \cdot \mathcal{P}$. We also suppose that there is a publicly known hash function h that sends arbitrary strings to points in $E[\ell]$.

Here is what Alice does to encrypt her message $M \in \mathbb{F}_{q^k}$:

(1) She chooses for instance $k_A =$ `bob@gmail.com`.
(2) She chooses a random $r \in \mathbb{Z}$ and computes $\mathcal{R} = r\mathcal{P}$.
(3) She computes $\mathcal{Q} = h(k_A)$.
(4) She computes $S = e_\ell(k_{\mathrm{TTP}}\mathcal{P}, \mathcal{Q})^r$.
(5) Then C is the pair $(\mathcal{R}, M + S)$.

Here is what the TTP does to generate Bob's private key:

(1) He computes $\mathcal{Q} = h(k_A)$.
(2) Then $k_B = k_{\text{TTP}}\mathcal{Q}$.

Here is what Bob does to recover the message:

(1) He computes $e_\ell(\mathcal{R}, k_B) = e_\ell(r\mathcal{P}, k_{\text{TTP}}\mathcal{Q}) = e_\ell(k_{\text{TTP}}\mathcal{P}, \mathcal{Q})^r = S$.
(2) He recovers $M = (M + S) - S$.

As in the case of Elgamal encryption (see Remark 2.7), the security of the Boneh-Franklin protocol can be reduced to the hardness of a problem that is a priori weaker than the discrete logarithm problem, namely the *decisional bilinear Diffie-Hellman problem*, see [2, 24.1.1].

3.4. **Other attacks and threats.** Recapitulating, in order for an elliptic curve $E/\mathbb{F}_q$ to suit for cryptographic purposes, we need that

(1) $\#E(\mathbb{F}_q)$ is a prime number ℓ (perhaps modulo a small factor),
(2) E should have a large embedding degree with respect to ℓ (unless one allows larger key lengths, see Sections 3.3.2 and 3.3.3 for why this might be useful) – this is generically satisfied.

Are there other 'weak' curves to be avoided? The answer is yes. One class consists of so-called *anomalous curves*, which are curves for which ℓ equals q (thus q should be prime). Note that the Weil pairing is not even defined for such curves, so the MOV attack does not apply at all. However, anomalous curves turn out to be *very* weak. Whereas the MOV attack is about transferring the discrete logarithm problem to the multiplicative group of a finite field, the discrete logarithm problem on anomalous curves can be transferred to the *additive group* of a finite field. In such groups, computing discrete logs is trivial. For a detailed description, see [1, V.3].

A seemingly bigger threat, the extent of which is not so well understood, is the *Weil descent attack*. The mathematics involved is more advanced. But very briefly, and at the risk of just confusing the reader, the idea is that an elliptic curve E over a finite field $\mathbb{F}_{q^d}$ can in a natural way be given the structure of a d-dimensional variety over $\mathbb{F}_q$. Roughly, this goes as follows: suppose E is defined by $y^2 = x^3 + Ax + B$. Let $\theta_1, \ldots, \theta_d$ be an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^d}$ (as a vector space) and expand

$$A = A_1\theta_1 + \cdots + A_d\theta_d \quad \text{and} \quad B = B_1\theta_1 + \cdots + B_d\theta_d.$$

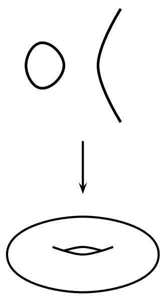
Rewriting

$$x = x_1\theta_1 + \cdots + x_d\theta_d \quad \text{and} \quad y = y_1\theta_1 + \cdots + y_d\theta_d$$

for new variables $x_1, \ldots, x_d, y_1, \ldots, y_d$, and substituting everything in $y^2 - (x^3 + Ax + B)$ eventually leads to an expression

$$f_1(x_1, \ldots, x_d, y_1, \ldots, y_d)\theta_1 + \cdots + f_d(x_1, \ldots, x_d, y_1, \ldots, y_d)\theta_d$$

with $f_1, \ldots, f_d \in \mathbb{F}_q[x_1, \ldots, x_d, y_1, \ldots, y_d]$. Then $f_1, \ldots f_d$ define a d-dimensional variety in affine $2d$-space, which is defined over $\mathbb{F}_q$. The same construction can be done on the projective level, and the resulting d-dimensional variety is called the *Weil descent* $W_{\mathbb{F}_{q^d}/\mathbb{F}_q}(E)$ of E. The reference picture you should have in mind is that of a complex elliptic curve $E/\mathbb{C}$, which is topologically a torus. But this torus can of course also be seen as a real surface. The Weil descent $W_{\mathbb{C}/\mathbb{R}}(E)$ is exactly this real surface.



Now there is a natural bijection

$$W_{\mathbb{F}_{q^d}/\mathbb{F}_q}(E)(\mathbb{F}_q) \leftrightarrow E(\mathbb{F}_{q^d}).$$

In particular, the Weil descent is endowed with an algebraic group structure, turning it into an *abelian variety* over $\mathbb{F}_q$. This abelian variety might contain the *Jacobian variety* of a curve of relatively high genus; we already dropped this notion at the end of Section I.3.2. With some probability, the above bijection maps our two points of interest $\mathcal{P}, \mathcal{Q} = k\mathcal{P} \in E(\mathbb{F}_q)$ into this Jacobian variety. Now for curves of genus $g \geq 3$, the discrete logarithm problem on their Jacobian is known to be somewhat easier than in the genus one (elliptic curves) case. For the details, we refer to [2].

The threat of Weil descent is reality: elliptic curves to which the above attack applies have indeed been constructed. In practice, one therefore works either

(1) over prime fields $\mathbb{F}_p$, where no Weil descent exists,
(2) over fields of large prime extension degree, e.g. $\mathbb{F}_{5^{113}}$, where the probability of mapping the discrete logarithm problem into a Jacobian variety in the (only possible) Weil descent becomes very small.

Let's sum up:

Summary 3.10. Here is an update of Section 3.1.1 about finding good elliptic curves for usage in key exchange protocols or digital signature schemes:

(1) Let $\mathbb{F}_q$ be a finite prime field, or a prime degree extension of a very small finite field, where $q \approx 2^{256}$.

(eliminate the Weil descent attack
and ensure 128-*bit security (Pollard* ρ*))*

(2) Take a random elliptic curve $E/\mathbb{F}_q$.
(3) Using the SEA algorithm, check whether $\#E(\mathbb{F}_q)$ is a prime number ℓ (possibly up to a small cofactor). If not, go back to (2).

(eliminate the Silver-Pohlig-Hellman attack)

(4) Compute the order of q in $\mathbb{F}_\ell^\times$. If it is small, go back to (2).

(eliminate the MOV attack)

(5) If $q = \ell$, then go back to (2).

(eliminate the anomalous attack)

Failures at Steps (4) and (5) are highly unlikely.

4. ECC in practice

4.1. **A summary of ECC's features.**

4.1.1. *Short key lengths.* We mentioned this a couple of times already. ECC's main alternatives are either based on the hardness of integer factorization, or on the hardness of discrete log computation in groups of the form $\mathbb{F}_q^\times$. Due to certain nontrivial attacks (elliptic curve factorization, number field sieve, index calculus), such cryptosystems are thought to require a key of 3072 bits in order to provide 128-bit security. In ECC, a key of 256 bits currently suffices. For small devices such as ID cards and cell phones, this makes a big difference. Moreover, looking towards the future where higher security levels will become standard, the relative difference will grow exponentially. To illustrate this: to provide 256-bit security, NIST recommends an RSA key of 15360 bits. This should be compared with an ECC key of 512 bits.

4.1.2. *Pairing-based cryptography.* In Section 3.3, we met two constructive applications of the Weil pairing: one-round three-party key exchange and identity-based cryptography. Especially the latter application is currently a hot topic. Although both constructions only depend on certain axiomatic properties of pairings (bilinearity, nondegeneracy), elliptic curves seem unavoidable. Remark that for pairing-based applications, one needs elliptic curves having small embedding degree, hence one loses the benefit of having short key lengths.

4.1.3. *Flexibility.* Another important advantage of ECC is its flexibility. Once the finite field $\mathbb{F}_q$ has been fixed – which is often done in advance, since some fields allow faster arithmetic than others – there is only one $\mathbb{F}_q^\times$, but there are many elliptic curves. This allows cryptographers to take into account other design parameters: a small embedding degree (for pairing-based applications), a small curve parameter $A \in \mathbb{F}_q$ to speed up arithmetic (cf. formula (1)), and so on. Remark that there is no need to stick to *Weierstrass curves* $y^2 = x^3 + Ax + B$. In Exercise I.1.9, we mentioned that any nonsingular cubic C and any choice of base point $\mathcal{O}_C$ gives rise to an algebraic group structure. Although this construction does not result in new groups in the mathematical sense, the formulas for addition and doubling may become more efficient. Two popular choices are *Montgomery curves* $By^2 = x^3 + Ax^2 + x$ (where $(A^2 - 4)B \neq 0$) with base point $(0, 1, 0)$ and *Hessian curves* $x^3 + y^3 + 1 = 3Axy$ (where $A^3 \neq 1$) with base point $(1, -1, 0)$. For more details on these and other forms, see `http://www.hyperelliptic.org/EFD`.

What about non-elliptic curves, or higher-dimensional varieties? Since the Diffie-Hellman protocol can be formulated in any group, it is natural to ask whether algebraic geometry has more to offer.

1. A theorem by Chevalley states that the only curves allowing an algebraic group structure are the non-projective $\mathbb{F}_q, +$ (affine line) and $\mathbb{F}_q^\times, \cdot$ (hyperbola, verify this!), and elliptic curves. (This is up to $\mathbb{F}_q$-isomorphism – whatever this means.) So nothing new here.
2. In higher dimensions, there is more lying around. However, apart from whether such varieties are interesting for cryptographic purposes, very often there are constructive problems. Can we efficiently compute on such varieties? Can we efficiently determine their number of rational points?

A particular class has seen thorough analysis: *Jacobians of curves.* The Jacobian of a projective curve is roughly defined as the 'smallest' projective variety that contains the curve and that can be endowed with an algebraic group structure. The Jacobian of an elliptic curve is the curve itself. The Jacobian of a curve of higher genus $g \geq 2$ is a variety of dimension g. An attractive property of the Jacobian of a curve C is that its group structure can be described as a *divisor class group* on C (see I.3 and in particular the remark at the end of I.3.2). This can be exploited for efficient arithmetic. Moreover, for certain curves, fast point counting on their Jacobian is possible.

Already in the late 80's, *hyperelliptic curve cryptography* (HECC) was proposed as a valuable alternative to ECC. However, for large values of g, HECC soon proved weaker than ECC, due to the applicability of index calculus. Gradually, the bound on the genus was brought down to $g \geq 4$, and recently even to $g \geq 3$ due to a trick by Smith. (These attacks are an essential ingredient in the Weil descent threat described in Section 3.4.)

By now, HECC has lost some of its status. This is strengthened by a wide-spread, somewhat fatalistic belief that any successful attack on ECC would probably also apply to HECC. It seems safe to state that, in the near future, HECC will not find its way to practice. Nevertheless, the genus 2 case remains completely unbroken and deserves further development. This is not a matter of holding tight to the last piece of wood floating around. First, the above attacks are fundamentally *not* applicable in the genus 2 case. Second, constructive aspects such as point counting and fast public key generation can be dealt with particularly well, certainly over finite fields of small characteristic. Third, it is certainly useful to have a competitive alternative to ECC. For instance, the best algorithms for public key generation on elliptic curves have benefited from the similar progress that was made in the hyperelliptic curve case. In this context, note also that many ECC protocols have been patented (mainly by CERTICOM). So one could think of HECC as a freeware version of ECC, forcing the commercial people to improve their product.

4.2. **Side-channel attacks.** This is an intriguing class of attacks that become highly relevant when proceeding to a practical implementation of ECC (or any other cryptosystem). Suppose Bob's computer is generating a public key $k_A = k_B \cdot \mathcal{P}$ from a private key k_B. Then the power it consumes, the sound it makes, the radiation emanating from it, the time it takes at certain steps, ... all leak information about k_B. Here is a notorious example:

Recall from Section 3.1.3 that Bob computes $k_B\mathcal{P}$ using the double-and-add method or one of its variants. In its simplest form, this means that Bob's computer iterates through the binary digits of k_B.

(1) If a `0` is processed then $\mathcal{Q} := 2\mathcal{Q}$ is computed.
(2) If a `1` is processed then $\mathcal{Q} := 2\mathcal{Q} \oplus \mathcal{P}$ is computed.

Thus, processing a `1` is somewhat harder than processing a `0`, because of the extra addition. This is reflected in the power consumption of Bob's computer. Every time a digit is processed, this will cause a small peak in the power consumption. If the digit is a `1`, this peak will be of a different kind than when a `0` is processed: it will probably be higher and last longer. So if the implementation is naive, Eve can almost *read* Bob's private key from the power consumption chart!

Side-channel attacks are typically countered by inserting *noise*. We won't go into further details on this.

4.3. **Will ECC ever be broken?** Although so far, no one seems to have found an attack that run substantially faster than Pollard's ρ method, this does not mean that such attacks do not exist. All attempts to *prove* the hardness of the discrete logarithm problem on elliptic curves ran into failure. This is not surprising: a proof that one cannot compute discrete logarithms in polynomial running time would solve the 'P vs. NP' problem, one of the most intriguing open problems in mathematics and theoretical computer science[4].

Exercise 4.1 (for those who know about P vs. NP). Why would such a proof solve this problem?

The P vs. NP problem is believed to be extremely difficult – virtually no progress has been made – and it is imaginable that none of the currently living people will ever see a solution to it. This brings us to the following unpleasant observation: our only security certificate is (and might still for a long while be) that a lot of people have spent many years of their professional life to crack ECC, without success.

For the rigorous mathematician, this security certificate seems worthless. But sometimes, in real life, an engineery attitude is appropriate. The fact that many smart people found no efficient way of solving the discrete logarithm problem, despite years of effort, *does* mean something. The designers of fully practical, ready-to-use cryptosystems, have quite some trust in the hardness of the DLP on elliptic curves. For them, the theoretical aspect of ECC is only one link in a long chain. Some of the other links are a bigger concern.

4.4. **ECC in daily life.** Anno 2009, ECC gradually sneaks into real life. So far, the most prominent step towards full acceptance is probably the inclusion of ECC-based protocols in the US Government's 'Suite B', which serves as the base for sending both unclassified and highly classified information. More precisely, the components of Suite B are

(1) AES for symmetric encryption,
(2) elliptic curve based Diffie-Hellman key exchange,
(3) ECDSA for digital signatures,
(4) a couple of hash functions (`SHA-256` and `SHA-384`).

A number of ECC subprotocols are patented, mainly by Certicom Corp., a Canadian company. These all have been licensed for US Government use.

Other companies currently using ECC are: Pitney Bowes (a digital postage mark company), Nintendo (for game saves in the Wii), BlackBerry (the well-known PDA), Microsoft (for managing digital rights in Windows Media Player), and many others.

[4]It is one of the seven *millennium problems*. For each of these problems, the Clay Mathematics Institute offers the first person to solve it a prize of $1,000,000$ dollars. Also the Birch & Swinnerton-Dyer conjecture, mentioned in I.1.4, is contained in the list.