Elliptic Curves in Cryptography

Lecture notes from course CIT413030 at TUM in summer 2024

Lorenz Panny

Version from September 30, 2024

First lecture (April 15)

1 Public-key cryptography and the discrete-logarithm problem

Cryptography is the art and science of securing (digital) communication. It sits right at the intersection of mathematics and computer science. The most important applications of elliptic curves in cryptography lie in *public-key cryptography*.

1.1 What is public-key cryptography?

Public-key cryptography (a.k.a. asymmetric cryptography) refers to types of cryptographic constructions where *not all participants have the same capabilities*. This is in contrast to symmetric cryptography (a.k.a. secret-key cryptography), where knowledge of a previously shared secret key enables everyone to do the same things, e.g., encrypt *and* decrypt. Typical public-key primitives include:

- **Public-key encryption**: The user "Alice" publishes a *public key* which allows anyone to *encrypt* a message in such a way that only she can *decrypt* it using the associated *private key*. (Real-world analogy: An open padlock.)
- Closely related: **Key exchange** over an insecure channel. Two parties "Alice" and "Bob" are communicating, an *eavesdropper* "Eve" listens, yet only Alice and Bob should be able to agree on a *shared secret* that Eve cannot learn.
- **Digital signatures**: The user "Alice" publishes a *public key* which allows anyone to *verify* signed messages that only she can produce using the associated *private key*. (Real-world analogy(?): Writing one's name on paper.)

1.2 The Diffie–Hellman key exchange

The idea of public-key cryptography was first published by Diffie and Hellman in a 1976 paper of only 11 pages titled "New Directions in Cryptography". Among other things, they described the following key-exchange method, which remains one of the most important cornerstones of modern cryptography:

- **Fixed parameters**: A "carefully chosen" prime p, and a primitive element $g \in \mathbb{F}_p^\times$.
- Alice's **private key**: Randomly chosen $a \in \{0, \ldots, p-2\}$.
- Alice's **public key**: The power $A = g^a \in \mathbb{F}_p$.
- Bob's **key pair** (b, B) is constructed identically.
- Their **shared secret** is $g^{ab} = B^a = A^b \in \mathbb{F}_p$.

The idea here is that performing an exponentiation is "easier" than computing a logarithm. Indeed, for Alice and Bob, the complexity of computing the public from the private key is only logarithmic using the *square-and-multiply algorithm*:

1 Lemma. *Let $(G, \cdot)$ be a group, $g \in G$, and $x \in \mathbb{Z}_{\geq 0}$. Then g^x can be computed using no more than $2\lfloor \log_2(x) \rfloor + 1$ multiplications in G.*

Proof. Let $x = 2^n x_n + 2^{n-1} x_{n-1} + \cdots + x_0$ with all $x_i \in \{0, 1\}$. (This is the binary expansion of x.) Then

$$g^x = \Big(\Big(\cdots \big((g^{x_n})^2 \cdot g^{x_{n-1}} \big)^2 \cdots \Big)^2 \cdot g^{x_1} \Big)^2 \cdot g^{x_0} .$$

□

On the other hand, to break the key exchange, it clearly suffices to recover x from g^x. How hard is that?[1]

1.3 The discrete-logarithm problem

2 Definition. Let $(G, \cdot)$ be a group and $g \in G$. The **discrete-logarithm problem (DLP)** is to find $x \in \mathbb{Z}$ given $g^x \in G$.

How can we approach this problem?

- Trivial brute force: Iterate through powers of g until g^x is discovered. The complexity is clearly $O(|G|)$.
- Smarter brute force: Let $m := \lceil \sqrt{|G|} \rceil$. Make a lookup table mapping $g^{mi} \mapsto i$ for $0 \leq i < m$. Then, iterate through elements $g^x \cdot g^{-j}$ for $0 \leq j < m$ until it is in the table. At that point, we have i and j such that $g^{mi} = g^x \cdot g^{-j}$, hence $mi + j$ is a solution to the DLP.[2]

  This is Shanks' *baby-step giant-step* algorithm (BSGS). Its complexity is $O(\sqrt{|G|})$.

It is known that the square-root complexity is essentially optimal for prime-order *generic groups*, that is, without using any specific properties of the underlying group.

$\Longrightarrow$ For generic groups, there is an *exponential separation* between computing exponentiations and logarithms.

3 Remark. We shall later see that there are algorithms with better complexity for generic *composite-order* groups.

4 Remark. In actual reality, *there are no generic groups*: To compute in a group, we have to know how elements are represented as bit strings and how the group operation is implemented. This knowledge usually enables us to do much more than just performing the operations provided by an abstract group.

Very simple example: For the *additive* group $(\mathbb{Z}/p, +)$, solving the DLP just amounts to *division modulo* p. Doing so requires "inspecting" the internals of the group element, concretely, lifting it from $\mathbb{Z}/p$ to $\mathbb{Z}$ and running the extended Euclidean algorithm.

Key point: The complexity of computing discrete logarithms really depends on the concrete representation of the group, *not just its isomorphism class*. In fact: The DLP in $G \cong \mathbb{Z}/n$ can be understood precisely as *computing* an isomorphism $G \xrightarrow{\sim} (\mathbb{Z}/n, +)$!

So, how far is $\mathbb{F}_p^\times = \big((\mathbb{Z}/p) \backslash \{0\}, \cdot \big)$ from a generic group?

1.4 Index calculus

Key observation: The finite field $\mathbb{F}_p = \mathbb{Z}/p$ is a quotient ring of $\mathbb{Z}$. Integers have many interesting features that abstract groups don't have: Among them are *prime numbers*. The core idea of **index calculus** is to factor elements and then manipulate the *exponents* of that factorization.[3]

[1] Beware: The *computational Diffie–Hellman problem* (CDH), i.e., computing g^{ab} from (g^a, g^b), could actually be easier than computing x from g^x. We will later (in Section 4.9) discuss techniques to argue in specific cases that breaking the key exchange is indeed equivalent to computing logarithms.

[2] Here, essentially, (i, j) is a base-m representation of x, and the algorithm enumerates both digits separately in order to "meet in the middle".

[3] The word "index" here is old-fashioned terminology for "discrete logarithm".

To illustrate the idea, consider the following basic example: Let's say we are trying to solve the DLP instance $g = \overline{27},\ g^x = \overline{243}$ in the finite field $\mathbb{F}_{257}$. By staring at the numbers intently, we notice that in fact $g = \overline{3}^3$ and $g^x = \overline{3}^5$. Hence, we can take a shortcut: The solution is just $x = 5 \cdot 3^{-1} \bmod 256 = 87$. Now, of course, it is *extremely* unlikely that a given DLP instance will be of this form. In general, however, we may try to search for powers of g and g^x that are powers of $\overline{3}$, but this essentially amounts to simply brute-forcing *two* logarithms to base $\overline{3}$, which is probably worse than solving the given DLP directly. However, the very same idea actually works if we consider product decompositions into a larger set of bases than just $\overline{3}$!

Concretely, consider a fixed *factor base* of distinct "small" primes $q_1, \ldots, q_k \neq p$. By a *relation* modulo p with respect to that factor base, we refer to any vector in the kernel of the group homomorphism

$$\varphi\colon \quad (\mathbb{Z}^k, +) \longrightarrow (\mathbb{F}_p^\times, \cdot\,), \quad \vec{v} \longmapsto \prod_{i=1}^{k} \overline{q_i}^{v_i},$$

i.e., any vector $\vec{v} \in \mathbb{Z}^k$ such that $\prod_{i=1}^k q_i^{v_i} \equiv 1 \pmod{p}$. The set of all relations $\ker \varphi$ is the *relation lattice* R; it is generated by k linearly independent vectors. Now, to solve a DLP (g, g^x), we may proceed as follows:

(1) Find a basis of R by repeatedly taking random products of the factor base and hoping that the result, after reduction modulo p, splits over the factor base again. In that case, we have found two (usually distinct) preimages under φ of the same element in $\mathbb{F}_p^\times$, hence their difference must lie in R.
(Note: This depends only on p, hence can be precomputed.)

(2) Find a decomposition $g = \varphi(\vec{v})$ by computing random products of g with the factor base and hoping that the result, after reduction modulo p, splits over the factor base. Similarly, find a decomposition $g^x = \varphi(\vec{w})$.

(3) Using linear algebra modulo $p-1$, compute $x \in \mathbb{Z}$ such that $x\vec{v} - \vec{w} \in R$. Output x as the solution.

Using results on *smoothness probabilities*, one can show that the total cost of the index-calculus algorithm is minimized for $k \approx \exp(const \cdot \sqrt{\log(p)\log\log(p)})$, leading to an overall complexity of

$$\exp\big(O(\sqrt{\log(p)\log\log(p)}\big)\,.$$

This is *subexponential* (but still superpolynomial!) complexity.

5 Remark. The discussion above is optimized for ease of understanding; it is a very crude way of doing index calculus. Proceeding in a much more clever way, as in the *Number Field Sieve*, leads to a heuristic runtime of $\exp\big(\widetilde{O}(\log(p)^{1/\mathbf{3}})\big)$.

(The notation $\widetilde{O}(f(x))$ is shorthand for $f(x) \cdot \log(f(x))^{O(1)}$: Linear in $f(x)$, possibly with additional logarithmic factors.)

6 Remark. The approach readily generalizes to arbitrary number fields by factoring elements into prime *ideals* rather than irreducible elements. This is why simply replacing $\mathbb{F}_p$ by a non-prime finite field does not thwart the attack.

Some numbers. Index calculus is the most effective attack for the DLP in finite fields. Its impact is that the sizes required for p are pretty huge nowadays: BSI's 2024 recommendation says "the length of p should be at least 3000 bits". There is also something called "RSA", which relies on integer factorization being hard-ish; it is equally big and slow.

By comparison, the recommended size for something that behaves more like a generic group is only 250 bits, leading to much faster and much more compact cryptographic constructions. But what could such a "more generic" group be?

1.5 Enter elliptic curves!

At last, now that we've successfully badmouthed the competition, here's the sales pitch:

- Well$_1$-chosen elliptic curves are as *close to generic groups* as it gets, in terms of known attacks.
- Well$_2$-chosen elliptic curves provide *bilinear pairings*, which are very unique and extremely useful.
- Well$_3$-chosen elliptic curves can be used (in a different way) to construct *post-quantum primitives*.

1.6 The "Q"-word...

Sad future: Large-scale quantum computers are expected to break *all* DLP-type problems, once they are actually built. However, that doesn't seem to have happened thus far, and elliptic curves remain the number-one industry standard.

Second lecture (April 22)

2 Elliptic curves, concretely

High-brow version for those who know: An elliptic curve is a smooth projective curve E of genus one with a chosen rational point O. By the Riemann–Roch theorem, the map $E \to \mathrm{Pic}^0(E),\ P \mapsto [P] - [O]$ is bijective, and we can pull back explicit algebraic formulas for the group law from $\mathrm{Pic}^0(E)$ to E, therefore E is an abelian variety. Done.

Rest of this lecture: Pretty much just that, but using significantly less heavy machinery, and with *formulas!*

Throughout, let K be a field.

7 Definition. The *projective space* $\mathbb{P}^n(K)$ of dimension n over K is the set of nonzero points in K^{n+1} modulo scaling by nonzero elements of K. (The base field K is sometimes omitted when it is clear from context or insignificant.) We write $P = (X_0 : X_1 : \ldots : X_n)$ for a point in $\mathbb{P}^n$.

The special cases $n = 1$ and $n = 2$ are known as the *projective line* and *projective plane* respectively. Notation: For $n = 1$ we use the variables (U, V) and for $n = 2$ we use the variables (X, Y, Z). Points with $Z = 0$ are "at infinity".

8 Example. The "point" $(0 : 0 : \ldots : 0)$ is *not* a valid projective point. For all $P = (X_0 : X_1 : \ldots : X_n)$ and all $\lambda \neq 0$ we have $P = (\lambda X_0 : \lambda X_1 : \ldots : \lambda X_n)$. For example, $(0 : 1 : 0) = (0 : -1 : 0)$ and $(1 : 2 : 3) = (3 : 6 : 9)$.

9 Example. One of the reasons for considering projective space is that it gets rid of "annoying special cases" in geometry: For example, on a projective plane, *any* pair of lines intersects in exactly one point, and this is true even for parallel lines! We will see a generalized statement of this sort below when discussing Bézout's theorem.

10 Definition. A polynomial $f \in K[X_0, \ldots, X_n]$ is *homogeneous* if all monomials in f have the same degree.
The data of a *projective plane curve* C consists of a nonconstant homogeneous polynomial $F \in K[X, Y, Z]$.
Points on C are exactly those $P = (X : Y : Z) \in \mathbb{P}^2(\overline{K})$ where $F(X, Y, Z) = 0$.

11 Lemma. *Let $f \in K[X_1, \ldots, X_n]$ of degree $d \geq 0$. Then there exists a unique homogeneous polynomial $F \in K[X_0, \ldots, X_n]$ of degree d such that $F(1, X_1, \ldots, X_n) = f$. (This polynomial F is referred to as the* homogenization *of f.)*

Proof. Multiply each monomial in f by an appropriate power of X_0 to make the degree equal d. □

This lemma allows us to specify a projective plane curve by a bivariate (not necessarily homogeneous) polynomial in $K[x, y]$ rather than a trivariate homogeneous polynomial in $K[X, Y, Z]$. We will often do this for ease of notation.

When working with elliptic curves concretely, it is extremely convenient to restrict to a particular shape of equation:

12 Definition. A *(short) Weierstraß curve* is a projective plane curve given by an equation of the form

$$y^2 = x^3 + ax + b$$

where $a, b \in K$.

Similarly, a *long Weierstraß curve* is a projective plane curve given by an equation of the form

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

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

13 Lemma. *If the characteristic of the base field is $\notin \{2, 3\}$, every long Weierstraß curve is isomorphic to a short Weierstraß curve.*

14 Lemma. *Let E be a (short or long) Weierstraß curve. Then E has exactly one point at infinity, and it is given by $\infty := (0 : 1 : 0)$.*

Proof. The homogenization is $Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$. Substituting $Z = 0$ leaves us with $X^3 = 0$, hence $X = 0$, and no condition on Y. Therefore $(0 : 1 : 0) \in \mathbb{P}^2$ is the unique solution. □

15 Definition. Let C be a projective plane curve defined by $F \in K[X, Y, Z]$. We say that C is *nonsingular at* $P \in C$ if $\left(\frac{\partial F}{\partial X}(P), \frac{\partial F}{\partial Y}(P), \frac{\partial F}{\partial Z}(P)\right) \neq (0, 0, 0)$. If C is nonsingular at all points, it is called *nonsingular* or *smooth*.

16 Example. The point $\infty = (0 : 1 : 0)$ on a Weierstraß curve is always nonsingular.

The Weierstraß curves $y^2 = x^3$ and $y^2 = x^3 - x^2$ both have a singularity (only) at the point $(0 : 0 : 1)$.

17 Theorem. *Any elliptic curve in the "high-brow" sense (smooth projective curve of genus one with a rational point) is isomorphic to a long Weierstraß curve. Conversely, every smooth (short or long) Weierstraß curve is an elliptic curve.*

18 Lemma. *A short Weierstraß curve is nonsingular if and only if the* discriminant $-16(4a^3 + 27b^2)$ *is nonzero. (The corresponding formula for long Weierstraß curves exists, but is omitted here for its bulkiness.)*

19 Remark. In cryptography, we often use more specialized curve forms for speed or other beneficial properties: There will be examples of this later. Still, the Weierstraß equation remains a very useful "standard form" for elliptic curves.

2.1 The group law

The low-brow version of the elliptic-curve group law is to write down explicit "addition formulas" which were figured out by someone else at some point, then prove in a tedious case-by-case analysis that they define an abelian group on E. In the following, we will consider an intermediate viewpoint, motivating the group law by its inherent geometric meaning, while still staying away from advanced, more general notions of algebraic geometry.

The starting point is the following classical result:

20 Bézout's Theorem. Let C_1 and C_2 be projective plane curves defined by $F_1, F_2 \in K[X, Y, Z]$ of degrees d_1, d_2. If F_1 and F_2 have no nonconstant common factor, then there are exactly $d_1 \cdot d_2$ intersection points of C_1 and C_2, counted with multiplicities.

One key consequence of this theorem for cubics is that given two points P, Q on a cubic curve, one can find a third point R by drawing a straight line (a degree-one curve) through P and Q and computing the last point of intersection. This is indeed what lies at the heart of the elliptic-curve group law for Weierstraß curves. To make this precise for special cases such as $P = Q$, we need some definitions:

21 Lemma. *Every line L in $\mathbb{P}^2$ can be given in* parametric form *by an embedding*

$$\begin{aligned}\mathbb{P}^1(K) &\hookrightarrow \mathbb{P}^2(K),\\ (U:V) &\longmapsto (H_X(U,V), H_Y(U,V), H_Z(U,V))\end{aligned}$$

where H_X, H_Y, H_Z are homogeneous polynomials of degree one.

Proof. Lines are projective plane curves of degree one, so L is given by a nonzero polynomial of the form $aX + bY + cZ$. Suppose (w.l.o.g.) that $a \neq 0$. Then $(H_X, H_Y, H_Z) = (-b/a \cdot U - c/a \cdot V, U, V)$ is a valid parameterization. □

22 Definition. Let C be a projective plane curve defined by a homogeneous polynomial $F \in K[X, Y, Z]$. Let L be a line given in parametric form (H_X, H_Y, H_Z) as in the lemma. For a point $P \in C \cap L$, pick $u_0, v_0 \in K$ such that $P = (H_X(u_0, v_0) : H_Y(u_0, v_0) : H_Z(u_0, v_0))$. The *intersection multiplicity* (or *order of intersection*) of C and L at P is the largest positive integer k such that $(v_0U - u_0V)^k$ divides $F(H_X, H_Y, H_Z) \in K[U, V]$.

23 Remark. This definition generalizes the usual concept of the multiplicity of a polynomial root, but the latter hardcodes the line L to be the x-axis. In the more general setting considered here, we allow arbitrary lines, and we work projectively to also allow apparent special cases like vertical lines (slope ∞).

24 Definition. For a projective plane curve C, a *tangent* at $P \in C$ is a line intersecting C at P with multiplicity ≥ 2.

25 Lemma. *The tangent of a Weierstraß curve E at ∞ is the "line at infinity" defined by $Z = 0$. It intersects with multiplicity 3.*

Proof. The tangent can be parameterized by $\mathbb{P}^1$ as $(U : V : 0)$, and ∞ is reached at $(u_0 : v_0) = (0 : 1)$. Substituting into the Weierstraß equation, we get $-U^3$, which is clearly divisible by $(v_0U - u_0V) = U$ exactly three times. □

26 Definition. Let E be a Weierstraß elliptic curve. Ad-hoc notation: For two points $P, Q \in E$, let $P * Q$ be the third point of intersection of a line through P and Q with E. Now, define a binary operation $+$ on E as follows:

$$P + Q := \infty * (P * Q)\,.$$

27 Sublemma. *For all points $P, Q \in E$, we have $P * (P * Q) = Q$.*

Proof. The line L through P and Q intersects at P, Q, and $P * Q$. The line L' through P and $P * Q$ intersects at P, $P * Q$, and a third point R. However, any two lines sharing two points must be identical, hence $L = L'$, and it follows that $R = Q$. □

28 Lemma. *This $+$ is a commutative binary operation with neutral element ∞. Moreover, $P + (\infty * P) = \infty$ for all $P \in E$.*

Proof. Commutativity is obvious: The line through P and Q clearly equals the line through Q and P, hence $P * Q = Q * P$, and all the other steps only depend on $P * Q$.

For the neutral element, the sublemma implies $\infty + P = \infty * (\infty * P) = P$.

Similarly, for the inverse elements, the sublemma yields $P + (\infty * P) = \infty * (P * (\infty * P)) = \infty * \infty$, which equals ∞ since the tangent line at ∞ intersects E with multiplicity 3. □

29 Remark. Associativity is significantly less obvious. (But it does hold!)

2.1.1 Explicit formulas

To summarize, we have the following result:

30 Theorem. *Let E be a Weierstraß elliptic curve. There exists a unique group law $+\colon E \times E \to E$ with neutral element ∞ and defined by the property that $P+Q+R=\infty$ holds if and only if P, Q, R lie on a line intersecting E at P, Q, R with the correct multiplicities.*

From this general definition, we can derive concrete formulas, which we shall do now. Note that we commonly write points $(X : Y : Z) \neq \infty$ using their *affine* coordinates (x, y), which are given by $x = X/Z$ and $y = Y/Z$.

31 Lemma. *For simplicity, consider the case of E being a short Weierstraß curve over a field of "large" characteristic, i.e., ≥ 5. (Everything works similarly for long Weierstraß curves and general characteristic.)*

(1) The negative of a point (x, y) is $(x, -y)$.

(2) The sum of two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ with $Q \neq -P$ is the point (x_3, y_3) where

$$x_3 = \lambda^2 - x_1 - x_2\,;$$
$$y_3 = \lambda(x_1 - x_3) - y_1\,.$$

Here λ is the slope of the line through P and Q; it is given by $\lambda = \frac{y_2-y_1}{x_2-x_1}$ when $x_2 \neq x_1$ and $\lambda = \frac{3x_1^2+a}{2y_1}$ otherwise.

Proof. (Homework.) □

Third lecture (April 29)

3 The number of points on an elliptic curve

Last time, we've seen how to construct a group structure on the group of points on an elliptic curve E over a field K. Since the explicit formulas are rational maps, it is immediately clear that $E(L)$ forms a subgroup of E for any field $L \supseteq K$.

In the following, we will consider the case that L is a finite field $\mathbb{F}_q$: Then, since $|\mathbb{P}^2(\mathbb{F}_q)| = q^2 + 1$ is finite, the group $E(\mathbb{F}_q) \subseteq \mathbb{P}^2(\mathbb{F}_q)$ is clearly finite as well. Notation: In this context, it is common to write $\#E(\mathbb{F}_q)$ for $|E(\mathbb{F}_q)|$.

Question for the coming lectures: What is the group structure of $E(\mathbb{F}_q)$, how does it impact the difficulty of the *elliptic-curve discrete logarithm problem* (ECDLP), and how can we make sure to choose a secure one?

3.1 The Pohlig–Hellman algorithm

Recall (first lecture):

- The *baby-step giant-step algorithm* solves the DLP in any group of order n in $O(\sqrt{n})$ steps.
- One can prove that this is essentially optimal for generic prime-order groups.

$\rightsquigarrow$ What about generic *composite-order* groups? Those are where the *Pohlig–Hellman algorithm* is advantageous.

First, notice that the ECDLP $(P, [x]P)$ only "lives" in the cyclic subgroup $\langle P \rangle$ anyway, hence we may immediately restrict our attention to cyclic groups. The algorithm crucially relies on the following decomposition, which allows us to reduce DLP instances to the case of cyclic groups of prime-power order:

32 Lemma. *Let E be a cyclic group of order $n \in \mathbb{Z}_{\geq 1}$. Factorize $n = \ell_1^{m_1} \cdots \ell_r^{m_r}$. Define the "projection maps"*

$$\pi_i\colon E \to E,\ P \mapsto [n/\ell_i^{m_i}]P\,.$$

The image G_i of each π_i is a (necessarily cyclic) subgroup of order $\ell_i^{m_i}$, and the collection of all π_i defines an isomorphism

$$E \xrightarrow{\sim} E_1 \times \cdots \times E_r\,.$$

Proof. Clearly the π_i are group homomorphisms. Domain and codomain have the same cardinality, hence it is enough to prove injectivity: Let $P \in E$ with all $\pi_i(P) = \infty$. For a generator $G \in E$, there exists a $k \in \mathbb{Z}$ with $P = [k]G$. Hence all $\pi_i([k]G) = [n/\ell_i^{m_i}][k]G = [nk/\ell_i^{m_i}]G = \infty$, implying that n divides $nk/\ell_i^{m_i}$ since G has order n. But this implies $\ell_i^{m_i} \mid k$. Since the $\ell_i^{m_i}$ are coprime, their least common multiple equals n, which must thus divide k as well. Hence $n \mid k$ and $P = [k]G = \infty$. □

To simplify a composite-order DLP (P, Q) using this, all we have to do is to (for all i) compute π_i, solve the DLP in E_i, and finally combine the results by solving a system of congruences modulo the $\ell_i^{m_i}$ using the explicit Chinese Remainder Theorem (CRT): The congruences are $x \equiv x_i \pmod{\ell_i^{m_i}}$ where x_i is a solution to the DLP $(\pi_i(P), \pi_i(Q))$.

33 Remark. In fact, the CRT is itself very similar to the isomorphism from the lemma when applied to $\mathbb{Z}/n$: The only difference lies in some normalization factors applied to each projection map to turn the isomorphism into a ring isomorphism rather than just a group isomorphism.

What's left to do is solving the DLPs in prime-power-order groups, which we'll do efficiently using a recursive method:

34 Subroutine (prime-power DLP): Given a DLP instance (P, Q) where P has order ℓ^k, proceed as follows: Solve the order-ℓ DLP with input $\big([\ell^{k-1}]P, [\ell^{k-1}]Q\big)$, yielding a result $x_0 \in \{0, ..., \ell - 1\}$. Set $P' := [\ell]P$ and $Q' := Q - [x_0]P$. Compute the solution $x_1 \in \{0, ..., \ell^{k-1} - 1\}$ of the order-ℓ^{k-1} DLP with input (P', Q'). Finally output $x_0 + \ell x_1$.

35 Lemma. *The prime-power DLP routine above is correct and runs in time $O(k\sqrt{\ell} + k^2 \log \ell)$.*

Proof. Consider the base-ℓ expansion $s = \sum_{i=0}^{k-1} s_i \ell^i$ of the actual solution $s \in \{0, ..., \ell^k - 1\}$. Then

$$[\ell^{k-1}]Q = [\ell^{k-1}][s]P = [\ell^{k-1} s_0]P$$

since the "exponents" in brackets can be taken modulo ℓ^k, so $x_0 = s_0$. Thus, we also get

$$Q' = [s]P - [s_0]P = \Big[\sum_{i=1}^{k} s_i \ell^i\Big]P = \Big[\sum_{i=1}^{k} s_i \ell^{i-1}\Big]P' .$$

Hence $x_1 = \sum_{i=1}^{k} s_i \ell^{i-1}$ and we conclude $x_0 + \ell x_1 = s$.

For the runtime, observe that every order-ℓ DLP takes time $O(\sqrt{\ell})$ using BSGS. The scalar multiplications take time $O(k \log \ell)$ each. Since there are k recursive calls in total, the runtime is $O\big(k(\sqrt{\ell} + k^2 \log \ell)\big)$ as claimed. □

36 Remark. When n is a product of a fixed set of primes, the runtime is therefore only $O((\log n)^2)$: Exponentially faster than running BSGS on the same DLP instance, which costs $O(\sqrt{n})$. However, the Pohlig–Hellman algorithm helps reduce the cost essentially always when n is composite, even if the prime factors are big.

Key takeaway: The hardness of DLP for generic-group algorithms comes from the largest prime factor of the order.

3.2 Baseline: Counting points by literal counting

As a first step, let's find an estimate for the number of points: Morally, since curves are one-dimensonal, the number of points ought to be on the order of q. We will see later that this is indeed true in a strict sense, but first, here's a heuristic:

37 Lemma. *Let* $E\colon y^2 = x^3 + ax + b$*. Then*

$$\#E(\mathbb{F}_q) = 1 + \sum_{x \in \mathbb{F}_q} \big(1 + \chi(x^3 + ax + b)\big)$$

where χ *is the quadratic character defined by*

$$\chi(\alpha) = \begin{cases} +1 & \text{if } \alpha \text{ is a nonzero square in } \mathbb{F}_q; \\ -1 & \text{if } \alpha \text{ is not a square in } \mathbb{F}_q; \\ 0 & \text{if } \alpha = 0. \end{cases}$$

Proof. Every x where $x^3 + ax + b$ is a nonzero square y^2 gives rise to two points $(x, \pm y) \in E(\mathbb{F}_q)$, every x where $x^3 + ax + b$ is a non-square gives rise to zero points in $E(\mathbb{F}_q)$, and every x where $x^3 + ax + b$ is zero gives rise to the single point $(x, 0) \in E(\mathbb{F}_q)$. □

38 Remark. This can easily be generalized to long Weierstraß curves, and hence to arbitrary characteristic, by considering the discriminant of the quadratic equation in y resulting from fixing x.

39 Corollary. *The number of points is bounded by* $1 \leq \#E(\mathbb{F}_q) \leq 2q + 1$.

40 Heuristic. Assume that q is odd. Since the squaring map $\alpha \mapsto \alpha^2$ is two-to-one on $\mathbb{F}_q^\times$, there are exactly $(q-1)/2$ squares in $\mathbb{F}_q$. Hence, if every $x^3 + ax + b$ value was uniformly random in $\mathbb{F}_q$, the expected value of the sum would amount to $1 + q \cdot \big(1 + \frac{q-1}{2q} \cdot 1 + \frac{1}{q} \cdot 0 + \frac{q-1}{2q} \cdot (-1)\big) = q + 1$, in line with the fact that *curves are one-dimensional*.

In reality, the values $x^3 + ax + b$ are of course not entirely random, but this reasoning still gives a reasonable approximation for the number of points. We will give a precise result in Section 3.7.

41 Remark. Simply evaluating the sum in the lemma takes time $\widetilde{O}(q)$. We can do *much* better than that.

3.3 Counting points using generic-group algorithms

In this section, we will approach the counting problem from the angle of generic-group algorithms. For this purpose, assume for the moment that the group in question is cyclic of order n. Elliptic-curve groups are generally *not* cyclic, but there are good reasons for considering this special case first: (1) it is easier, (2) it often suffices for cryptography, and (3) the resulting insights are needed for the general case, anyway.

42 Remark. Since the ECDLP $(P, [x]P)$ only "lives" in the cyclic subgroup $\langle P \rangle$ anyway, it is useful to consider the problem of computing the order of a given element instead: We have to find the smallest $k \in \mathbb{Z}_{\geq 1}$ such that $[k]P = \infty$.

43 Remark. By Lagrange's theorem, element orders must be divisors of the group order. Hence, for cyclic groups, the exact order can be found with high confidence by repeatedly computing the order of a random element and outputting the least common multiple of the found orders. (For general abelian groups, the same procedure computes the *exponent* instead of the order.)

- If nothing is known about the actual order of the group, this is a "Monte Carlo" algorithm: Deterministic runtime, but there is a nonzero chance that the result is wrong.
- If sufficiently restrictive a priori bounds on the group order are known, it turns into a "Las Vegas" algorithm: Probabilistic runtime, but the result is guaranteed to be correct.

Now, notice how computing the order of an element is essentially solving the ECDLP (P, ∞) with the additional constraint that the smallest positive solution is required. We can thus start with the baby-step giant-step algorithm:[4]

[4] Historical detail: This application is in fact the original use case for the BSGS algorithm. Shanks used it in 1971 to compute class numbers.

44 Subroutine (BSGS): Given a point $P \in E(\mathbb{F}_q)$, set $m := \lceil\sqrt{2q+1}\rceil$, make a table which maps $[i][m]P \mapsto i$ for all $i \in \{0, ..., m-1\}$, and search for $j \in \{1, ..., m\}$ such that $-[j]P$ is in the table, giving an integer i. Then return $mi + j$.

This procedure returns some positive integer k such that $[k]P = \infty$. However, this k may not be minimal, which would lead to wrong conclusions later. To fix the issue, we can use the following algorithm:

45 Subroutine (order finding): Given a point P and $k \in \mathbb{Z}_{\geq 1}$ such that $[k]P = \infty$, compute the prime factorization $\ell_1^{m_1} \cdots \ell_r^{m_r}$ of k. For each $i \in \{1, ..., r\}$, find the largest integer f_i such that $\left[k/\ell_i^{f_i}\right]P = \infty$. Output $k/\prod_{i=1}^{r} \ell_i^{f_i}$.

46 Lemma. *The order-finding routine above is correct. Its runtime is the cost of factoring k plus $\operatorname{polylog}(k)$ group operations.*

Proof. The actual element order divides all $k/\ell_i^{f_i}$ by construction. The output $k/\prod \ell_i^{f_i}$ is their greatest common divisor. If any ℓ_i could be divided out more times, then f_i would have had to be larger since $k/\ell_i^{f_i}$ is a multiple of $k/\prod \ell_i^{f_i}$. □

Combining the algorithms, we therefore have a fully generic order-computation algorithm for cyclic groups. Its runtime in group operations scales as the square root of the group size.

47 Remark. There is a major issue when trying to use this approach in the context of elliptic-curve cryptography: The expected runtime is essentially the same as the effort to break the ECDLP using BSGS even if the curve turns out to be a worst-case choice for BSGS. We therefore need a signicicantly faster way to count points, similar to how users require a way of computing $[x]G$ much faster than simply adding G to itself x times (see Section 1.2). This approach will have to go beyond generic groups and use a lot more specific knowledge about elliptic curves.

3.4 Division polynomials

Let K be a field of characteristic 0 and suppose the short Weierstraß curve $y^2 = x^3 + ax + b$ with $a, b \in K$ is an elliptic curve, i.e., that $4a^3 + 27b^2 \neq 0$. Consider the weighted polynomial ring $K[x, y]_w$ with weights $\deg(x) = 2$ and $\deg(y) = 3$ and define polynomials $\psi_m \in K[x, y]_w$ recursively by

$$\begin{aligned}
\psi_1 &= 1\,; \\
\psi_2 &= 2y\,; \\
\psi_3 &= 3x^4 + 6ax^2 + 12bx - a^2\,; \\
\psi_4 &= 4y \cdot (x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)\,; \\
\psi_m &= \psi_{k+2}\psi_k^3 - \psi_{k-1}\psi_{k+1}^3 && \text{for } m = 2k+1 \geq 5\,; \\
\psi_2\psi_m &= (\psi_{k-1}^2\psi_{k+2} - \psi_{k-2}\psi_{k+1}^2)\psi_k && \text{for } m = 2k+0 \geq 6\,.
\end{aligned}$$

The significance of the division polynomials lies in the fact that they manage to essentially pinpoint the kernel of the multiplication-by-m map on the elliptic curve:

48 Theorem. *Let $P \in E$. Then $[m]P = \infty$ if and only if $P = \infty$ or $P = (x, y)$ and $\psi_m(x, y) = 0$.*

Proof sketch. There are essentially two strategies:

- Lifting the elliptic curve to the complex numbers and parameterizing it by the Weierstraß $\wp$ function, with which these identities are more easily verified.
- Painfully calculating everything algebraically using the addition formulas for the elliptic curve. The advantage is that this will work in any characteristic (using long Weierstraß curves).

49 Lemma. *We have $\psi_m \in K[x, y^2]$ if m is odd and $\psi_m \in yK[x, y^2]$ if m is even.*

Proof. (Homework.) □

50 Remark. This means we can substitute y^2 by x^3+ax+b in these polynomials and obtain polynomials in x only, possibly multiplied by a single power of y. In the following, we will use the convention that $\psi_m \in K[x]$ or $\psi_m \in yK[x]$.

51 Remark. This also reveals the motivation behind the (a priori perhaps strange-looking) choice of weights: It implies $\deg(y^2) = \deg(x^3 + ax + b)$.

52 Lemma. *Recall $\deg(x) = 2$ and $\deg(y) = 3$. Then $\deg(\psi_m) = m^2 - 1$ and the leading coefficient[5] of ψ_m equals m.*

Proof. (Homework.) □

53 Remark. Since the recursive formulas are purely algebraic, one can also keep the values a, b symbolic and thus obtain "generic" division polynomials which will be valid for any short Weierstraß curve after substituting the coefficients.

Fourth lecture (May 6)

In addition to characterizing the m-torsion points, division polynomials actually allow us to easily express the scalar-multiplication maps as morphisms of algebraic curves:

54 Theorem. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over a field K of characteristic $\notin \{2, 3\}$. Define*

$$\phi_m := x\psi_m^2 - \psi_{m+1}\psi_{m-1}\,;$$
$$4y\omega_m := \psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2\,.$$

where we again substitute y^2 by $x^3 + ax + b$ in ϕ_m and ω_m. Then, for all $P = (x, y) \in E$,

$$[m]P = \left(\frac{\phi_m}{\psi_m^2}(x), \frac{\omega_m}{\psi_m^3}(x, y)\right).$$

3.5 The group structure

55 Definition. Let E be an elliptic curve and $n \geq 1$. The *n-torsion subgroup* of E is

$$E[n] = \ker[n] = \{P \in E : [n]P = \infty\}\,.$$

Similarly, the *$\mathbb{F}_q$-rational* n-torsion subgroup is simply

$$E(\mathbb{F}_q)[n] := E(\mathbb{F}_q) \cap E[n]\,.$$

As a simple corollary to the main theorem about division polynomials, we get two structure results:

56 Corollary. *Let E be an elliptic curve over a field. Then $|E[n]| \leq n^2$.*

Proof. As usual, only for short Weierstraß curves: It suffices to show that there are no more than $n^2 - 1$ distinct pairs $(x, y) \in E$ such that $\psi_n(x, y) = 0$. From Section 3.4, we have $\deg \psi_n = n^2 - 1$ and either $\psi_n \in K[x, y^2]$ or $\psi_n \in yK[x, y^2]$. To unify the two cases and simplify notation we consider ψ_n^2 as a univariate polynomial in x: On the curve E, this polynomial vanishes at the same inputs as ψ_n, but with double multiplicity each time. The *unweighted* degree of ψ_n^2 is precisely $n^2 - 1$, hence there are at most $(n^2 - 1)/2$ distinct roots over the closure, and every one of them gives rise to at most two points. □

[5] In principle, the concept of leading coefficient is not well-defined in multivariate polynomial rings. To us, here, it refers to the coefficient attached to the unique monomial of maximal degree after replacing all occurrences of y^2 by $x^3 + ax + b$. Equivalently, this value equals the sum of all coefficients attached to monomials of maximal degree.

57 Corollary. *Let E be an elliptic curve over a field K of characteristic $\neq \{2,3\}$ and let $n \geq 1$. Then there are positive integers $n_1 \mid n_2 \mid n$ such that*

$$E[n] \;\cong\; \mathbb{Z}/n_1 \times \mathbb{Z}/n_2\,.$$

Proof. By the classification of finite abelian groups, the only thing to prove is that the rank cannot be greater than two. But if there exists an injective group homomorphism $\mathbb{Z}/n_1 \times \mathbb{Z}/n_2 \times \mathbb{Z}/n_3 \hookrightarrow E[n]$ with $2 \leq n_1 \mid n_2 \mid n_3 \mid n$, then with $\ell := \gcd(n_1, n_2, n_3) \geq 2$ we get $(\mathbb{Z}/\ell)^3 \hookrightarrow E[\ell]$, which contradicts the fact that $|E[\ell]| \leq \ell^2$. □

In fact, a much stronger result is known about the structure of n-torsion subgroups on elliptic curves:

58 Theorem. *Let E be an elliptic curve over a field K and let $m \geq 1$. If $\mathrm{char}(K) = p > 0$, write $m = p^r \cdot m'$ with $p \nmid m'$; otherwise, let $m' = m$. Then*

$$E[m] \;\cong\; \mathbb{Z}/m \times \mathbb{Z}/m' \quad \textit{or} \quad E[m] \;\cong\; \mathbb{Z}/m' \times \mathbb{Z}/m'\,.$$

Proof sketch. It suffices to prove this for prime powers m: The general case follows from it using the classification of finite abelian groups. For $m = \ell^k > 1$, the a priori possible group structures are $\mathbb{Z}/\ell^{k_1} \times \mathbb{Z}/\ell^{k_2}$ where $0 \leq k_1 \leq k_2 \leq k$. Then, two cases are distinguished:

- If $\ell = p$, then the (weighted) degree of ψ_m modulo p becomes smaller than $m^2 - 1$ since the leading coefficient of ψ_m was shown to be $p^k \equiv 0 \pmod{p}$. Hence $k_1 = k_2 = k$ is impossible. Since this also holds true for $k = 1$, there cannot be a subgroup of $E[p^k]$ isomorphic to $\mathbb{Z}/p \times \mathbb{Z}/p$ and we must have $k_1 = 0$. Now, the tricky part is to prove that $k_2 \in \{0, k\}$: This follows from the (non-obvious and rather technical) fact that $\psi_p \in K[x, y]$ is either a nonzero constant in K or of the form $h(x^p, y^p)$ where h has distinct roots.
- In the other case $\ell \neq p$, the (weighted) degree of ψ_m remains $m^2 - 1$ modulo p. Here the entire proof rests on the (again non-obvious but technical) fact that the roots of the m-division polynomial are all distinct if $m \neq 0 \in K$.

59 Definition. An elliptic curve E over a field of characteristic $p{>}0$ is called *supersingular*[6] if $E[p] = \{\infty\}$, else *ordinary*.

3.6 Isogenies and endomorphisms

All(?) efficient point-counting algorithms rely heavily on properties of the *Frobenius endomorphism*, a special type of elliptic-curve morphism.

60 Definition. Let E, E' be two elliptic curves over a field K. An *elliptic-curve homomorphism* is a map $\varphi\colon E \to E'$ which is (1) given by rational maps, and (2) a group homomorphism. As usual, an *isomorphism* is an invertible homomorphism, and an *endomorphism* of E is a homomorphism from E to itself. An *isogeny* is a non-constant elliptic-curve homomorphism.

Concretely, for a Weierstraß equation in $\mathbb{P}^2(K)$, the data of a homomorphism consists of nonzero homogeneous polynomials $F, G, H \in \overline{K}[X, Y, Z]$ of the same degree, such that the map

$$(x : y : z) \mapsto (F(x, y, z) : G(x, y, z) : H(x, y, z))$$

takes points on E to points on E'. Note that it can happen that F, G, H vanish simultaneously at some points P. In that case, it is always possible to replace F, G, H by another set of defining polynomials which do not all vanish at P, while producing the same map wherever defined. (Thus, the map defined by F, G, H extends uniquely to all of E.)

[6] Beware: The word "supersingular" is an old-fashioned way of saying "very special", referring to the fact that these curves are relatively rare. It has nothing to do with singular points — all elliptic curves are nonsingular by definition.

61 Example. An example are the homogenizations of the polynomials $F = \phi_m\psi_m, G = \omega_m, H = \psi_m^3$ from Section 3.4.

62 Definition. Let E be an elliptic curve over a finite field $\mathbb{F}_q$ of characteristic $p > 0$. Its *(p-power) Frobenius isogeny* is

$$\pi_p \colon E \to E^{(p)},\ (x, y) \mapsto (x^p, y^p)$$

where $E^{(p)}$ is the curve defined by applying the map $a \mapsto a^p$ to all coefficients of the equations defining E.

The *(q-power) Frobenius endomorphism* is

$$\pi_q \colon E \to E,\ (x, y) \mapsto (x^q, y^q)\,.$$

63 Definition. We say that an isogeny $\varphi \colon E \to E'$ is *separable* if it does not factor through a Frobenius isogeny. Otherwise, it is *inseparable*. An isogeny that consists of only a Frobenius composed with isomorphisms is *purely inseparable*.

64 Definition. We define the *degree* of an elliptic-curve homomorphism as follows:

- The constant zero map has degree 0.
- For a separable isogeny, the degree is cardinality of its kernel.
- The degree of a p^r-power Frobenius isogeny is p^r.
- Composing isogenies multiplies the degrees.

65 Remark. The "correct" definition of the degree is the degree as a morphism of varieties. For algorithmic applications, the explicit version here is often more convenient.

66 Theorem. *For every elliptic-curve homomorphism $\varphi \colon E \to E'$, there exists a unique* dual $\widehat{\varphi} \colon E' \to E$ *defined by the property*

$$\widehat{\varphi} \circ \varphi = [\deg \varphi] \colon E \to E \quad \textit{and} \quad \varphi \circ \widehat{\varphi} = [\deg \varphi] \colon E' \to E'\,.$$

67 Remark. Homomorphisms running between the same pair of elliptic curves can be added and subtracted pointwise.[7] We let $\mathrm{Hom}(E, E')$ denote the group of elliptic-curve homomorphisms $E \to E'$. Similarly, $\mathrm{End}(E) := \mathrm{Hom}(E, E)$ forms a (not necessarily commutative) ring with composition of homomorphisms as multiplication.

68 Lemma. *The map $\mathbb{Z} \to \mathrm{End}(E)$ given by sending an integer m to the multiplication-by-m map $[m] \colon E \to E$ is an injective ring homomorphism.*

69 Lemma. *For any elliptic-curve homomorphisms $\varphi, \psi \colon E \to E'$ we have the following properties:*

- $\widehat{\widehat{\varphi}} = \varphi$.
- $\widehat{\varphi + \psi} = \widehat{\varphi} + \widehat{\psi}$.
- $\widehat{\varphi \circ \psi} = \widehat{\psi} \circ \widehat{\varphi}$.

70 Definition. For an elliptic-curve endomorphism ϑ, let the *trace* of ϑ be the unique integer t satisfying $\vartheta + \widehat{\vartheta} = [t]$.

71 Lemma. *Every elliptic-curve endomorphism ϑ satisfies its* characteristic polynomial

$$\vartheta^2 - [\mathrm{tr}\,\vartheta]\vartheta + [\deg \vartheta] = [0]\,.$$

Proof. (Homework.) □

[7] It follows from the addition formulas that this is well-defined.

3.7 Frobenius & the Hasse bound

Using the machinery just introduced, we will now relate the Frobenius endomorphism to the number of points on an elliptic curve over a finite field and use it to deduce a much tighter bound than the obvious $1 \leq \#E(\mathbb{F}_q) \leq 2q+1$.

72 Lemma. *Let E be an elliptic curve defined over $\mathbb{F}_q$. Then $E(\mathbb{F}_q) = \ker\big(\pi_q - [1]\big)$.*

Proof. An element $\alpha \in \overline{\mathbb{F}_q}$ satisfies $\alpha^q = \alpha$ if and only if $\alpha \in \mathbb{F}_q$. Hence, for $P = (x,y) \in E$, we get $(\pi_q - [1])(P) = \infty$ if and only if $\pi_q(P) = P$ if and only if $(x^q, y^q) = (x,y)$ if and only if $P \in E(\mathbb{F}_q)$. □

In order to use the fact that $E(\mathbb{F}_q) = \ker(\pi_q - [1])$ to count points, we need a tiny technical lemma:

73 Lemma. *For $E/\mathbb{F}_q$, the endomorphism $\pi_q - [1]$ is separable.*

74 Corollary. *For $E/\mathbb{F}_q$, we have $\#E(\mathbb{F}_q) = q + 1 - \operatorname{tr}\pi_q$.*

Proof. By the two lemmata, we have $\#E(\mathbb{F}_q) = |\ker(\pi_q - [1])| = \deg(\pi_q - [1])$. This quantity equals

$$\deg(\pi_q - [1]) = (\widehat{\pi_q} - [1])(\pi_q - [1]) = \widehat{\pi_q}\pi_q - \widehat{\pi_q} - \pi_q + 1 = [\deg \pi_q + 1 - \operatorname{tr}\pi_q]\,. \qquad \square$$

75 Hasse's Theorem. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then

$$q + 1 - 2\sqrt{q} \;\leq\; \#E(\mathbb{F}_q) \;\leq\; q + 1 + 2\sqrt{q}\,.$$

Proof. It remains to show that $|\operatorname{tr}(\pi_q)| \leq 2\sqrt{q}$. To do so, we write $t := \operatorname{tr}\pi_q$ and compute the (necessarily non-negative) degree of the endomorphism $[2q] - [t]\pi_q$:

$$0 \leq \deg([2q] - [t]\pi_q) = ([2q] - [t]\widehat{\pi}_q)([2q] - [t]\pi_q) = [4q^2] - [2qt](\widehat{\pi}_q + \pi_q) + [t^2 q] = [4q^2 - t^2 q]\,.$$

Dividing by q on both sides then reveals $t^2 \leq 4q$, which was to be shown. □

When we are dealing with a curve defined over a small field but are interested in the point count over a larger field, there is a neat shortcut:

76 Lemma. *Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and suppose $\#E(\mathbb{F}_q) = q + 1 - t$. Define the recursive sequence $t_0 := 2$, $t_1 := t$, and $t_k = t \cdot t_{k-1} - q \cdot t_{k-2}$ for $k \geq 2$. Then $\#E(\mathbb{F}_{q^k}) = q^k + 1 - t_k$ for all $k \geq 1$.*

Proof. (Homework.) □

Fourth lecture (May 13)

77 Remark. Given the preceding result, it may seem tempting to use elliptic curves defined over fields of very small characteristic but with points over some extension field that's large enough for cryptography. Such curves are known as "subfield curves". Example: Curve $E/\mathbb{F}_2$, points in $E(\mathbb{F}_{2^{256}})$. There are two issues with this — one minor, one major:

- Any curve as in the lemma necessarily has a composite number of points, since $E(\mathbb{F}_q) \leq E(\mathbb{F}_{q^k})$. However, there may still be a subgroup of $E(\mathbb{F}_{q^k})$ which does not contain points from any subfield and has large prime order, which is good enough for cryptography.
- Curves in high-degree finite fields (and subfield curves in particular) are generally considered risky, since it appears possible that index-calculus techniques may apply to such curves after all, which would reduce the security level very significantly when compared to prime fields of similar sizes.

3.8 Counting points in polynomial time (Schoof/SEA)

Finally, we can discuss the big ideas behind Schoof's celebrated 1985 algorithm to count points over finite fields of large characteristic in polynomial time. There are two key ideas:

- The functional equation $\pi_q^2 - [\operatorname{tr} \pi_q]\pi_q + [\deg \pi_q] = [0]$ can be tested by substituting points into it.
- Points of any given sufficiently small order can be constructed with the help of the division polynomials.

78 Lemma. *Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and let $P \in E$ be a point of some order $\ell \in \mathbb{Z}_{\geq 1}$. Then $\pi_q^2(P) - [k]\pi_q(P) + [q]P = \infty$ holds if and only if $k \equiv \operatorname{tr} \pi_q \pmod{\ell}$.*

Proof. The "if" part is clear since $\pi_q^2 - [\operatorname{tr} \pi_q]\pi_q + [\deg \pi_q] = [0]$ as maps. For the other direction, suppose given a point P and integer k with $\pi_q^2(P) = [k]\pi_q(P) - [\deg \pi_q]P$. Subtracting the correct functional equation yields $\infty = [k - \operatorname{tr} \pi_q]\pi_q(P)$, hence $\operatorname{ord}(\pi_q(P)) = \operatorname{ord}(P) = \ell$ must be a divisor of $k - \operatorname{tr} \pi_q$. □

A very simple point-counting algorithm would now iterate over small primes ℓ, construct a field extension where the division polynomial has a root so that a point of order ℓ can be constructed, iterate over all the ℓ possible choices of k until the value of $\operatorname{tr}(\pi_q) \bmod \ell$ was recovered, then finally compute $\operatorname{tr}(\pi_q)$ as an integer using CRT. This is already *randomized* polynomial-time, but there is a neat trick that renders the algorithm *deterministic* polynomial-time and practically saves plenty of time on factoring polynomials and building field extensions:

79 Lemma. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over a field K and $\ell \in \mathbb{Z}_{\geq 2}$. Consider the quotient ring*

$$R := K[X,Y]/(Y^2 - X^3 - aX - b,\ \psi_\ell(X,Y))\,.$$

Let $P = (x,y) \in E$ be a point of order ℓ defined over some extension field L/K. Then the map

$$f\colon R \to L,\ X \mapsto x,\ Y \mapsto y$$

is a ring homomorphism.

Proof. This is because $y^2 = x^3 + ax + b$ and $\psi_\ell(x,y) = 0$. □

What this means in practice is that we can take any algebraic computation in L that works only with elements in the image of f, such as the "finding the trace modulo ℓ subroutine" of Schoof's algorithm, and pull it back to R. In fact, we can simply perform the entire computation over R without even fixing L or P a priori. This is the neat trick!

80 Remark. Instead of computing modulo $\psi_\ell(X,Y)$, we may compute modulo $\psi_\ell(X, X^3 + aX + b)$, which has the same roots (albeit not necessarily with the same multiplicities) but is always univariate. Some sources go as far as calling $\psi_\ell(X, X^3 + aX + b) \in K[X]$ "the" division polynomial.

81 Remark. Computing in $R = K[X,Y]/(Y^2 - X^3 - aX - b,\ h(X))$ for some nonconstant polynomial $h \in K[X]$ is best viewed as working in the isomorphic ring

$$R_Y := R_X[Y]/(Y^2 - (\overline{X^3 + aX + b}))$$

where

$$R_X := K[X]/(h(X))$$

instead. Concretely, this means doing arithmetic in $K[X,Y]$ or $K[X][Y]$ while (e.g. after each operation) replacing each occurrence of Y^{2k} by $(X^3 + aX + b)^k$, then reducing the result $g_0(X) + g_1(X) \cdot Y$ to

$$(g_0 \bmod h)(X) + (g_1 \bmod h)(X) \cdot Y\,.$$

This keeps the size of an element of R in terms of the number of coefficients in K bounded by $2\deg(h)$.

82 Remark. Beware: The ring R does have zero divisors, so the elliptic-curve addition formulas can fail when trying to divide by nonzero nonunits. While treating this properly in theory causes quite a bit of trouble, there is a very easy practical solution (also known as a "really cool hack") for it: Finding a zero divisor basically means accidentally stumbling upon a proper divisor of ψ_ℓ. But then we can simply continue with one of the factors in place of ψ_ℓ and by construction it is still guaranteed that there is a ring homomorphism which maps (X, Y) to a point of order ℓ on E, which is all we need for Schoof's algorithm. We refer to the point $(X, Y) \in E(R)$ and its multiples as *symbolic* points.

83 Schoof's algorithm.
Input: The coefficients in the Weierstraß equation of an elliptic curve $E/\mathbb{F}_q$.
Output: The trace of Frobenius $\mathrm{tr}(\pi_q)$.

(1) Initialize $A := 1$ and $S := \{\}$ and $\ell := 1$.

(2) While $A < 4\sqrt{q}$: (Recall the Hasse bound: $|\mathrm{tr}(\pi_q)| \leq 2\sqrt{q}$.)

  (a) Update ℓ to the next prime after ℓ.

  (b) Find $k \in \{0, ..., \ell-1\}$ such that $k \equiv \pi_q \pmod{\ell}$ by evaluating the equation $\pi_q^2 + [q \bmod \ell] \stackrel{?}{=} [k]\pi_q$ at a (possibly symbolic) point of order ℓ.

  (c) Set $A := \ell A$ and $S := S \cup \{(k, \ell)\}$.

(3) Using the Chinese Remainder Theorem (CRT), compute an integer $t \in [-A/2; A/2]$ such that $t \equiv k \pmod{\ell}$ for all $(k, \ell) \in S$. Return t.

After how many primes ℓ does the outer loop terminate?

84 Theorem. *There exists a constant k_0 such that for all $k \geq k_0$, the product of the first k primes is bounded below by $\exp(k \ln k)$.*

In particular, this bound implies that the algorithm finishes with $\ell \in O(\log q)$.

Complexity: Testing each possible value of k consists of $O(\log q)$ arithmetic operations in the quotient

$$K[X, Y]/(Y^2 - X^3 - aX - B,\ \psi_\ell(X, Y))$$

where arithmetic takes $O((\log q \cdot \deg \psi_\ell)^2) = O((\log q \ell^2)^2) \subseteq O((\log q)^6)$ bit operations using naïve multiplication and $\widetilde{O}(\log q \cdot \deg \psi_\ell) = \widetilde{O}((\log q)\ell^2) \subseteq \widetilde{O}((\log q)^3)$ bit operations using asymptotically fast multiplication. There are $O(\ell) = O(\log q)$ such k to try for each ℓ, and there are $O(\log q)$ different choices of ℓ.

$\Longrightarrow$ The complexity of a naïve version of Schoof's algorithm is $O((\log q)^8)$ bit operations; fast arithmetic reduces this to $\widetilde{O}((\log q)^5)$.

85 Remark. After Schoof published his original algorithm, Elkies and Atkin proposed optimizations, among which Elkies' in particular makes a huge difference in practice: Instead of working with division polynomials, which represent the entire subgroup $E[\ell]$, he directly constructs a degree-ℓ divisor of ψ_ℓ which represents a cyclic order-ℓ subgroup of $E[\ell]$.

With this improvement, and by using asymptotically fast multiplication, the complexity of the Schoof–Elkies–Atkin algorithm becomes $\widetilde{O}((\log q)^4)$.

As promised earlier, Schoof's algorithm is exponentially better at counting points than the baby-step giant-step algorithm, which is definitely a cause for celebration.

86 Remark. As discussed earlier, working with a large prime-order group is *necessary* for the hardness of the discrete-logarithm problem, but it is by no means sufficient, even on an elliptic curve. We will see some examples of totally insecure curves and curves with degraded security (in spite of having large prime-order subgroups) later.

4 Efficient & secure elliptic-curve cryptography

Most of the time, for simplicity, we've been working with affine coordinates $(x, y) \in E$ on a (short or long) Weierstraß curve. This is convenient for toy examples and for general mathematical algorithms due to its generality and ease of implementation, but it is not optimal for fast and secure, "serious" cryptographic implementations. We will now discuss some ways in which elliptic curves can fail to provide security for entirely practical reasons, and show how to hopefully prevent many of those issues.

4.1 Efficiency: Eliminating inversions

Generally speaking, computing an inverse in a finite field is significantly more costly than computing a multiplication. Simple example: Computing $x^{-1} \in \mathbb{F}_p$ using Fermat's little theorem amounts to computing $x^{p-2} \in \mathbb{F}_p$. Using basic square-and-multiply, this takes $\Theta(\log(p))$ multiplications in $\mathbb{F}_p$. For large p, inversions can thus incur a very large cost. Hence, in the interest of efficiency, it makes sense to trade off an inversion for a few multiplications whenever possible.

The standard way of doing are *projective coordinates*: Instead of storing an element x in $\mathbb{F}_p$ as (say) an integer in $\{0, ..., p-1\}$, store it as a *pair* of two integers (x_0, x_1) with $x_0 \in \{0, ..., p-1\}$ and $x_1 \in \{1, ..., p-1\}$, which is understood to represent the element $x_0/x_1 \in \mathbb{F}_p$. To compute with this representation, we may simply use the calculation rules for fractions: The product of two elements (x_0, x_1) and (y_0, y_1) becomes (x_0y_0, x_1y_1), and similarly the sum becomes $(x_0y_1 + x_1y_0, x_1y_1)$.

87 Remark. Given the name and nature of this technique, one may be inclined to assume it is restricted to projective varieties (such as elliptic curves and other abelian varieties). However, the same idea works in full generality for any kind of algebraic computation![8] As such, the connection to projective space is almost a coincidence.

The main purpose of projective coordinates is speed, but in fact, it also helps quite a bit with side-channel resistance (to be discussed soon): The reason is that inversions are "naturally" tricky to harden against physical attacks.

4.2 Efficiency: Eliminating the y

Another, much less obvious, trick used in elliptic-curve cryptography for efficiency is based on the simple observation that *almost all of the information in an elliptic-curve point is contained in the x-coordinate*. From this, the idea follows naturally to transmit only the x-coordinate instead of full points when performing (say) a key exchange à la Diffie–Hellman. However, does it work? (Spoiler: Yes!)

88 Lemma. *Let P be a point on a (long) Weierstraß elliptic curve E and $n \in \mathbb{Z}$. Then $[n](-P) = [-n]P = -[n]P$. Ad-hoc notation: Write $\kappa\colon E \to \mathbb{P}^1$, $(x : y : z) \mapsto (x : z)$. Then $\kappa([n]P) = \kappa([n](-P))$.*

Proof. The equality $[n](-P) = [-n]P = -[n]P$ holds true in any group. For the second claim, recall the negation formula $-(X : Y : Z) = (X : -Y - a_1X - a_3 : Z)$ for (long) Weierstraß curves. □

Fifth lecture (May 27)

89 Corollary. *Let E/K be a (long) Weierstraß curve over a field K. For any $n \in \mathbb{Z}$ and $(X : Z) \in \mathbb{P}^1(K)$, there exists a unique $(X' : Z') \in \mathbb{P}^1(K)$ such that for all $Q \in E$ with $\kappa(Q) = (X : Z)$ we have $\kappa([n]Q) = (X' : Z')$.*

Proof. All preimages of $(X : Z)$ under κ are negatives of one another. Hence the claim follows from the lemma. □

[8] "Algebraic" here means: A sequence of operations in some ring.

90 Remark. The image $\mathcal{K}$ of κ is known as the *Kummer line* of E; it is defined as the quotient of E by the negation map $P \mapsto -P$.[9] We'll use the notation $\pm P$ for the equivalence class of a point $P \in E$ on the Kummer line $\mathcal{K}$.

Note that cryptographers usually speak of "x-only arithmetic" when referring to computations on the Kummer line.

91 Definition. Let $\texttt{xMUL}_n \colon \mathcal{K} \to \mathcal{K}$ be the map that takes each $(X : Z)$ to the associated $(X' : Z')$ from the corollary.

In practice, one often works with $Z, Z' \neq 0$ and (by some slight abuse of notation) views $\texttt{xMUL}_n$ as a map on affine x-coordinates only (hence the name).

From the theorem at the end of Section 3.4 about the representation of $[n]$ in terms of polynomials, it is evident that $\texttt{xMUL}_n$ is algebraic. (It can be written as a rational function.) However, this representation is not very useful for cryptography, as its size is $\Theta(n^2)$ field elements, which totally negates the advantage that users gain over attackers by using double-and-add. So, we need a version of double-and-add that works on the Kummer line only: This is not automatic since the Kummer line no longer carries a group structure after identifying each point with its negative.

- Clearly, doublings are no problem on the Kummer line: The doubling formula is dependent on x only, anyway.
- Additions are trickier: They really do require the sign of the input points, as there is otherwise no way to distinguish $\pm(P+Q) \in \mathcal{K}$ and $\pm(P-Q) \in \mathcal{K}$. The solution for this is "differential addition":

92 Lemma. *Let $P, Q \in E$. The three Kummer points $\pm P, \pm Q, \pm(P-Q)$ uniquely determine $\pm(P+Q) \in \mathcal{K}$. For (long) Weierstraß curves,[10] there exist algebraic formulas to calculate $\pm(P+Q)$ from $\pm P, \pm Q, \pm(P-Q)$.*

Proof. (Homework.) □

93 Definition. Write $\texttt{xDBL} \colon \mathcal{K} \to \mathcal{K}$ for "x-only doubling", i.e., mapping $\pm P \in \mathcal{K}$ to $\pm[2]P$. (In other words, $\texttt{xDBL} = \texttt{xMUL}_2$.) Write xADD for "$x$-only (differential) addition", which maps triples of the form $(\pm P, \pm Q, \pm(P-Q)) \in \mathcal{K}^3$ to $\pm(P+Q)$.

Now, how do we compute $\texttt{xMUL}_n$ using xDBL and xADD?

94 The Montgomery ladder.
Input: Integer $n \in \mathbb{Z}$, Kummer line point $\pm P \in \mathcal{K}$.
Output: The Kummer line point $\texttt{xMUL}_n(\pm P) = \pm[n]P$.

(1) If $n < 0$, set $n := -n$.

(2) Compute the binary expansion $n = \sum_{i=0}^{\ell-1} b_i 2^i$ with each $b_i \in \{0, 1\}$.

(3) Initialize $\pm R_0 := (1 : 0) \in \mathcal{K}$ and $\pm R_1 := \pm P \in \mathcal{K}$.

(4) For k ranging from $\ell - 1$ down to 0:

  (a) Set $\pm R_{1-b_k} := \texttt{xADD}(\pm R_0, \pm R_1, \pm P)$.

  (b) Set $\pm R_{b_k} := \texttt{xDBL}(\pm R_{b_k})$.

(5) Return $\pm R_0$.

95 Lemma. *The Montgomery ladder is correct.*

[9] As an algebraic variety, the Kummer line is simply $\mathbb{P}^1(K)$; however, it inherits additional structure from E.

[10] Actually, this holds in utmost generality, but we cannot prove this, nor need it, here.

Proof. First, notice that $\lfloor n/2^j \rfloor = \sum_{i=j}^{\ell-1} b_i 2^{i-j}$ for all $j \geq 0$.

We claim that the following invariant holds: At the beginning of the main loop, for $w \in \{0, 1\}$,

$$\pm R_w = \texttt{xMUL}_{\lfloor n/2^{k+1} \rfloor + w}(\pm P)\,.$$

(In particular, this implies that the inputs to `xADD` are valid.) To verify the loop invariant, let $(\pm R'_0, \pm R'_1)$ denote the values of the $\pm R_w$ at the *end* of the main loop for some given k. Hence

$$\begin{aligned} \pm R'_{1-b_k} &= \texttt{xADD}(\pm R_0, \pm R_1, \pm P) = \texttt{xMUL}_{2\lfloor n/2^{k+1} \rfloor + 1}(\pm P)\,; \\ \pm R'_{b_k} &= \texttt{xDBL}(\pm R_{b_k}) = \texttt{xMUL}_{2\lfloor n/2^{k+1} \rfloor + 2b_k}(\pm P) \end{aligned}$$

by assumption. Now, from

$$2\lfloor n/2^{k+1} \rfloor = 2 \sum_{i=k+1}^{\ell-1} b_i 2^{i-k-1} = \sum_{i=k+1}^{\ell-1} b_i 2^{i-k} = \lfloor n/2^k \rfloor - b_k\,,$$

it follows that

$$\begin{aligned} \pm R'_{1-b_k} &= \texttt{xMUL}_{\lfloor n/2^k \rfloor - b_k + 1}(\pm P)\,; \\ \pm R'_{b_k} &= \texttt{xMUL}_{\lfloor n/2^k \rfloor - b_k + 2b_k}(\pm P)\,. \end{aligned}$$

Substituting in $b_k = 0$ and $b_k = 1$ yields $\pm R'_0 = \texttt{xMUL}_{\lfloor n/2^k \rfloor}(\pm P)$ and $\pm R'_1 = \texttt{xMUL}_{\lfloor n/2^k \rfloor + 1}(\pm P)$, as claimed. □

96 Remark. A major benefit of the Montgomery ladder, besides saving the y-coordinate, is that it is comparatively easy to harden against side-channel attacks: This is the topic of Sections 4.4 and 4.5.

97 Remark. There exist other cryptographic protocols where x-only arithmetic is not advantageous: One prominent example where this typically happens are digital-signature algorithms, which we'll discuss later.

4.3 Invalid-curve attacks

Generally, high-level descriptions of cryptosystems tend to specify that certain mathematical objects be transmitted, but do not necessarily say how these things are represented as bit strings and how the implementation could detect it when invalid data is received. Depending on context, implementations may explicitly check the validity of the received data and abort if something is wrong,[11] or not check anything and simply perform possibly ill-defined or dangerous computations on the received data, anyway.

98 Example. Suppose Alice and Bob are doing a Diffie–Hellman key exchange in the group $(\mathbb{F}_p^\times, \cdot)$. The most obvious way to represent elements of $\mathbb{F}_p^\times$ is as elements of $\mathbb{F}_p$, which are in turn represented as as integers between 0 and $p-1$. However, $0 \in \mathbb{F}_p \setminus \mathbb{F}_p^\times$!

Hence, if Bob[12] transmits $0 \in \mathbb{F}_p$ as his public key, he is technically sending a "thing" that lies outside the valid public-key space. In this particular case the shared secret would always end up being zero. This is not catastrophic since all Bob achieves is to simply void the security of the key exchange, which he could do anyway by simply broadcasting the shared secret to the world, but it is an example of an invalid input and there are other settings where Bob can learn information about Alice's private key(!) using the same kind of trick.

The example illustrates how the idealized view of the Diffie–Hellman protocol assumes that the received group elements are actually valid: The public keys should be elements of $\mathbb{F}_q^\times$ or points on a particular elliptic curve. In this section, we investigate what happens in the context of elliptic-curve Diffie–Hellman when this (often implicit) assumption is violated.

[11] We note in passing that there are cryptographic schemes where checking the validity of a message received from another participant is provably as hard as breaking the scheme, hence there really is no way to perform this check. In such cases, resilience against maliciously crafted invalid inputs must be built using a careful workaround.

[12] Note that Bob may very well be malicious, and that "Bob" may in fact be someone else entirely. *"On the Internet, nobody knows you're a dog."*

4.3.1 Small-subgroup attacks

Suppose Bob is performing an ECDH key exchange with Alice, who uses the same key pair $(\alpha, [\alpha]G)$ every time. After the shared secret is established, they are to communicate securely using the shared secret and symmetric cryptography.

Standard **example**: Bob is an internet user who connects to a webserver Alice. After the initial TLS handshake, the first thing that happens is that Bob's browser sends an encrypted HTTP request using the shared secret, and Alice should respond with the requested data (i.e., typically a cute photo of a cat).

Now, in the case that Bob for some reason gets the wrong shared secret, he can clearly distinguish this from the normal behaviour: Since Alice cannot even decrypt Bob's request, she cannot possibly respond the same way as before.

The bottom line is that Bob (in typical deployments of ECDH) has access to an *oracle* which takes a public-key point P from Bob, computes a shared secret on Alice's side with Alice's private key, and Bob gets to *test* whether his own alleged shared secret matches the one Alice actually obtained.

99 Remark. In cryptography, an **oracle** is an idealized black box which takes a well-defined set of inputs and returns the result of a particular specified operation on it, possibly involving secret key material.

They are often used in the context of attacks (construct an attack given some oracle coming from implementation mistakes) and "provable" security (construct a solver for some claimed hard problem assuming there exists an attack).

If the curve E being used has a point P of "small" order $\ell \geq 2$, then this oracle can in fact be used to learn information about Alice's private key a: Bob will keep sending that point P as his public key and iteratively guess $[0]P$, $[1]P$, ..., $[\ell-1]P$ as the shared secret. With some point $[k]P$, the connection will succeed: then Bob has learned that Alice's shared secret $[\alpha]P$ equals $[k]P$, therefore $\alpha \equiv k \pmod{\ell}$.

Now, of course, the curves actually used for ECDH are designed to have large prime order (or perhaps order very close to a large prime with a small "cofactor"), so this attack does not teach Bob very much about Alice's private key: Guessing $\Theta(q)$ many points (over the wire!) will certainly not be more efficient than a straightforward BSGS attack on Alice's public key, which takes time $O(\sqrt{q})$ and does not require interaction with the oracle, i.e., Alice.

However, when things go wrong, the same technique *can* apply!

4.3.2 Invalid curve points

We will now assume Alice has a fixed ECDH key pair $(\alpha, [\alpha]G)$ and we assume access to the following oracle: Given two points $P = (x, y) \in \mathbb{F}_q^2$ and $S \in \mathbb{F}_q^2$, compute the elliptic-curve scalar multiplication $[\alpha]P$ using the standard short Weierstraß addition formulas and return `yes` if $[\alpha]P = S$, otherwise `no`.

Crucially, no test is performed whether the point P actually lies on the curve $E\colon y^2 = x^3 + ax + b$. In order to determine what happens to P in this case, we recall the addition formulas for short Weierstraß curves from Section 2.1.1:

- The negative of (x, y) is $(x, -y)$.
- The sum of (x_1, y_1) and (x_2, y_2) is either equal to ∞ or (x_3, y_3) where $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$. Here $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ or $\lambda = \frac{3x_1^2 + a}{2y_1}$.

The key point to notice is that *none of these formulas involves the curve coefficient b.* Thus, the exact same elliptic-curve addition formulas will work on any other elliptic curve $E'\colon y^2 = x^3 + ax + b'$ with $b' \in \mathbb{F}_q$!

This is useful because there exists, for any $P = (x, y) \in \mathbb{F}_q^2$, such a b': It is readily determined by $b' = y^2 - x^3 - ax$ with the given values of x and y substituted in. Therefore, we can reinterpret Alice's computation of a scalar multiplication on E with an invalid point as a scalar multiplication on E'.

100 Remark. Notice how sending a point outside E totally breaks the idealized view where a public key can only be a point on E, and that what happens if this is violated is very much implementation-dependent. The fact that the formulas still "work" and do something meaningful — computing on E' — is somewhat natural here, but nothing of the sort can be expected to occur in other types of cryptographic constructions in general.

Either way, the big issue is that E' may be a much weaker curve than E in the context of ECDH! The primary reason for this to happen is when the number of points on E' contains a small prime factor, since we can then simply run a small-subgroup attack as described above.

101 Remark. One may be tempted to think that a fully smooth-order curve E' is the best for an attacker: However, it is crucial to note that *Alice computes her public key with the real base point* and so her public key will remain as strong as ever.

Given a much stronger oracle that does not only answer `yes` or `no` but actually *returns* the shared secret $[\alpha]P$ obtained by Alice, we may however use a fully smooth curve and solve for α after a single oracle query using Pohlig–Hellman.

This type of oracle does not make sense in pure DH key exchange, but can appear in more complicated protocols. It is also natural in fault attacks (to be discussed later).

Finding weak curves. For this attack, finding weak curves is extremely straightforward: We simply brute-force search for b' such that $\#E'(\mathbb{F}_q)$ has a small (not necessarily prime) divisor ℓ.

Running the attack. After enough curves E' with varying such ℓ have been obtained, we may run the small-subgroup attack for all those curves E' with their corresponding ℓ to learn congruence conditions on Alice's private key α. As usual, enough such conditions allow us to recover $\alpha \in \mathbb{Z}$ using the Chinese remainder theorem.

Number of queries. For simplicity, assume that the ℓ are all prime. Let's say $\#E(\mathbb{F}_q) = n$, so that $\alpha \in \{0, ..., n-1\}$. Then we need $\prod_{i=1}^{r} \ell_i \geq n$, where ℓ_i are the distinct small prime factors for the invalid curves E'. For each invalid curve, we expect to require $\Theta(\ell_i)$ oracle queries. Hence the total number of queries is in $O((\log n)^2)$, which takes exponentially less time than a brute-force attack on the ECDLP $(\alpha, [\alpha]G)$.

Practical example: For a 256-bit n, it is enough to take primes ≤ 200 for the ℓ_i, which results in a worst-case workload of about 4200 queries and about half of that on average.

Sixth lecture (June 3)

4.3.3 Twist security

At first sight, invalid-curve attacks no longer apply when x-only coordinates are used (as one should), since the y-coordinate is a big part of what allowed us to force a point onto a different curve with particular properties. However, as it turns out, there is still *one* "other" curve on which we can send invalid points!

The starting point is that not all $x \in \mathbb{F}_q$ are valid x-coordinates for points defined over $\mathbb{F}_q$: For about half of all $x \in \mathbb{F}_q$, the corresponding y-coordinate must lie in a quadratic extension $\mathbb{F}_{q^2}$. Thus, by sending invalid x-coordinates, we still land outside the intended Diffie–Hellman group $E(\mathbb{F}_q)$.

What is this group? Write $\#E(\mathbb{F}_q) = q + 1 - t$ as usual.[13] From Lemma 76, we get

$$\begin{aligned}\#E(\mathbb{F}_{q^2}) &= q^2 + 1 - (t \cdot t - q \cdot 2) = (q+1)^2 - t^2 \\ &= (q+1-t)(q+1+t) = \#E(\mathbb{F}_q) \cdot (q+1+t)\,.\end{aligned}$$

This suggests that the order of points with $x \in \mathbb{F}_q$ but $y \notin \mathbb{F}_q$ should divide the "new" part $q+1+t$. Recalling that the set $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points was $\ker(\pi - 1)$, we can prove this by looking at the action of Frobenius:

102 Lemma. *Let E be a short Weierstraß elliptic curve defined over $\mathbb{F}_q$. Then all points $P = (x, y)$ with $x \in \mathbb{F}_q$ and $y \notin \mathbb{F}_q$ lie in $\ker(\pi_q + 1)$.*

Proof. Since $x^q = x$, the image $\pi_q(P)$ must be either P or $-P$. If it was P, then $y^q = y$, which would contradict the assumption that $y \notin \mathbb{F}_q$. Hence $\pi_q(P) = -P$ and the claim follows. □

[13] The integer t is the trace of the q-power Frobenius.

103 Remark. The inclusion from the lemma is not an equality: Points with $y = 0$ lie in $\ker(\pi_q + 1)$ but don't satisfy $y \notin \mathbb{F}_q$. There is also the point at infinity in $\ker(\pi_q + 1)$.

104 Remark. The subgroup $\ker(\pi_q + 1)$ is sometimes called the *trace-zero subgroup* of $E(\mathbb{F}_{q^2})$.

Using this characterization, we can find a multiple of the orders of points with $x \in \mathbb{F}_q$ but $y \notin \mathbb{F}_q$:

105 Lemma. *Let $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q + 1 - t$. Then $\#\ker(\pi_q + 1) = q + 1 + t$.*

Proof. We have

$$\deg(\pi_q + 1) = (\pi_q + 1)(\widehat{\pi}_q + 1) = \pi_q\widehat{\pi}_q + 1 + \pi_q + \widehat{\pi}_q = \deg \pi_q + 1 + \operatorname{tr} \pi_q = q + 1 + t\,. \qquad \square$$

Points in the trace-zero subgroup are also often referred to as "points on the twist" since they become $\mathbb{F}_q$-rational after taking an isomorphism to the *quadratic twist* — hence the name "twist (in)security".

106 Definition. Let $E\colon y^2 = f(x)$ be an elliptic curve over K with $f(x)$ a cubic polynomial and let τ be a nonsquare in K. The *quadratic twist* of E, sometimes written E^t or $\widetilde{E}$, is the elliptic curve $y^2 = \tau f(x)$.

(There is a coordinate transform to convert this curve form into a standard Weierstraß equation.)

107 Lemma. *Let $E/\mathbb{F}_q$ be given by $y^2 = f(x)$ and consider its quadratic twist $E^t\colon y^2 = \tau f(x)$ where τ is a nonsquare in $\mathbb{F}_q$. Let π_q and π_q^t denote the q-power Frobenius endomorphisms of E and E^t respectively.*

Consider the isomorphism

$$\iota\colon E \to E^t,\ (x, y) \mapsto (x, \sqrt{\tau}y)$$

where $\sqrt{\tau}$ is a fixed square root of τ in $\mathbb{F}_{q^2}$. Then $\iota(\ker(\pi_q + 1)) = \ker(\pi_q^t - 1) = E^t(\mathbb{F}_q)$.

Proof. Note that $\pi_q^t \circ \iota = -\iota \circ \pi_q$ by an explicit calculation using the fact that $\sqrt{\tau}^q = -\sqrt{\tau}$ as before. Hence,

$$\ker(\pi_q^t - 1) = \ker(-\iota\pi_q\iota^{-1} - 1) = -\ker(\iota(\pi_q + 1)\iota^{-1}) = -\iota(\ker(\pi_q + 1))$$

since ι is an isomorphism. Note that we can drop the sign since $-G = G$ for any (sub)group G. $\square$

Twist attacks. All of this suggests an invalid-curve attack for x-only coordinates in case the "actual" group order $q + 1 - t$ is safe but the "twist" order $q + 1 + t$ is not (i.e., has only small prime divisors). The attack works exactly as before, except that we can no longer choose the insecure curve E' freely; rather, we must live with what is given (the quadratic twist).

Twist security. Since checking whether a given x-coordinate is valid may be fairly expensive, and since implementers may not always remember to perform safety checks which aren't necessary for the system to "*work*", it has become standard to prevent all such issues from the beginning by using curves where *both* the curve $E(\mathbb{F}_q)$ as well as its quadratic twist $E^t(\mathbb{F}_q)$ have a large prime factor in the order. These are known as *twist-secure* curves.

Curve25519. Perhaps the most prominent representative of this family, and the example that seems to have moved the concept into mainstream cryptography, is the curve *Curve25519* given by

$$y^2 = x^3 + 486662x^2 + x$$

over the prime field $\mathbb{F}_p$ with $p = 2^{255} - 19$. It satisfies

$$\#E(\mathbb{F}_p) = 8 \cdot 723700557733226221397318656304299424085711635937990760600195093828545425098 9\,;$$
$$\#E^t(\mathbb{F}_p) = 4 \cdot 1447401115466452442794637312608598848160326344765032579786049412540737390799 7\,.$$

4.4 Physical security and side-channel attacks

The previous section is an instance of something that happens in cryptography, everywhere, all the time:

108 Law of Leaky Abstractions. All non-trivial abstractions, to some degree, are leaky.

— Joel Spolsky (cofounder of *Stack Overflow*)

Earlier it was the abstraction that "curve points" can be sent over the wire, when in fact the wire transmits zeroes and ones (if you're lucky). As a matter of fact, the real world is scary: Physics itself causes quite a few difficulties when trying to deploy a-priori-flawless cryptography in a secure manner. First, we will survey the most important types of *side-channel attacks*.

4.4.1 Power consumption and emissions

Scenario: Smartcards, which includes things like bank cards or the electronic portion of some types of travel documents. These cards contain a tiny computer processor as well as a private key which is used to authorize transactions. Ideally, the cards should be *uncloneable*; in particular, it must not be possible to extract the private key.

However, for such cards, the environment (such as a manipulated payment terminal) can observe plenty of physical side effects of the computation the card is performing while it authorizes a transaction. One of them is **power consumption**: In a nutshell, in typical digital computers, storing a 0 requires an amount of power that's different from storing a 1. Another physical effect, in a sense the second side of the same coin, is **electromagnetic radiation**: Any time a current flows through a wire or circuit, this changes the electromagnetic field around the various parts of the chip.

In both cases, the resulting leakage *correlates* to some extent with the secret data being processed, and more often than not some very careful measurements and post-processing of the data can reveal information about the secret.

4.4.2 Fault attacks

Scenario: Similar types of applications as above. However, in addition to passively measuring physical side effects of the computation, the attacker may now also ***actively* tamper with the device** to cause it to **misbehave** in a way that makes it spit out secrets or skip crucial checks.

Common techniques to artificially trigger faults include playing around with the power supply ("voltage glitching"), manipulating the timing of the signal flow inside the chip ("clock glitching"), or injecting an electromagnetic or laser pulse in some part of the chip to induce spurious currents and thus flip bits.[14]

A very basic example are *loop-abort faults*: Imagine that an attacker manages to change the conditional jump back to the start of the main loop in Algorithm 94 so that the loop is terminated after only ℓ' instead of the intended ℓ iterations. Then, according to the proof of Lemma 95, the returned value is the scalar multiple of *the top ℓ' bits of the private key* with the given point. Timing the loop abort carefully will thus allow an attacker to solve a much smaller DLP. Repeatedly aborting at later points of the calculation and keeping the current known bits of the private key updated thus allows an attacker attacker to fully recover the private key with only a few successful fault injections.

4.4.3 Timing attacks

Scenario: Servers on a local or remote network.

> Cryptosystems often take slightly different amounts of time to process different inputs. Reasons include performance optimizations to bypass unnecessary operations, branching and conditional statements, RAM cache hits, processor instructions (such as multiplication and division) that run in non-fixed time, and a wide variety of other causes.

[14] In fact, very similar kinds of faults can occur naturally due to cosmic radiation: Engineers have to minimize the risk of such events when designing things like computers that go to space, or other critical control hardware (even here on Earth).

> Performance characteristics typically depend on both the encryption key and the input data (e.g., plaintext or ciphertext).
>
> While it is known that timing channels can leak data or keys across a controlled perimeter, intuition might suggest that unintentional timing characteristics would only reveal a small amount of information from a cryptosystem (such as the Hamming weight of the key).
>
> However, attacks are presented which can exploit timing measurements from vulnerable systems to find the entire secret key.
>
> — Paul C. Kocher, "Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems", 1996

The way *timing attacks* generally work is by repeatedly sending carefully chosen inputs to the target system, which then leaks some *information about its secrets* via **timing differences inside the processor** resulting from various things.

Timing attacks are, among the classes of side-channel attacks I discuss here, the easiest to mitigate when writing software: That is the topic of Section 4.5.

4.4.4 Microarchitectural side channels

Scenario: Web browsers (which execute code from the internet on your machine), cloud computing (which runs virtual computers belonging to separate users on one physical computer).

Modern computers are extremely complex devices: Even if you program in assembly language, the machine model exposed to assembly is essentially "fake": It is emulated by a bunch of even more low-level controllers (instruction decoder, memory-management unit, …) that have a number of arithmetic/logic units (ALUs) and different kinds of memory (including registers) and various other components available to them. Using these resources, the processor is designed to execute a given program as efficiently as possible on the available hardware.

In particular: "Memory" actually contains many different types of memory. As a rule of thumb, the closer to the actual processing core some data lies, the faster it is to access, but the amount of very fast memory is limited by chip design considerations as well as (fundamentally) the speed of light.

Hence, modern CPUs employ multiple layers of *caching* to automatically detect data that is needed frequently and keep it in a fast memory location close to the processing core. However, when combined with the fact that malicious processes may run in parallel on the same machine, these caches may leak information if one is not careful: They can **reveal *to other processes* which memory locations have been accessed**, based on how long they take to load.

There are plenty of other *microarchitectural side channels* like this: "Spectre" and "Meltdown" refer to entire classes of attacks that exploit advanced CPU optimizations (in this case, *speculative execution*) to access data that "should" be hidden if the CPU was not trying to be clever.

4.4.5 There's more!

Imagination is almost endless when it comes to side-channel attacks.

See for example Genkin–Shamir–Tromer, "Acoustic Cryptanalysis" (2016), where they record the sound made by a laptop while running RSA decryption "using a plain mobile phone placed next to the computer" and recover the private key using the acoustic emissions.

Seventh lecture (June 10)

4.5 "Constant-time" software

In a nutshell, "constant-time" means that the execution time of an algorithm is independent of some of its inputs (which are marked as secrets).[15]

[15] Note how this does not at all imply that the time taken must actually be constant. It is also distinct from the complexity-theoretic meaning $O(1)$.

In practice, this means we must not use any (1) branches conditional on secrets, and (2) instructions with input-dependent timings if the input is a secret. The most important subclass of the latter are (3) memory accesses with secret-dependent locations.

Are the cryptographic algorithm we've seen so far "constant-time"?

- The square-and-multiply or double-and-add algorithm very much isn't: Every branch in the main loop depends on one bit of the private key.
- The standard elliptic-curve addition formulas aren't: They involve a case distinction based on whether $P \stackrel{?}{=} Q$ or not.
- The Montgomery ladder (Algorithm 94) is not *as stated* since the memory accesses for loading and storing either R_{b_k} or R_{1-b_k} depend on the secret bit. We can however fix this using a *constant-time conditional swap*.

109 Constant-time conditional swap.
Input: Secret bit `c` $\in$ `{0,1}`, two "objects" `x[0..l-1]` and `y[0..l-1]` of the same size (`l` bytes).
Effect: Swap `x` and `y` if `m` is set to `1`, nothing otherwise — in constant time.

(1) Set byte `m := 0 - c`. (This value either has all bits set to `0` or all bits set to `1`.)

(2) For `i` from `0` to `l-1`:

  (a) Set byte `t := m AND (x[i] XOR y[i])`. (These logic operations are performed bitwise.)

  (b) Set `x[i] := x[i] XOR t`.

  (c) Set `y[i] := y[i] XOR t`.

110 Remark. It is clear that the algorithm as written has control flow and memory accesses that is independent of `c`.

Note however that (depending on the particular programming language) nothing may stop a compiler from replacing the constant-time version by an "optimized" version that is totally vulnerable against side channels. From a cryptographic-engineering perspective, this is a shortcoming of traditional compiler design, which focuses on preserving semantic equivalence only (but not on side-channel safety).

With this conditional swap plugged into the Montgomery ladder (main loop: `CSWAP`, `xDBL`, `xADD`, `CSWAP`), it is automatically constant-time assuming the x-only formulas are. We have seen in homework set 4, exercise 1 (cf. set 6, exercise 2, part 1) that there are x-only formulas which require case distinctions only for special cases that are easily avoided, which implies that the Montgomery ladder can be implemented in an entirely branch-free manner using such formulas.

Constant-time "`if`"s... work very similarly: We simply compute the result of both branches, then apply a conditional assignment (using bit operations) of the result we want, as selected by a boolean variable converted to a bitmask.

111 Remark. There are fixes and workarounds for all of the issues described earlier. Power or EM leakage is usually hidden using "masking": Using randomized redundant representations for all the secrets internally so that the leakage is hidden by the unknown randomness inside the representation. Fault attacks can be prevented from succeeding in various algorithmic ad-hoc ways, or by hardening the hardware itself. Microarchitectural side channels are mitigated using a combination of software fixes in compilers, operating systems, and CPU microcode updates. (CPU manufacturers, however, typically choose to only mitigate those issues where it is easy to do so without too much of a performance impact: Their #1 marketing argument is performing well in benchmarks, not security.)

4.6 Montgomery curves

For performance, it is very common to divert from the short Weierstraß form in cryptographic practice:

112 Definition. Let K be a field. A *Montgomery (elliptic) curve* is a projective plane curve of the form

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

with $A, B \in K$ with $B \neq 0$ and $A \notin \{\pm 2\}$. (I call the case $B = 1$ "untwisted", but this is not yet standard.)

One advantage of Montgomery curves is that they always exhibit the point $(0, 0)$ of order 2 which makes some things very nice computationally.

113 Lemma. *Let E be a Montgomery curve and $P = (x, y) \in E$ with $x \neq 0$. Writing $T = (0, 0) \in E$, we have*

$$P + T = (1/x, -y/x^2).$$

Proof. (Homework.) □

114 Lemma. *Let E be a Montgomery curve. Consider points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on E such that $0 \notin \{x_1, x_2, x_1 - x_2\}$. Write $P + Q = (x_3, y_3)$ and $P - Q = (x_4, y_4)$. Then*

$$x_3 x_4 (x_1 - x_2)^2 = (x_1 x_2 - 1)^2.$$

Proof. Omitted: This works the same as homework set 4, exercise 1, part 2. □

As a consequence, there are very nice combined and streamlined x-only doubling-and-addition formulas:

115 Lemma. *Consider the Montgomery curve $y^2 = x^3 + Ax^2 + x$ over a field K and suppose given the value $\mathtt{a24} = (A+2)/4$. Let $\pm P = (X_2 : Z_2) \in \mathcal{K}$, $\pm Q = (X_3 : Z_3) \in \mathcal{K}$, and $\pm(P - Q) = (X_1 : 1) \in \mathcal{K}$, such that $X_1 \neq 0$. The following sequence of operations*[16] *computes $\pm[2]P = (X_4 : Z_4)$ and $\pm(P + Q) = (X_5 : Z_5)$ using 5 multiplications, 4 squarings, 1 multiplication by $\mathtt{a24}$, and 8 additions or subtractions.*

(A1) $\mathtt{A} = \mathtt{X}_2 + \mathtt{Z}_2$
(S1) $\mathtt{AA} = \mathtt{A}^2$
(A2) $\mathtt{B} = \mathtt{X}_2 - \mathtt{Z}_2$
(S2) $\mathtt{BB} = \mathtt{B}^2$
(A3) $\mathtt{E} = \mathtt{AA} - \mathtt{BB}$
(A4) $\mathtt{C} = \mathtt{X}_3 + \mathtt{Z}_3$
(A5) $\mathtt{D} = \mathtt{X}_3 - \mathtt{Z}_3$
(M1) $\mathtt{DA} = \mathtt{D} \cdot \mathtt{A}$
(M2) $\mathtt{CB} = \mathtt{C} \cdot \mathtt{B}$
(A6) $\mathtt{t}_0 = \mathtt{DA} + \mathtt{CB}$
(S3) $\mathtt{X5} = \mathtt{t}_0^2$
(A7) $\mathtt{t}_1 = \mathtt{DA} - \mathtt{CB}$
(S4) $\mathtt{t}_2 = \mathtt{t}_1^2$
(M3) $\mathtt{Z}_5 = \mathtt{X}_1 \cdot \mathtt{t}_2$
(M4) $\mathtt{X}_4 = \mathtt{AA} \cdot \mathtt{BB}$
(a24) $\mathtt{t}_3 = \mathtt{a24} \cdot \mathtt{E}$
(A8) $\mathtt{t}_4 = \mathtt{BB} + \mathtt{t}_3$
(M5) $\mathtt{Z}_4 = \mathtt{E} \cdot \mathtt{t}_4$

Proof. Omitted. □

[16] Source: `https://hyperelliptic.org/EFD/g1p/auto-code/montgom/xz/ladder/mladd-1987-m.op3`

As a consequence, this "ladder step" can be used inside a Montgomery ladder with no case distinctions, at least if the generator is not the point $(0, 0)$ of order two.[17]

116 Remark. Some notes on the notation for the cost of these operations:

- Typically, the cost of algebraic computations is dominated by the cost of the multiplicative operations: This effect is so extreme that people often assume additions to be "free".
- According to conventional wisdom, the cost of a squaring is about 80% of the cost of a multiplication. That is why squarings are counted separately.
- Similarly, the cost of the multiplication by `a24` depends a lot on the value of this constant. Extreme example: If `a24` equals, say, 5, then this multiplication can be replaced by five additions, which are (usually) much cheaper.

We may now perform a simple cost estimate: For each bit of the input scalar, the Montgomery ladder performs 5 or 6 multiplications and 4 squarings, for a total of ≈ 10 multiplication equivalents per bit. Looking at these numbers informs the expected performance difference between 250-bit ECDH and 3000-bit finite-field Diffie–Hellman (recall that the resulting security levels are comparable; cf. the remark at the end of Section 1.4): With elliptic curves we are performing about 10 times more base-field operations, but the field can be about 10 times smaller. If schoolbook multiplication is used, the cost of a ℓ-bit multiplication scales as $\Theta(\ell^2)$, hence in this case we might expect elliptic curves to be about

$$\frac{3000^2}{10 \cdot 250^2} \approx 14.4$$

times faster. And indeed, a quick experiment with `openssl` gets very close to this number:

```
$ openssl speed -seconds 1 ecdh
Doing 256 bits  ecdh ops for 1s: 25812 256-bits ECDH ops in 0.99s
$ openssl speed -seconds 1 ffdh
Doing 3072 bits  ffdh ops for 1s: 1930 3072-bits FFDH ops in 1.00s
```

Note that $25812/1930 \approx 13.37$.

There are more factors affecting the performance difference between elliptic curves and finite fields: The most important one is not the speed of arithmetic, but the cost of attacks. Indeed, key lengths for finite-field Diffie–Hellman scale as a cubic function of the security level, while for elliptic curves the scaling is linear.

4.7 Edwards curves

As we've seen above, one way to avoid the case distinction in the formulas is to make sure they won't appear in the cryptographic protocol we are implementing. This is more difficult for full point arithmetic than for x-only arithmetic: In all formulas we've seen so far, doublings and adding a point and its negative are special.

117 Remark. This discussion begs the question if there need to be exceptional cases at all. The answer is yes: It has been proved that every single addition formula for an elliptic curve must have special cases that mandate the use of a different formula.

(See Bosma and Lenstra, "Complete Systems of Two Addition Laws for Elliptic Curves", Journal of Number Theory (1995).)

Avoiding the exceptional cases can be done on a protocol level (see above). Another way of tackling the problem at a lower level, when choosing the curve, is to work with *complete formulas*: Those are set up in such a way that the exceptional cases are all excluded because they sit in an extension field. This should in principle be possible for all elliptic-curve shapes, but was historically first achieved for **Edwards curves** which also enjoy some other nice properties.

[17] This is not a restriction in practice since doing cryptography in a group of order two is fairly pointless.

118 Definition. Let K be a field of characteristic $\neq 2$. A *(twisted) Edwards curve* over K is a projective plane curve

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

where $0 \notin \{a, d, a - d\}$.

As written, this curve is *not* an elliptic curve, not even in the high-brow view of things: It is singular!

119 Lemma. *Let E be a (twisted) Edwards curve. Then there are two points at infinity on E, namely $(0 : 1 : 0)$ and $(1 : 0 : 0)$. Further, the points at infinity are (the only) singular points on E.*

Proof. (Homework.) □

However, the curve is "essentially equivalent" to an elliptic curve, in the sense that there is an invertible partial map to an elliptic curve that is defined almost everywhere:

120 Lemma. *Let E be a twisted Edwards curve given by $a, d \in K$ as above. Define the Montgomery curve $M\colon Bv^2 = u^3 + Au^2 + u$ where $A = 2(a + d)/(a - d)$ and $B = 4/(a - d)$. Then*

$$f\colon \quad E \dashrightarrow M,\ (x, y) \longmapsto \left(\frac{1 + y}{1 - y}, \frac{1 + y}{(1 - y)x}\right)$$

is a well-defined map except at $(0, -1) \in E$ and the two points at infinity on E. Conversely, the (partial) inverse

$$f^{-1}\colon \quad M \dashrightarrow E,\ (u, v) \longmapsto \left(\frac{u}{v}, \frac{u - 1}{u + 1}\right)$$

is a well-defined map except at the points of order two $(\ldots, 0) \in M$, as well as the point at infinity on M.

Proof. Verifying that these maps end up on the correct curve (where defined) is a straightforward computation.

We homogenize to treat the points at infinity correctly:

$$F = \big((z + y)x : (z + y)z : (z - y)x\big)\,.$$

Points where this map is undefined are those where $(z + y)x = (z + y)z = (z - y)x = 0$. Hence

$$(x = 0 \vee z = -y) \wedge (z = 0 \vee z = -y) \wedge (x = 0 \vee z = y)$$

If $x \neq 0$, then $z = -y \wedge z = y$, implying $y = z = 0$ since the characteristic was not 2; hence $(1 : 0 : 0)$ is the only such point. The case $z = 0$ implies $xy = 0$, hence $(0 : 1 : 0), (1 : 0 : 0) \in E$ are also exceptional.

The other direction works just the same. □

121 Remark. The map from Lemma 120 is a *birational equivalence*: This notion is very close to an isomorphism, but it is not quite since there are exceptional points where the map is really not defined (and no amount of rewriting the formulas into equivalent formulas can fix that).

This explains, among other things, how E can be a singular curve while M is non-singular.

122 Lemma. *Consider a (twisted) Edwards curve $E\colon ax^2 + y^2 = 1 + dx^2y^2$ over a field K of characteristic $\neq 2$ where a is a square and d is a non-square. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be K-rational points on E. Then*

$$P + Q = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right).$$

The neutral element with respect to this addition law is $\mathcal{O} = (0, 1)$. The negative of $P = (x, y)$ is $(-x, y)$.

Proof. We'll skip the derivation of the formulas. Proving that the formulas always work is homework. □

4.8 Digital signatures from elliptic curves

We'll now discuss signature schemes in some detail. Note that there are many digital-signature schemes based on elliptic curves; the most popular one is probably still *ECDSA*.

Signature schemes. As hinted in Section 1.1, the key cryptographic requirement for digital signatures is to be *unforgeable*: Given someone's public key and an arbitrary number of existing signatures, noone should be able to produce a signature for another message that the public-key owner did not sign.

At the heart of many (most?) signature schemes lies an *(interactive) identification scheme*. We'll first discuss these, since they are conceptually a bit simpler, and then see how to transform them into a (by definition *noninteractive*) signature scheme.

4.8.1 (Interactive) identification protocols

...are a way for a *prover* holding a private key a for some public key $[a]G$ to convince a *verifier* that they indeed know the secret a — without revealing it, of course. Their applications in cryptography are plentiful, including as a standalone primitive (e.g., authentication) and as a building block for other primitives (e.g., signatures) and more complicated protocols.

The fixed public parameters for the following are an (elliptic-curve) group E together with a "generator" point $G \in E$ of large prime order ℓ.

123 Identification from Diffie–Hellman.
Situation: The *prover* has a DH key pair (a, A) where $A = [a]G$.

(1) **Verifier:** Choose a random "nonce" $r \in \{0, ..., \ell-1\}$ and send $R := [r]G$ to the prover.

(2) **Prover:** Compute the Diffie–Hellman shared secret determined by A and R, i.e., the point $S := [a]R$, and send it to the verifier.

(3) **Verifier:** Compute the Diffie–Hellman shared secret as $[r]A$ and check that it matches the point S sent by the verifier.

This protocol clearly works, and it is secure unless an attacker can break the Diffie–Hellman key exchange, but it is not straightforward to turn it into a signature scheme because *the verifier has to keep the value r secret*. (There is a conceptually pleasing workaround for this based on pairings, which we will also see later.)

Luckily, there are other identification protocols for Diffie–Hellman public keys that are more convenient for the purpose of building a signature scheme:

124 Schnorr's identification protocol (1990ish).
Situation: The *prover* has a DH key pair (a, A) where $A = [a]G$.

(1) **Prover:** Choose a random "nonce" $r \in \{0, ..., \ell-1\}$ and send $R := [r]G$ to the verifier.

(2) **Verifier:** Pick a random challenge value $c \in \{0, ..., \ell-1\}$ and send it to the prover.

(3) **Prover:** Compute the scalar $s := (r + a \cdot c) \bmod \ell$ and send it to the verifier.

(4) **Verifier:** Check that $[s]G = R + [c]A$.

125 Remark. Schnorr's identification protocol is an instance of a "Σ-protocol".

126 Lemma. *The identification protocol is correct.*

Proof. This is a straightforward verification that $[s]G = R + [c]A$. □

At the heart of the security lie two results: First, *soundness*, which implies that anyone who passes the protocol must with overwhelming probability actually know a:

127 Lemma. *Given two distinct challenge-response pairs (c_1, s_1) and (c_2, s_2) for the protocol with fixed a and R, one can recover the private key a.*

Proof. We have $(s_2 - s_1) \equiv a(c_1 - c_2) \pmod{\ell}$, hence $a = (s_2 - s_1)/(c_1 - c_2) \bmod \ell$. □

128 Remark. Constructing such an *extractor* for the straightforward identification protocol from Diffie–Hellman (Protocol 123) is significantly more difficult: It boils down to reducing the discrete-logarithm problem (DLP) to the *computational Diffie–Hellman problem* (CDH), i.e., breaking the Diffie–Hellman key exchange. It is not known how to do this in full generality; we will however see a very interesting partial solution later.

Second, we require that the protocol can be repeated indefinitely without leaking any information about a. This is known as the *zero-knowledge property* and it is often proved by demonstrating *simulatability*:

129 Lemma. *There is a polynomial-time algorithm which, for a fixed key pair (a, A), on input the challenge value c, outputs a pair (R, s) such that the "transcript" (R, c, s) has the same distribution as in the protocol.*

Proof. The simulator can work backwards by solving the verification equation for R: Choose $s \in \{0, ..., \ell-1\}$ at random and let $R = [s]G - [c]A$.

It remains to show that the distribution of s in the real protocol is also uniform: This is because r is uniform in $\{0, ..., \ell-1\}$, hence $s = (r + a \cdot c) \bmod \ell$ is. □

The moral implication is that an observer who gets to see only the public values really cannot possibly learn anything from them: After all, the attacker could simply have generated transcripts with an identical distribution on their own, without requiring the use of any secrets at all!

As a combination of these two results, we thus get:

130 Corollary. *The identification protocol is secure assuming that computing discrete logarithms is hard.*

Now, in order to turn an identification protocol into a signature scheme, the only thing left to do is to *replace the verifier by a hash function*. What's that?

4.8.2 Hash functions

An absolutely crucial ingredient for the next construction (and most cryptography!) are *hash functions*. They take an arbitrary-length bit string and return a fixed-length bit string, the *hash value*:

$$H\colon \{0,1\}^* \to \{0,1\}^n\,.$$

In many cases, a hash value serves as something like a random-looking "fingerprint" of the input data. Various computational tasks are required to be difficult for H depending on the application:

- Preimage resistance: Given some hash value $v \in \{0,1\}^n$, it should be hard to find any $m \in \{0,1\}^*$ such that $H(m) = v$.
- Second-preimage resistance: Given some input $m_1 \in \{0,1\}^*$, it should be hard to find a *different* $m_2 \in \{0,1\}^*$ with $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.
- Collision resistance: It should be hard to find any pair $m_1, m_2 \in \{0,1\}^*$ with $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.

Well-known hash functions include *MD5* (dead), *SHA1* (dead), and *SHA2* and *SHA3* (currently alive).

131 Remark. In theory, in order for any computational task to be difficult, we should speak about *families* of hash functions. The reason is that for any fixed hash function, an attacker can simply precompute a constant amount of data (say, 2^{1000} input-output pairs) and find collisions or preimages in time $O(1)$.

(This is of course not a meaningful attack — rather, it highlights a shortcoming of some of the basic definitions of complexity theory when it comes to cryptography.[18])

[18] See also Bernstein & Lange: "Non-uniform cracks in the concrete: the power of free precomputation", Asiacrypt 2013.

4.8.3 The Fiat–Shamir transform

...is a generic way to construct a non-interactive protocol from an interactive one, in certain cases. It is part of the design blueprint of many different families of signature schemes.

The key point in Fiat–Shamir is to replace all messages received from the verifier by a hash function which digests as input the entire transcript of the previous interactions. That way, assuming the hash function is "secure" in some sense (see below), the prover can execute the entire protocol on their own, and a verifier can later check the correctness without interacting with the prover directly at all — i.e., just what we want from a signature scheme!

132 Remark. For this, it is crucial that all the messages from the verifier can be publicly generated: Otherwise, either too much information is revealed to the prover, rendering the proof system insecure, or the verifier would later struggle trying to verify the correctness of the given transcript (for lack of knowing the secrets).

Protocols with the property that all messages from the verifier can be generated without requiring any secret randomness are known as *public-coin* protocols.

After the Fiat–Shamir transform was applied, an attacker has more leverage: They may now retry the identification protocol at will until their challenge comes out favorable, or they may even try to attack the hash function directly. As such, assumptions going in the general direction of second-preimage resistance is typically the class of requirements we ask of a hash function in this situation.

4.8.4 Two concrete signature schemes

We'll now look at two (very similar) concrete incarnations of Schnorr's signature protocol which are both obtained using the Fiat–Shamir transform.

133 Remark. In both cases, the jump from a noninteractive identification scheme to a *signature* scheme is made by simply incorporating the message m into the transcript: Thus, it gets fed to the hash function together with the random value R when computing the challenge.

134 Schnorr's signature algorithm.
Key generation: Choose a random private key $a \leftarrow \{0, ..., \ell - 1\}$. The public key is $A := [a]G$.
Signing of a message m: Choose a random "nonce" $r \leftarrow \{0, ..., \ell - 1\}$. Compute the values $R := [r]G$, $h := H(R \parallel m)$, and finally $s := (r + h \cdot a) \bmod \ell$. The signature is (h, s).
Verification: Let $R' := [s]G - [h]A$. Recompute $h' := H(R' \parallel m)$. Check whether $h' = h$.

It is straightforward to check that the signature scheme works. Why is it secure? The basic idea is that an attacker cannot control the hash value $h = H(R \parallel m)$, and therefore the only way for someone who does *not* know the private key a to produce a signature is to solve the verification equation $R = [s]G - [h]A$ for the value s, which means solving the DLP $(G, R + [h]A)$.

135 Remark. An actual security proof would have to model possible attacks against the various parts of the protocol (ECDLP, the hash function, ...) and obtain a joint upper bound on the success probability of an efficient attacker parameterized by spending a given amount of time. Such security reductions are of paramount importance in cryptographic engineering (but would lead us somewhat astray in this course).

There are many variants of Schnorr's signature scheme, with small differences in how the randomness for r is generated and how the verification equation is checked at the end. For good measure, here is another variant, which was specified to use Edwards curves for speed (hence the name).

136 The EdDSA signature algorithm.
Key generation: Choose a random scalar $a \leftarrow \{0, ..., \ell - 1\}$ and a random bit string $seed \in \{0, 1\}^*$; these values make up the private key. The public key is $A := [a]G$.
Signing: Let $r := H(seed \| m)$ and compute $R := [r]G$. Let $h := H(R \| A \| m)$ and $s := \big(r + h \cdot a\big) \bmod \ell$. The signature is (R, s).
Verification: Compute $h' := H(R \parallel A \parallel m)$. Check whether $[s]G = R + [h']A$.

The differences to the previous version are:

- The value r is generated deterministically from the private key and input using the hash function. This renders the entire signature algorithm deterministic, which helps prevent catastrophic security fails that can occur if bad randomness sources are used during signing.[19]
- The hash used to produce the challenge value h includes the public key A: This is a layer of defense against *multi-target attacks*, where an attacker tries to forge a signature for any of multiple keys at once.
- The verification is closer to the original (interactive) sigma protocol: It is simply given the public values from the sigma protocol and checks that s is a solution to the challenge problem. In the previous version, the verifier instead recomputed the point R from the given values h and s and checked that it matches the correct challenge for this R.

  The reason for Schnorr's original choice is mostly historic: When done in finite fields instead of elliptic curves, the size of a group element is significantly bigger than the size of a scalar.[20] Thus, in that setting the signature (h, s) would've been much smaller than (R, s). With elliptic curves, there is no difference in size between the two.

137 Remark. The only practically used instantiation of EdDSA is *Ed25519*, which uses the Edwards version of the curve *Curve25519* we've seen before.

Ninth lecture (June 24)

4.9 The Diffie–Hellman problem

We've seen the Diffie–Hellman key exchange: Alice and Bob have private keys a and b and associated public keys $A = [a]G$ and $B = [b]G$; their shared secret is $S = [ab]G = [a]B = [b]A$.

138 Definition. Finding S given (A, B) is the *computational Diffie–Hellman problem* (CDH).

The obvious way for an attacker to compute S given A and B is to break one of the DLPs (G, A) or (G, B) to recover a or b, then compute S as $[a]B$ or $[b]A$. Question: Is this the only way one could find S?

139 Remark. For both DLP and CDH, one can consider two variants of the problem depending on whether G is an input or a fixed constant. The two variants are actually polynomial-time equivalent, so we may drop the distinction and work with the choice that's more convenient for any given purpose.

4.9.1 Reducing DLP to CDH

It is not known unconditionally whether CDH is as hard as DLP. However, there are some very interesting results in this direction, which work for special cases or under heuristic assumptions, and lead to satisfactory conclusions in practice.

140 Theorem (den Boer). Let G be an element of a group of prime order ℓ such that $\ell - 1$ is polynomially smooth.[21] Then DLP in the group $\langle G \rangle$ can be reduced to CDH in $\langle G \rangle$ in time polynomial in $\log(\ell)$.

The key idea underlying this reduction and the more general reduction below is that we get an implicit arithmetic "in the exponent" by computing in $\langle G \rangle$: Clearly it holds that $[a]G + [b]G = [a+b]G$ for all $a, b \in \mathbb{Z}$; this is an *addition* "in the exponent". Moreover, given a CDH oracle, we may additionally map $[a]G, [b]G$ to the shared secret $[ab]G$: This is a *multiplication* "in the exponent". We have a ring!

141 Definition. The *black-box field* $\boxed{\mathbb{F}_\ell}$ in $\langle G \rangle$ using a CDH oracle is given by the set $\langle G \rangle$ with arithmetic operations given by the group law for $+$ and the CDH oracle for $\cdot$. We write $\boxed{a}$ for the element $[a]G$ of $\boxed{\mathbb{F}_\ell}$.

[19] Fun fact: The famous 2010 hack of Sony's PlayStation 3 was made possible because of exactly this.

[20] The typical choice were "Schnorr groups": Subgroups of prime order ℓ inside an $\mathbb{F}_p^\times$ with $\ell \ll p$. Example: $\ell \approx 2^{256}$, $p \approx 2^{3072}$.

[21] This means: The ℓ here comes from an infinite set of primes for which there exists some fixed polynomial f such that for all $\ell \geq 2$ in the family, no prime factor of $\ell - 1$ is greater than $f(\log(\ell - 1))$.

In addition to the standard operations $+$ and $\cdot$ of a ring, we require some other functionality:

- Inversions in $\boxed{\mathbb{F}_\ell}$ can be computed using Fermat's little theorem (i.e., as $\boxed{a}^{-1} = \boxed{a}^{\ell-2}$), which unfortunately takes time $\Theta(\log \ell)$: a very significant overhead over the non-black-box setting.
- Squareness testing in $\boxed{\mathbb{F}_\ell}$ is similarly easy: For odd ℓ the Legendre symbol modulo ℓ is just $x \mapsto x^{(\ell-1)/2}$, which can be computed in time $\Theta(\log \ell)$.
- *Computing* a square root in $\boxed{\mathbb{F}_\ell}$ is feasible, too: If $\ell \equiv 3 \pmod 4$ we simply have $\sqrt{\boxed{a}} = \boxed{a}^{(\ell+1)/4}$. If on the other hand $\ell \equiv 1 \pmod 4$, then Cipolla's algorithm can be used: Sample $\boxed{s} \in \boxed{\mathbb{F}_\ell}$ at random until $\boxed{s}^2 - \boxed{a}$ is a nonsquare; then $\sqrt{\boxed{a}} = \big(\boxed{s} + X\big)^{(\ell+1)/2}$ in the polynomial quotient ring $\boxed{\mathbb{F}_\ell}[X]/\big(X^2 - \big(\boxed{s}^2 - \boxed{a}\big)\big)$ which is a black-box version of $\mathbb{F}_{\ell^2}$. In both cases, the cost is $\Theta(\log \ell)$.
- Encoding from $\mathbb{F}_\ell$ to $\boxed{\mathbb{F}_\ell}$ is easy: This means computing $a \mapsto \boxed{a} = [a]G$, which can be done in time $\Theta(\log(a \bmod \ell)) \subseteq O(\log \ell)$ using double-and-add.
- Decoding, however, is (generally) hard: This amounts to solving the DLP $(G, [a]G)$, for which the best generic algorithm takes expected time $\Theta(\sqrt{\ell})$.

However, there is a truly amazing trick: In order to recover a from $\boxed{a}$, one may perform arbitrary algebraic computations *in the black-box group* to obtain knowledge of relations that $\boxed{a}$ satisfies in $\boxed{\mathbb{F}_\ell}$ with respect to other, *known* elements, and recompute a from these relations "in the clear". The reduction of den Boer is a concrete (and the historically first) instantiation of this idea using the group $\mathbb{F}_\ell^\times$:

Proof of Theorem 140. We are given a DLP instance $(G, [a]G)$ and a CDH oracle. Proceed as follows:

(1) Find a generator g of the group $\mathbb{F}_\ell^\times$. Using the oracle, encode g into $\boxed{\mathbb{F}_\ell}$.

(2) Solve the DLP $\big(\boxed{g}, \boxed{a}\big)$ in the black-box unit group $\boxed{\mathbb{F}_\ell}^\times$ using generic algorithms.

(3) Let $x \in \mathbb{Z}/(\ell-1)$ be the solution. Recover $a \in \mathbb{Z}/\ell$ by computing g^x in $\mathbb{F}_\ell$.

By the assumption that $\ell - 1$ is polynomially smooth, generic DLP algorithms take polynomial time for the black-box DLP encountered in the reduction. In particular, the number of CDH oracle queries (i.e., multiplications in the black-box field) is also polynomial. □

142 Remark. The primary optimization goal for the generic DLP solver here is to minimize the number of CDH oracle calls (i.e., black-box multiplications). As such, one may happily trade a few oracle calls for huge amounts of "offline" computation, which skews the picture of what the "best" algorithm is in this setting.

The reduction of den Boer is neat, but as stated it is inherently limited to rare special cases where $\ell - 1$ is smooth. It has been suggested to specifically choose ℓ such that this reduction works and therefore the security of Diffie–Hellman could be proven to rest on the hardness of DLP itself, but this is by far not mainstream, partly since a more general reduction is available!

143 Theorem (Maurer). Let G be an element of a group of prime order ℓ. Suppose given an elliptic curve E over $\mathbb{F}_\ell$ such that $\#E(\mathbb{F}_\ell)$ is polynomially smooth. Then DLP in the group $\langle G \rangle$ can be reduced to CDH in $\langle G \rangle$ in time polynomial in $\log(\ell)$.

Proof. This works similarly as before, with some slightly tricky details:

(1) Find a generator P of $E(\mathbb{F}_\ell)$ and encode it into $E\big(\boxed{\mathbb{F}_\ell}\big)$, giving $\boxed{P}$.

(2) Sample $r \in \mathbb{F}_\ell$ randomly until $\boxed{a} + \boxed{r}$ is the x-coordinate of a point $\boxed{Q}$ in $E\big(\boxed{\mathbb{F}_\ell}\big)$.

(3) Solve the black-box ECDLP $\big(\boxed{P}, \boxed{Q}\big)$ in $E\big(\boxed{\mathbb{F}_\ell}\big)$, giving a solution $x \in \mathbb{Z}/\#E(\mathbb{F}_\ell)$.

(4) Recover Q by computing $[x]P$ in $E(\mathbb{F}_\ell)$. Find $a + r$ as the x-coordinate of Q, and thus a.

As before, the expected cost (in particular, number of CDH oracle calls) of this procedure is polynomial in $\log(\ell)$ since $\#E(\mathbb{F}_\ell)$ was assumed to be polynomially smooth. □

144 Remark. Given some prime ℓ it is not generally easy to *find* such a curve E. As such, the asymptotic complexity of the reduction is superpolynomial (but subexponential) when E is not given.

145 Remark. Replacing $\mathbb{F}_\ell^\times$ by some other algebraic group over $\mathbb{F}_\ell$, with elliptic curves in particular being a very useful choice, is a recurring theme in algebraic algorithms: It shows up in topics such as primality proving (ECPP) and integer factorization (ECM).

146 Remark. Note that the construction of the "auxiliary curve" E for a given ℓ can be done as a once-and-for-all precomputation. Various cryptographers have spent time on finding "good" such curves for cryptographically relevant group orders ℓ: As a recent example, May & Schneider in 2023 computed such auxiliary curves for a list of standard (or de-facto standard) elliptic curves, including $\approx 2^{37}$-smooth options for *Curve25519* and `secp256k1`.

By comparison, note that standard estimates for smoothness probabilities indicate that we should expect there to exist a 700-smooth number within the Hasse interval for the 256-bit group size of these curves.

4.10 The *decisional* Diffie–Hellman problem

147 Definition. The *decisional Diffie–Hellman problem* (DDH) asks, given a "Diffie–Hellman triple" (A, B, S), whether S is the correct shared secret for the Diffie–Hellman public keys A and B or not.

148 Remark. Decisional versions of computational problems are very common in cryptography: The property that *one cannot even tell the solution apart from a non-solution* reflects the idea that learning *anything at all* about the solution is supposed to be difficult. Hence, this rules out all kinds of partial attacks on secret data.

> *"The Decision Diffie–Hellman assumption (DDH) is a gold mine."*
>
> — Boneh: The decision Diffie–Hellman problem (1998)

Note that the apparent hardness of DDH is what causes the failure of the basic Diffie–Hellman identification scheme (Protocol 123) to be public-coin: If DDH was solvable, one could simply sample the challenge from the public-key space at random without knowing the corresponding secret and solve DDH to check. Well… There is a way to do this using pairings!

———— Tenth lecture (July 1)

5 Pairings

…are another cryptographic building block obtained from elliptic curves.

149 Definition. Consider three groups $(G_1, +), (G_2, +), (G_T, \cdot)$.[22] For our purposes, a *pairing* is a map

$$e\colon\ G_1 \times G_2 \longrightarrow G_T$$

which is:

- Bilinear: For all $P, P' \in G_1$ and $Q, Q' \in G_2$, the identities $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$ and $e(P, Q + Q') = e(P, Q) \cdot e(P, Q')$ hold.
- Non-degenerate: If $e(P, Q) = 1$ for all Q, then $P = 0$, and if $e(P, Q) = 1$ for all P, then $Q = 0$.

For e to be a *cryptographic* pairing, we additionally require that G_1, G_2, G_T have prime order ℓ, and that…

- the pairing is efficiently computable (say, in polynomial time).

[22] The convention to write G_1, G_2 additively while G_T is written multiplicatively is because G_1, G_2 will be subgroups of an elliptic curve while G_T will be a subgroup of $(\mathbb{F}_q^\times, \cdot)$ for some q.

- DLP in all three groups G_1, G_2, G_T is (conjectured to be) hard.
- inverting the pairing is (conjectured to be) hard.

  The *pairing inversion problem* is, given $z \in G_T$ and $P \in G_1$ (or $Q \in G_2$), to find $Q \in G_2$ (or $P \in G_1$) such that $e(P, Q) = z$ is satisfied.

Note that the definition of bilinearity simplifies to $e([a]P, [b]Q) = e(P, Q)^{ab}$ and that of non-degeneracy simplifies to $e \neq 1$ in the case of prime-order groups.

Also note that $G_1 \cong G_2$ since both are cyclic of order ℓ, but a priori the obvious way of evaluating such an isomorphism would involve computing discrete logarithms, which is supposed to be hard. However, there are cases where we can compute homomorphisms between G_1 and G_2 efficiently. Cryptographic pairings are commonly classified according to what is possible in that regard:

150 Definition. The literature refers to the following types of pairings:

- **Type 1**: $G_1 = G_2$. (This is referred to as a "symmetric" pairing.)
- **Type 2**: $G_1 \neq G_2$ but there is an efficiently computable group homomorphism $G_2 \to G_1$.
- **Type 3**: $G_1 \neq G_2$ and there are no efficiently computable group homomorphisms between G_1 and G_2.

Before we get into the mathematical and algorithmic details of pairings, let's look at the application I promised earlier.

5.1 Signatures from pairings

Recall that the security of the basic Diffie–Hellman identification scheme (Protocol 123) rests on the hardness of the *computational* Diffie–Hellman (CDH) assumption. In Section 4.10 it was pointed out that the scheme could be made public-coin, hence give rise to a signature scheme via Fiat–Shamir, *if only* DDH could be solved.

151 Definition. A *gap Diffie–Hellman group* is one in which CDH is (apparently) hard whereas DDH is easy.

152 Lemma. *Type-1 pairings give rise to gap groups.*

Proof. To solve DDH with input $\big([a]P, [b]P, S\big)$, we can test the property $e(P, S) \stackrel{?}{=} e\big([a]P, [b]P\big)$. □

This resolves the issue that the verifier in Protocol 123 was required to keep a secret; the DDH solver relieves them of this duty:

153 Boneh–Lynn–Shacham's identification scheme.
Assumption: $\langle G\rangle$ has a type-1 pairing e, and we have an efficient hash function $H\colon \{0,1\}^* \to \langle G\rangle$.
Situation: The *prover* has a DH key pair (a, A) where $A = [a]G$.

(1) **Verifier:** Choose a random *seed* $m \in \{0,1\}^*$ and let $R := H(m)$. Send R to the prover.

(2) **Prover:** Compute the Diffie–Hellman shared secret determined by A and R, i.e., the point $S := [a]R$, and send it to the verifier.

(3) **Verifier:** Check that $e(G, S) = e(A, R)$.

154 Remark. It is necessary for security that H obtains an element of $\langle G\rangle$ somehow "directly"; in particular, this can *not* be done by picking a random $r \in \{0, ..., \ell-1\}$ and computing $[r]G$.

Note that random sampling from $\langle G\rangle$ directly is not always feasible; It depends on the specifics of how the group is realized computationally.

155 Remark. The *BLS signature scheme* is the result of applying the Fiat–Shamir transform to this (now rendered public-coin) identification protocol: The only difference is that m becomes the message to be signed, rather than a random value.

5.2 Computing pairings on elliptic curves

We will now discuss how to build pairings from elliptic curves, first in theory, then in practice.

5.2.1 Divisors of functions

Note that all of the things in this section can be generalized to other types of algebraic varieties; we shall stick to elliptic curves to keep things concrete.

156 Definition. Let E be an elliptic curve. A *divisor on E* is a *finite formal $\mathbb{Z}$-linear combination of points on E*. Concretely, its data is a function $n\colon E \to \mathbb{Z}$ such that the set $n^{-1}(\mathbb{Z}\backslash\{0\})$, its *support*, is finite. We usually write divisors using the notation $D = n_1 \cdot (P_1) + \cdots + n_r \cdot (P_r)$ where each $P_i \in E$ and each $n_i \in \mathbb{Z}$. The *degree* of D is the sum $n_1 + \cdots + n_r$.

The main purpose of divisors is to encode "signed multisets" of points. In particular, they serve as a convenient notation for representing data like the order of vanishing of a function:

157 Definition. Let E be a (long) Weierstraß elliptic curve. Given a point $P \in E$, we call the following functions u_P on E the *uniformizer* at P:

- For $P = \infty$, we let $u_P = x/y$.
- For $P = (x_0, y_0)$ with $P \neq -P$, we let $u_P = x - x_0$.
- For $P = (x_0, y_0)$ with $P = -P$, we let $u_P = y - y_0$.

158 Remark. The general definition of uniformizer is non-unique, so speaking of "the" uniformizer is a slight abuse of terminology. (Uniformizers are also called "local parameters".)

159 Definition. Let f be a function on E, e.g., a rational expression in terms of x, y, and let $P \in E$. After possibly replacing f by a different description (cf. Definition 60), we may assume that it maps P to some value $f(P) \in \mathbb{P}^1$. If $f(P) \neq \infty$, we let $\nu_P(f)$ denote the smallest positive integer v such that $(u_P^{-v} f)(P) \neq 0$. In the case $f(P) = \infty$, we let $\nu_P(f) = -\nu_P(1/f)$.

The *divisor* of f is

$$\operatorname{div}(f) := \sum_{P \in E} \nu_P(f) \cdot (P)\,.$$

(In particular, the map $P \mapsto \nu_P(f)$ is zero almost everywhere.)

160 Lemma. *Divisors of functions have degree zero.*

161 Remark. The "correct" definition of the group law on an elliptic curve is phrased in terms of divisors: Two divisors are *equivalent* if they differ by a *principal divisor*, i.e., a divisor of some function. The set of equivalence *classes* of degree-zero divisors on E is denoted by $\operatorname{Pic}^0(E)$. Then, the map

$$\begin{aligned} E &\longrightarrow \operatorname{Pic}^0(E), \\ P &\longmapsto \big[(P) - (\infty)\big] \end{aligned}$$

is a bijection, and pulling back the obvious group law (formal adding) from $\operatorname{Pic}^0(E)$ to E recovers a group law on the points of E. Indeed, all principal divisors are sums of divisors of *lines*, which is how our original description of the group law (Theorem 30) relates to this definition.

One advantage is that this definition immediately generalizes to the group law on Jacobians of higher-genus curves; in that case, however, the map above is not surjective, which is why the group law really only works on the Jacobian and cannot be pulled back from Pic^0 to the curve points.

162 Definition/Lemma/Theorem (it depends). Consider some points $P_1, \ldots, P_n$ on an elliptic curve. Then $P_1 + \cdots + P_n = \infty$ holds in the elliptic-curve group if and only if the divisor $(P_1) + \cdots + (P_n) - n(\infty)$ is principal.

163 Definition. Let E be an elliptic curve, D a divisor on E, and f a function on E whose divisor has support disjoint from D. Writing $D = \sum_{i=1}^{r} n_i \cdot (P_i)$, we define

$$f(D) := \prod_{i=1}^{r} f(P_i)^{n_i} .$$

(This generalizes the definition of function evaluation to the group of divisors.)

164 Theorem (Weil reciprocity). Let E be an elliptic curve and f, g functions on E whose divisors have disjoint support. Then $f(\operatorname{div} g) = g(\operatorname{div} f)$.

5.2.2 The Weil pairing

165 Definition. Let E be an elliptic curve over a field K and $n \in \mathbb{Z}_{\geq 1}$. Let μ_n denote the subgroup of n^{th} roots of unity in $\overline{K}^{\times}$. The *order-n Weil pairing*

$$e_n \colon \quad E[n] \times E[n] \longrightarrow \mu_n$$

is defined as follows: Given two points $P, Q \in E[n]$, let D_P and D_Q denote two divisors with disjoint support and such that $D_P \sim (P) - (\infty)$ and $D_Q \sim (Q) - (\infty)$. Choose two functions f_P and f_Q on E with divisors $n \cdot D_P$ and $n \cdot D_Q$ respectively. Then

$$e_n(P, Q) \;=\; \frac{f_Q(D_P)}{f_P(D_Q)} .$$

166 Theorem. *The Weil pairing is well-defined, bilinear, non-degenerate unless n is divisible by the characteristic, and antisymmetric.*

Proof. A key part of the proof is that a function having a given divisor is unique up to scaling.

To show that e_n maps to μ_n, notice that

$$e_n(P, Q)^n = \frac{f_Q(D_P)^n}{f_P(D_Q)^n} = \frac{f_Q(n \cdot D_P)}{f_P(n \cdot D_Q)} \underset{\substack{\uparrow \\ \text{(Weil reciprocity)}}}{=} \frac{f_P(n \cdot D_Q)}{f_P(n \cdot D_Q)} = 1 .$$

Now for independence from the choice of D_P, D_Q: Let D'_P be another divisor equivalent to $(P) - (\infty)$ and f'_P a function with divisor $n \cdot D'_P$. Then $D'_P = D_P + \operatorname{div}(h)$ for some function h, and since a function with given divisor is essentially unique we may assume $f'_P = \lambda h^n f_P$ for some $\lambda \in \overline{K}^{\times}$. Notice that $(\lambda f)(D) = f(D)$ for any degree-0 divisor D. Hence

$$\frac{f_Q(D'_P)}{f'_P(D_Q)} = \frac{f_Q(D_P) \cdot f_Q(\operatorname{div}(h))}{(\lambda f_P)(D_Q) \cdot h^n(D_Q)} = \frac{f_Q(D_P)}{f_P(D_Q)} \cdot \frac{f_Q(\operatorname{div}(h))}{h^n(D_Q)} = e_n(P, Q) \cdot \underbrace{\frac{f_Q(\operatorname{div}(h))}{h(n \cdot D_Q)}}_{\substack{=1 \\ \text{(Weil reciprocity)}}} .$$

The same argument works for changing D_Q. Bilinearity is "obvious" since $D \mapsto f(D)$ is homomorphic from the divisor group to $\overline{K}^{\times}$ where defined, and since $f_{D+D'} = \lambda f_D f_{D'}$ for some constant $\lambda \in \overline{K}^{\times}$. Antisymmetry is obvious by looking at the formula. We'll skip the nondegeneracy proof. □

Eleventh lecture (July 8)

One very useful interpretation of the Weil pairing is that it is essentially a "hidden" determinant:

167 Lemma. *Let P, Q two points in $E[n]$, for instance forming a basis. Write $\zeta := e_n(P, Q)$. Then*

$$e_n([a]P + [b]Q, [c]P + [d]Q) \;=\; \zeta^{\det \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)} .$$

Proof. Using bilinearity and antisymmetry,

$$\begin{aligned} e_n([a]P + [b]Q, [c]P + [d]Q) &= e_n(P,P)^{ac} \cdot e_n(P,Q)^{ad} \cdot e_n(Q,P)^{bc} \cdot e_n(Q,Q)^{bd} \\ &= 1 \cdot \zeta^{ad} \cdot \zeta^{-bc} \cdot 1 = \zeta^{ad-bc} = \zeta^{\det\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)}. \end{aligned}$$

□

The main application in pairing-based cryptography of the Weil pairing is to construct a type-1 pairing; thus, we would like to restrict it to a cyclic subgroup. However, due to antisymmetry, the pairing is actually constant on cyclic subgroups: From $e_n(P,P) = e_n(P,P)^{-1}$ it follows that $e_n(P,P) = 1$. To turn the Weil pairing into a type-1 pairing, we need some more work. The missing ingredient is given by *distortion maps*: Endomorphisms of E taking points from a certain cyclic subgroup to independent points.

168 Definition. Two points $P, Q \in E[n]$ are *independent* in the n-torsion if $[u]P + [v]Q = \infty$ with $u, v \in \mathbb{Z}$ implies that $u, v \in n\mathbb{Z}$. In that case, we say that P, Q is an *n-torsion basis*.

169 Lemma. *Two points $P, Q \in E[n]$ form an n-torsion basis if and only if $e_n(P,Q)$ has multiplicative order n.*

Proof. (Homework.) □

170 Definition. Consider $n \geq 1$ and $G \in E$ of order n. We say that ϑ is a *distortion map* (for the group $\langle G \rangle$) if the pair $\big(G, \vartheta(G)\big)$ forms a basis of the n-torsion subgroup $E[n]$.

Given a distortion map, a type-1 pairing is indeed readily constructed from the Weil pairing:

171 Lemma. *Let ϑ be a distortion map for a cyclic group $\langle G \rangle$ of order ℓ. Then*

$$\widehat{e} \colon \langle G \rangle \times \langle G \rangle \to \mu_\ell,\ (P,Q) \mapsto e_\ell(P, \vartheta(Q))$$

is a type-1 pairing.

Proof. Clearly $\widehat{e}$ is bilinear since e_ℓ is bilinear and ϑ is linear.

For non-degeneracy, write $\zeta = \widehat{e}(G,G)$. Then $\widehat{e}([a]G, [b]G) = \widehat{e}(G,G)^{ab}$; hence, if there exists any $a \neq 0$ such that $\widehat{e}(G,G)^{ab} = (\widehat{e}(G,G)^a)^b$ does not vary with b, then $\widehat{e}(G,G) = 1$. (Same for a, b swapped.)

It remains to prove that $\widehat{e}(G,G) = e_\ell(G, \vartheta(G)) \neq 1$. To do so, let $T := \vartheta(G)$. By Lemma 169, the value $\zeta := e_\ell(G,T) \neq 1$ is nonzero. Hence $\widehat{e}(G,G) = \zeta \neq 1$. □

How does one construct such a distortion map?

172 Example. Let $E \colon y^2 = x^3 + x$ over a field of characteristic $p \geq 3$ and let $\ell \equiv 3 \pmod 4$ be prime. Then the automorphism $\iota \colon (x,y) \mapsto (-x, \sqrt{-1} \cdot y)$ is a distortion map for all subgroups of order ℓ.

Proof. (Homework.) □

173 Remark. This example generalizes easily: The standard method for constructing supersingular elliptic curves (Bröker's algorithm) outputs curves over $\mathbb{F}_p$ together with a small-degree irrational endomorphism.

5.2.3 The embedding degree

Notice that, in order to have any hope of using the Weil pairing in a computationally explicit manner, we will have to work with points in the n-torsion subgroup $E[n]$. Recall from Theorem 58 that this group is usually isomorphic to $\mathbb{Z}/n \times \mathbb{Z}/n$, so the question is to which extension we have to go in order to find two independent points of order n.

174 Definition. Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and ℓ a prime dividing $\#E(\mathbb{F}_q)$ but not dividing q. The *embedding degree* of E with respect to ℓ is the smallest $k \geq 1$ such that $E[\ell] \subseteq E(\mathbb{F}_{q^k})$.

In other words, the embedding degree is the degree of the extension required to access the second component of the ℓ-torsion, which is necessary (and sufficient) to obtain a nontrivial order-ℓ Weil pairing. A convenient characterization is the following:

175 Lemma. *In the situation of Definition 174, for any $k \geq 1$, we have $E[\ell] \subseteq E(\mathbb{F}_{q^k})$ if and only $\ell \mid (q^k - 1)$.*

Proof. (The "$\Rightarrow$" direction is homework.)

For the other direction, suppose $\ell \mid (q^k - 1)$. It suffices to show that the q^k-power Frobenius endomorphism π_{q^k} of E restricts to the identity on $E[\ell]$. By assumption, there exists a cyclic subgroup $\langle P \rangle \subseteq E$ of order ℓ where π_q acts trivially. Let Q be a point in $E[\ell]$ that is independent from P.

There exist (unique) $u, v \in \mathbb{F}_\ell$ such that $\pi_q(Q) = [u]P + [v]Q$. Let $Q' := \pi_q(Q) - Q = [u]P + [v-1]Q$. If $Q' = \infty$, then $\pi_q(Q) = Q$ and we are done (for any k). Otherwise, $v \neq 1$, thus P, Q' are independent, and

$$\pi_q(Q') = \pi_q([u]P + [v-1]Q) = [u]P + [u(v-1)]P + [v(v-1)]Q = [uv]P + [v^2 - v]Q = [v]Q' .$$

Recalling from Lemma 71 that $\pi_q^2 - t\pi_q + q = 0$ where $t = \operatorname{tr}(\pi_q) = q + 1 - \#E(\mathbb{F}_q)$, we may plug Q' into this equation:

$$0 = \pi_q^2(Q') - [t]\pi_q(Q') + [q]Q' = [v^2 - tv + q]Q'$$

In consequence, $0 = v^2 - tv + q = (v-1)(v-q)$, where the factorization works because it was assumed that $\ell \mid \#E(\mathbb{F}_q) = q + 1 - t$ and thus $t \equiv q + 1 \pmod{\ell}$. This shows $v = q$.

In conclusion, the action of π_q on our ℓ-torsion basis (P, Q') is given by $\pi_q(P) = P$ and $\pi_q(Q') = [q]Q'$. In particular, $\pi_{q^k}(Q') = Q'$ if and only if $q^k \equiv 1 \pmod{\ell}$, as claimed. □

176 Remark. In the following, keep in mind that writing down $E[n]$ requires passing to an extension whose degree is generically very large — but it is possible to force it to be much smaller. We'll defer the question how to construct *pairing-friendly* curves to later; for now, everything will simply be done in the algebraic closure.

5.2.4 The multiplicative transfer attack

The historically first use of pairings in cryptography (Menezes–Okamoto–Vanstone, 1991) was actually destructive rather than constructive. It allows to *reduce ECDLP to a finite-field DLP*, where the order of the finite field is (typically) controlled by the *embedding degree* of the curve.

177 Remark. There are examples where the resulting DLP lands in a subfield whose degree over the base field is even less than the embedding degree. We will ignore this detail for simplicity.

178 The MOV attack.

Input: An elliptic curve $E/\mathbb{F}_q$ and an order-ℓ ECDLP instance (G, A) on E.
Output: A finite field $\mathbb{F}_{q^k}$ and two elements $\zeta, \omega \in \mathbb{F}_{q^k}$ such that ζ has order ℓ and $\zeta^a = \omega$ where $[a]G = A$.

(1) Find the embedding degree $k \geq 1$ of E for the subgroup $\langle G \rangle$.

(2) Compute a point $T \in E(\mathbb{F}_{q^k})$ such that (G, T) form a basis of the ℓ-torsion.

(3) Return $\zeta := e_\ell(G, T)$ and $\omega := e_\ell(A, T)$.

The key benefit for an attacker is that, as we've seen in Section 1.4, the complexity of DLP in a finite field is (asymptotically as well as practically) much lower than the complexity of ECDLP. Hence, this attack puts a big dent in the security of curves *with small embedding degree*. The workaround for non-pairing curves is simply not to make an intentionally bad choice. The solution for curves which are *intended* to be "pairing-friendly" is to balance the size of the base field with the embedding degree in order to render the costs of ECDLP on the curve and of DLP in the finite field about equal.

5.2.5 Computing pairings via Miller functions

As seen in Definition 165, the core task in computing the Weil pairing (and this is also true for other types of pairings) is to evaluate a function with a given divisor at some point. Indeed, for cryptographic sizes, the functions being evaluated here are exponentially large, so it may a priori seem difficult to even write them down. Luckily, one can get around this issue by using specific functions which can be written as a product which is much more compact than the expanded polynomial expression. This is due to Miller (1986).

179 Remark. In general, I'll be talking about "the" function with a given divisor. This is sloppy: In order to make it strictly rigorous we should also control the "leading coefficients" of all functions. I will generally sweep this issue under the rug.

180 Definition. Let $P \in E$ be a point and $k \geq 1$ an integer. "The" k^{th} *Miller function* associated to the point P is a function $f_{k,P}$ with divisor $k(P) - ([k]P) - (k-1)(\infty)$. (Recall that this is unique up to scaling.)

Notice that $\operatorname{div}(f_{n,P}) = n(P) - n(\infty)$, which looks like the divisor we need for the Weil pairing. The $-([k]P)$ term is introduced so that the divisor becomes principal; note that there is no function with divisor $k(P) - k(\infty)$ unless $[k]P = \infty$. These types of functions are moreover particularly convenient since they lend themselves to being decomposed as a product in square-and-multiply style:

181 Lemma. *Let E be a Weierstraß curve. For two points $P, Q \in E$, let $L_{P,Q}$ denote the line through P and Q; as usual this is taken to refer to a tangent in case $P = Q$. Then the function $g_{P,Q} := L_{P,Q}/L_{P+Q,-(P+Q)}$ has divisor*

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

Proof. Recall from the definition of the group law (Theorem 30) that the line $L_{P,Q}$ intersects the curve in the three points P, Q, and $-(P+Q)$. Since principal divisors have degree zero, this must be balanced by a triple pole at infinity. Similarly for $L_{P+Q,-(P+Q)}$. Hence

$$\begin{aligned}\operatorname{div}(L_{P,Q}) &= (P) + (Q) + (-(P+Q)) - 3(\infty)\\ \operatorname{div}(L_{P+Q,-(P+Q)}) &= (P+Q) + (-(P+Q)) + (\infty) - 3(\infty)\end{aligned}$$

and therefore

$$\operatorname{div}(g_{P,Q}) = \operatorname{div}(L_{P,Q}) - \operatorname{div}(L_{P+Q,-(P+Q)}) = (P) + (Q) - (P+Q) - (\infty)\,.$$

□

182 Lemma. *Let P be a point of order n on a Weierstraß curve E For any $k \geq 1$, let $f_{k,P}$ denote the k^{th} Miller function for P, i.e., a function with divisor $k(P) - ([k]P) - (k-1)(\infty)$. Then the $f_{k,P}$ satisfy the recursion*

$$f_{k+k',P} = f_{k,P} \cdot f_{k',P} \cdot g_{[k]P,[k']P}\,.$$

Proof. Plugging in the known divisors of Miller functions and of $g_{P,Q}$, we get

$$\begin{aligned}\operatorname{div}(f_{k+k',P}) - \operatorname{div}(f_{k,P}) - \operatorname{div}(f_{k',P}) &= (k+k')(P) - ([k+k']P) - (k+k'-1)(\infty)\\ &\quad -k(P) + ([k]P) + (k-1)(\infty)\\ &\quad -k'(P) + ([k']P) + (k'-1)(\infty)\\ &= ([k]P) + ([k']P) - ([k+k']P) - (\infty)\\ &= \operatorname{div}(g_{[k]P,[k']P})\,.\end{aligned}$$

□

Twelfth lecture (July 15)

183 Miller's algorithm.
Input: Two points P, Q on an elliptic curve E; the order $n \geq 1$ of P.
Output: The result of the Miller function $f_{n,P}$ evaluated at Q.

(1) Compute the binary expansion $n = \sum_{i=0}^{\ell-1} b_i 2^i$ with each $b_i \in \{0, 1\}$.

(2) Initialize $f := 1$ and $V := P$.

(3) For k ranging from $\ell - 1$ down to 0:

  (a) Set $f := f^2 \cdot g_{V,V}(Q)$ and $V := [2]V$.

  (b) If $b_k = 1$: Set $f := f \cdot g_{V,P}(Q)$ and $V := V + P$.

(4) Return f.

As usual, the loop invariant is that f holds the value $f_{\lfloor n/2^k \rfloor, P}(Q)$ at the end of the main loop. Similarly for V. Clearly, this algorithm runs in time $\Theta(\ell) = \Theta(\log(n))$.

184 Remark. A key idea in Miller's algorithm is to include the evaluations at Q inside the algorithm, so that the (very big) functions $f_{k,P}$ are never explicitly computed in full: Only their evaluations at Q appear.

Now, let us concretize the definition of the Weil pairing: While the definition is phrased in terms of any divisors in the correct class, one uses a particular shape of divisor in practice:

185 Lemma. *Let $P, Q \in E[n]$. Find a point $T \in E$ such that $T \notin \{\infty, P, Q, Q - P\}$. Then*

$$e_n(P, Q) = \frac{f_{n,Q}(P + T) f_{n,P}(-T)}{f_{n,P}(Q - T) f_{n,Q}(T)}.$$

Proof. Clearly all individual evaluations can be done without hitting a zero or pole: Both $P + T = Q$ and $Q - T = P$ imply $T = Q - P$, which was excluded.

Thus, we need to check that the result is correct. Indeed, from the definition with $D_P = (P + T) - (T)$ and $D_Q = (Q) - (\infty)$, we have

$$e_n(P, Q) = \frac{f_{n,Q}(P + T)/f_{n,Q}(T)}{f_1(Q)/f_1(\infty)}$$

where $\operatorname{div}(f_1) = n(P + T) - n(T)$. But f_1 is just $f_{n,P}$ with a translation by $-T$ applied to the input, hence $f_1(Q)/f_1(\infty) = f_{n,P}(Q - T)/f_{n,P}(-T)$. □

186 Corollary. *Let $P, Q \in E[n]$ be distinct. Then*

$$e_n(P, Q) = (-1)^n \frac{f_{n,Q}(P)}{f_{n,P}(Q)}.$$

Proof sketch. The idea is to let $T \to \infty$ in the previous expression, which can be made rigorous by examining the formal expansion.

Put this way, the computation of the Weil pairing requires two "Miller loops". However, over the course of proving Theorem 166, we've already seen that even the simpler map $(P, Q) \mapsto f_P(Q)$ already enjoys bilinearity! So, is there a way to create a cryptographic pairing from this?

5.3 The Tate pairing

Recall that f_P is a function with divisor $n(P) - n(\infty)$ and that D_Q is a divisor equivalent to $(Q) - (\infty)$. (We'll silently assume that the supports of D_Q and $(P) - (\infty)$ are disjoint whenever needed.)

187 Theorem. *Let $\mathbb{F}_q$ be a finite field of characteristic p and E an elliptic curve over $\mathbb{F}_q$. Let $n \in \mathbb{Z}_{\geq 1}$ not divisible by p and suppose $\mu_n \subseteq \mathbb{F}_q$. Then*

$$\begin{aligned} e\colon\ E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/[n]E(\mathbb{F}_q) &\longrightarrow \mathbb{F}_q^\times/(\mathbb{F}_q^\times)^n \\ (P, \overline{Q}) &\longmapsto f_P(D_Q) \end{aligned}$$

is a nondegenerate bilinear pairing.

188 Remark. The "interface" of this pairing is significantly less "clean" than for the Weil pairing: In particular, the second input and the result are *cosets* of their respective groups *modulo n^{th} multiples* rather than elements.

The motivation for quotienting by n^{th} multiples is as follows: Since the pairing is to be bilinear, we require $e(P, [n]Q) = e([n]P, Q) = e(\infty, Q) = 1$. Thus we should quotient the second input by n^{th} multiples. But then the output varies with the choice of coset representative for the input, so we also have to quotient out the subgroup $e(P, [n]E(\mathbb{F}_q)) = e(P, E(\mathbb{F}_q))^n = (\mathbb{F}_q^\times)^n$ in the target group $\mathbb{F}_q^\times$. (The last equality relies on nondegeneracy.)

Proof of Theorem 187. Everything follows from the same type of reasoning as for the Weil pairing. □

In many applications, we do need to compute a standardized representative of the result of the pairing. The easiest way of doing so is to apply the isomorphism $\mathbb{F}_q^\times/(\mathbb{F}_q^\times)^n \xrightarrow{\sim} \mu_n$ which is given by exponentiation:

189 Definition. Let e denote the order-n Tate pairing. The *reduced Tate pairing* is

$$\begin{aligned} \widehat{e}\colon\ E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/[n]E(\mathbb{F}_q) &\longrightarrow \mu_n \\ (P, \overline{Q}) &\longmapsto e(P, \overline{Q})^{(q-1)/n}. \end{aligned}$$

The exponentiation to the power of $(q-1)/n$ is commonly called the *final exponentiation*.

190 Remark. The final exponentiation is also necessary for security: There exist examples where the pairing would be *insecure* if the exact representative of the output computed by Miller's algorithm was revealed!

191 Remark. The word "reduced" is often omitted when referring to the reduced Tate pairing.

In practice, to recover the standard framework of a cryptographic group action $G_1 \times G_2 \to G_T$ with $G_1 \cong G_2 \cong G_T \cong \mathbb{Z}/\ell$, it is common for one of the inputs to lie in the base field of the curve, while the other lies in an extension to the embedding degree (Definition 174). Depending on the situation, one may restrict the field of definition of either input. It is therefore also common to write the (reduced) Tate pairing as a map

$$E(\mathbb{F}_{q^k})[n] \times E(\mathbb{F}_q)/[n]E(\mathbb{F}_q) \longrightarrow \mu_n \leq \mathbb{F}_{q^k}^\times$$

or

$$E(\mathbb{F}_q)[n] \times E(\mathbb{F}_{q^k})/[n]E(\mathbb{F}_{q^k}) \longrightarrow \mu_n \leq \mathbb{F}_{q^k}^\times$$

where k is the embedding degree.

5.3.1 A neat optimization

In this case, the final exponentiation can be sped up significantly using the fact that the exponent is of the special form $(q^k-1)/n$: The key lies in the fact that X^k-1 factors as a product of the cyclotomic polynomials $\Phi_d \in \mathbb{Z}[X]$ for $d \mid k$, and that $n \mid \Phi_k(q)$ by construction (Definition 174). Therefore we get

$$\alpha^{(q^k-1)/n} = \left(\alpha^{F(q)}\right)^{\Phi_k(q)/n}$$

where $F(X) = (X^k-1)/\Phi_k$. Now, the polynomials $F(X)$ tend to have coefficients ± 1 and be fairly sparse. For example, for $k=12$ we get $F(X) = X^8 + X^6 - X^2 - 1$.

Why does this help? The key observation is that *the q-power Frobenius is $\mathbb{F}_q$-linear*. Therefore, we may (once and for all) precompute the action of $\pi\colon\ \alpha \mapsto \alpha^q$ on the fuinite field $\mathbb{F}_{q^k}$ as a $k \times k$ matrix over $\mathbb{F}_q$, then evaluate $\alpha^{F(q)}$ as a short and simple product of some powers of Frobenius applied to α. Example: For $k=12$, the Frobenius acts on a $\mathbb{F}_q$-basis of $\mathbb{F}_{q^k}$ as a matrix $M \in \mathbb{F}_q^{12\times 12}$, and we have

$$\alpha^{F(q)} = \frac{M^8(\alpha) \cdot M^6(\alpha)}{M^2(\alpha) \cdot \alpha}.$$

Hence, in those cases, the size of the "actual" remaining exponentation by $\Phi_k(q)/n$ can be reduced by a large factor compared to the entire exponentiation by $(q^k-1)/n$ naïvely. This improvement is more pronounced the more prime factors k has. Example comparison:

- Computing $\alpha^{(q^{12}-1)/n}$ naïvely takes about $\approx 12\log(q)$ multiplications in $\mathbb{F}_{q^{12}}$.
- Computing $\alpha^{F(q)}$ using matrix-vector products takes about $3 \cdot 12^3 \approx 5000$ multiplications in $\mathbb{F}_q$, followed by just a few operations in $\mathbb{F}_{q^{12}}$.
  (If $\mathbb{F}_{q^{12}}$ is implemented as naïve arithmetic, one multiplication in $\mathbb{F}_{q^{12}}$ costs about the same as 150 multiplications in $\mathbb{F}_q$.)
- Computing the final exponentiation of $\alpha^{F(q)}$ to the power of $\Phi_{12}(q)/n$ takes no more than $\approx 4\log(q)$ multiplications in $\mathbb{F}_{q^{12}}$.

Thus, as $q \to \infty$, this saves about $3/4$ of the total work for the final exponentiation.

5.4 Constructing pairing-friendly curves

Finally, let us discuss how to *find* pairing-friendly curves. We will mainly focus on one important family of such curves. First, some preliminaries, which apply to several known methods for constructing pairing-friendly curves.

Notation: We would like an elliptic curve defined over $\mathbb{F}_q$ with trace of Frobenius $t := \operatorname{tr}(\pi_q)$, such that $n := \#E(\mathbb{F}_q) = q+1-t$ factors as $n = h \cdot \ell$ with h small (perhaps $h=1$) and ℓ a large prime, and such that the embedding degree k for the order-ℓ subgroup is fairly small, but not *too* small. (The main example below will have $k=12$.)

First, recall from Lemma 175 that having embedding degree exactly k means $\ell \mid (q^k-1)$ but $\ell \nmid (q^d-1)$ for all smaller $d \mid k$. Since $q^k - 1 = \prod_{d\mid k} \Phi_d(q)$, this is equivalent to saying $\ell \mid \Phi_k(q)$. From the requirement $q+1-t = h\ell \equiv 0 \pmod{\ell}$, we get $\Phi_k(q) \equiv \Phi_k(t-1) \pmod{\ell}$.

The basic idea now is to find pairing-friendly curves by first choosing a suitably-sized t, looking for a suitable divisor $\ell \mid \Phi_k(t-1)$, and checking if the remaining conditions (existence of a h such that q is a prime power) hold. The tricky part is that constructing a curve with a given q and t is in general difficult: The only case where we know how to do this efficiently is when the fundamental discriminant of $\mathbb{Z}[\pi_q]$ is small, i.e., when $t^2 - 4q = dv^2$ with d a small (negative) integer, using the so-called CM method. Quite a bit of effort has gone into fabricating this situation, exploiting insights from algebraic number theory and the theory of diophantine equations. We will only discuss one particularly neat (and simple) special case below.

5.4.1 The Barreto–Naehrig construction

This family of elliptic curves from 2005 is actually given as a parameterized family: There are *polynomial expressions* for t, q, n in terms of some parameter X such that whenever q is a prime, it is straightforward to construct the associated pairing-friendly curve.

Here, we consider the fixed choice $k = 12$: This embedding degree is (still) a good choice given current performance estimates for finite-field DLP algorithms: For a 250-bit group, the size of the base field ends up being about 3000 bits, which exactly matches the BSI recommendation mentioned at the end of Section 1.4.

It is also very convenient because $\Phi_{12} = X^4 - X^2 + 1$, a remarkably low-degree polynomial given the magnitude of k.

In terms of sizes, if we want $\ell \approx q$, notice that $t \approx \sqrt{q} \approx \sqrt{n}$ from the Hasse bounds. hence we should have $\deg(n) = 2\deg(t)$. Recalling that $n \mid \Phi_k(t-1)$, we immediately see that $\deg(t) = 1$ is not a possible choice since Φ_{12} is irreducible. The next best thing is $\deg(t) = 2$, which turns out to work!

192 Lemma. $\Phi_{12}(6X^2) = (36X^4 + 36X^3 + 18X^2 + 6X + 1)(36X^4 - 36X^3 + 18X^2 - 6X + 1)$.

Proof. Calculation. □

From this, the BN construction follows quickly: We let

$$\begin{aligned} & t(X) := 6X^2 + 1 \\ & n(X) := 36X^4 + 36X^3 + 18X^2 + 6X + 1 \\ \rightsquigarrow \quad & q(X) := n(X) - 1 + t(X) = 36X^4 + 36X^3 + 24X^2 + 6X + 1\,. \end{aligned}$$

For these choices, if we can find a curve over $\mathbb{F}_q$ such that $\mathrm{tr}(\pi_q) = t$, then this curve will be pairing-friendly for order n by construction. We only first have to find some X where both $q(X)$ and $n(X)$ are primes, but this can (practically) be done using brute force.[23]

Finding the curve. The last thing to do is to find an elliptic curve with prescribed base field and Frobenius trace. As it turns out, in this particular case, this is particularly easy! The key ingredient is that

$$t(X)^2 - 4q(X) = -3(\underbrace{6X^2 + 4X + 1}_{=:u(X)})^2\,.$$

Hence, the Frobenius on such a curve can be embedded in the quadratic field $\mathbb{Q}(\sqrt{t^2 - 4q}) = \mathbb{Q}(\sqrt{-3})$.

But we know such a curve, namely $y^2 = x^3 + b$ with $b \in \mathbb{F}_q$. Indeed, this curve has an automorphism

$$\omega\colon\ (x, y) \mapsto (\zeta_3 x, y)$$

where ζ_3 is any primitive cube root of unity in $\overline{\mathbb{F}_q}$. If q is a prime satisfying $q \equiv 1 \pmod 3$, we can prove that $\pi_q \in \mathbb{Z}[\omega]$, so we want a curve with Frobenius $\pi_q = (t+u)/2 + u\omega$. As it turns out, exactly one of the six isomorphism classes of curves $y^2 = x^3 + b$ over $\mathbb{F}_q$ has precisely this Frobenius, so we can brute-force b.

5.5 Pairing optimizations

Cryptographic pairings were made practical using a plethora of cool tricks. Here's a non-exhaustive list of some of the techniques:

- Working on twists (cf. Lemma 107). For instance, for BN curves, it is possible to encode elements of $G_2 \leq E(\mathbb{F}_{q^{12}})$ as points in $\widetilde{E}(\mathbb{F}_{q^2})$, where $\widetilde{E}$ is a sextic twist of the curve.
- Convenient models for extension fields. For instance, in the setup above, it makes sense to build $\mathbb{F}_{p^{12}}$ as a *tower* of extensions $\mathbb{F}_p \subseteq \mathbb{F}_{p^2} \subseteq \mathbb{F}_{p^6} \subseteq \mathbb{F}_{p^{12}}$. It is also convenient to use simple defining polynomials for each intermediate extension, for example given by simply adjoining a root of the correct degree.
- Final exponentiation exploiting the linearity of Frobenius; see above in Section 5.3.
- Sparse constants. For example, in the final exponentiation, having fewer bits set in the scalar accelerates the square-and-multiply algorithm.[24]

The bottom line is that pairings on contemporary de-facto standard curves at a 128-bit security level can be evaluated with timings on the scale of ≈ one millisecond per pairing computation on standard hardware. (For comparison: Sage takes about 200 times longer for a generic Weil pairing for the same family of curves and pairing group order.)

[23] Recall that the density of primes in $\{1, \ldots, N\}$ is $\approx 1/\ln(N)$, hence this brute-force search is polynomial-time in the target size $\log(q)$ unless there is some mathematical reason why the polynomials $q(X)$ and $n(X)$ represent fewer primes. (There isn't.)

[24] Note that using naïve square-and-multiply is okay here since the scalar is not a secret!