A Short Note on Cryptography using Elliptic Curves, Bilinear Pairings and Lattices

Martin M. Lauridsen

Technical University of Denmark

February 13, 2016

1 Introduction

In this short note, we introduce the basic concepts of elliptic curves, bilinear parings and lattices. We describe how these objects from the world of mathematics, can be used to define cryptographic schemes. The *uses* we present are not new. For example, we present several ways to obtain secure public-key encryption schemes using elliptic curves, pairings and lattices. We have known how to do public-key encryption since the 70'es, namely using RSA. The reason why we care about the schemes presented in this note is, that they, in some sense, improve upon what RSA can offer us. Particularly, lattice-based encrypting using NTRU (described in Section 5.7) gives us a faster, more compact public-key encryption scheme than RSA, which on top of that is that it is secure against quantum algorithms. On a similar note, we describe e.g. a version of the well-known Diffie-Hellman protocol which uses the additive group of an elliptic curve, rather than the multiplicative group $\mathbb{Z}_p$ for a prime p, because the system is more compact (for the same security level).

In this note, we present several examples of the concepts and methods we introduce, as we go along. The examples will all involve Sage code, which, at the leisure of the reader, can be simply tried out and verified in a Sage terminal. Sage is a free and open-source mathematics software system. It offers an amazing range of capabilities, and you can download, install and run it for free in less than ten minutes. For more information, visit `http://www.sagemath.org/`. The reader is encouraged to take the time to set up a working installation of Sage before starting, such that code from the examples can simply be copied and pasted into the Sage terminal.

1.1 Notation

In this note, we use p for a prime number and denote $q = p^k$ for some integer k. We use $\mathbb{Z}, \mathbb{N}, \mathbb{R}$ and $\mathbb{Q}$ to denote the integers, natural numbers, real numbers and rationals, as usual. We use $\mathbb{F}_q$ to denote the unique finite field consisting of q elements. For a set S, we use $\sharp S$ to denote the size of the set. For a point P on an elliptic curve, we write $x(P)$ and $y(P)$ to mean the x and y coordinates of P, respectively. The special linear group of degree n over a field K is denoted $SL_n(K)$. It is the set of $n \times n$ matrices over K with determinant ± 1.

2 Elliptic Curves

An *elliptic curve* is a mathematical object which has certain nice properties. We can perform computations using the points that *lie on* an elliptic curve, and we can define problems using them that are *computationally hard*. This hardness can be used to obtain secure cryptosystems, as we shall see. Also, elliptic curves have applications in number theory, for example for factoring integers, an approach due to Lenstra, which we describe in Section 3.4.

Definition 1 (Generalized Weierstrass equation). *A* generalized Weierstrass equation *is a bivariate equation of the form*

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6. \tag{1}$$

The equation is defined over some field K, which means that the variables X, Y and the constants a_1, a_2, a_3, a_4 and a_6 are taken from K.

Definition 2 (Elliptic curve). *Let K be a field. An* elliptic curve *over K, denoted $E(K)$, is defined by a generalized Weierstrass equation of the form* (1) *for which the* discriminant $\Delta_E \neq 0$*, where*

$$\Delta_E = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6, \tag{2}$$

and

$$\begin{aligned} b_2 &= a_1^2 + 4a_2 \\ b_4 &= 2a_4 + a_1a_3 \\ b_6 &= a_3^2 + 4a_6 \\ b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2. \end{aligned} \tag{3}$$

The elliptic curve $E(K)$ is comprised of the solutions $(X, Y) \in K^2$ to the curve equation, together with a special point $\mathcal{O}$ called the point at infinity. *As such,*

$$E(K) = \{(X, Y) \in K^2 \mid Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6\} \cup \{\mathcal{O}\}. \tag{4}$$

The elliptic curves that shall be of interest to us are all defined over *finite* fields, i.e. fields with a finite number of elements. In particular, we shall henceforth (except for a few illustrative cases) always use $\mathbb{F}_q$ as the underlying field, where $q = p^k$ with p prime and $k \geq 1$. When q is prime, i.e. $q = p$, we sometimes write $\mathbb{F}_p$ directly to make this clear.

Example 1. Consider the elliptic curve over $\mathbb{Z}_{13}$ defined by $E : Y^2 = X^3+3X+8$, i.e. the generalized Weierstrass equation with $a_1 = a_2 = a_3 = 0$, $a_4 = 3$ and $a_6 = 8$. We first implement a function computing the discriminant:

```
def discriminant(a1, a3, a2, a4, a6):
    b2 = a1^2 + 4*a2
    b4 = 2*a4 + a1*a3
    b6 = a3^2 + 4*a6
    b8 = a1^2 * a6 + 4*a2*a6 - a1*a3*a4 + a2*a3^2 - a4^2
    return -b2^2 * b8 - 8*b4^3 - 27*b6^2 + 9*b2*b4*b6
```

and check that $\Delta_E \neq 0$:

```
sage: discriminant(0,0,0,3,8)
-29376
```

We now write a function, given a generalized Weierstrass equation, assuming the discriminant is non-zero and given a finite field K, determines the points on the curve (besides $\mathcal{O}$):

```
def points_on_curve(a1, a3, a2, a4, a6, K):
    # assumes K is a finite field
    points = []
    for x in K.list():
        for y in K.list():
            if y^2 + a1*x*y + a3*y == x^3 + a2*x^2 + a4*x + a6:
                points = points + [(x,y)]
    return points
```

We determine the points:

```
sage: points_on_curve(0,0,0,3,8,GF(13))
[(1, 5), (1, 8), (2, 3), (2, 10), (9, 6), (9, 7), (12, 2),
   (12, 11)]
```

2.1 The Group Law: Addition of Points

When we talk about elements of $E(\mathbb{F}_q)$, we refer to them as *points* on the elliptic curve $E(K)$. As previously hinted, we can use elliptic curves for performing computations. In particular, we can define an addition operation on the points of the elliptic curve, such that $E(\mathbb{F}_q)$ together with this addition operation forms an Abelian group. We recall that an additive Abelian group $(G,+)$ has the properties

1. (Closure) $\forall a,b \in G : a+b \in G$
2. (Associativity) $\forall a,b,c \in G : (a+b)+c = a+(b+c)$
3. (Neutral element) $\exists e \in G : \forall a \in G : a+e = e+a = a$
4. (Inverse element) $\forall a \in G : \exists b \in G : a+b = b+a = e$
5. (Commutativity) $\forall a,b \in G : a+b = b+a$

Consider a finite field $\mathbb{F}_q$ and an elliptic curve $E(\mathbb{F}_q)$ over $\mathbb{F}_q$ defined by a generalized Weierstrass equation (1). Now, consider two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ from $E(\mathbb{F}_q)$ and define how to *add* those points, i.e. determine $P_1 + P_2 \in E(\mathbb{F}_q)$.

When $x_1 \neq x_2$, let

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{and} \qquad \mu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \tag{5}$$

and when $x_1 = x_2$ and $P_2 \neq -P_1$ let

$$\lambda = \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} \qquad \text{and} \qquad \mu = \frac{-x_1^3 + a_4 x_1 + 2a_6 - a_3 y_1}{2y_1 + a_1 x_1 + a_3}. \tag{6}$$

If $P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$, then the coordinates of P_3 are given by

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2 \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3. \end{aligned} \tag{7}$$

Characteristic $p > 3$. When the characteristic of $\mathbb{F}_q$ satisfies $\mathrm{char}(\mathbb{F}_q) > 3$, the curve equation for $E(\mathbb{F}_q)$ has a particularly simple form,

$$Y^2 = X^3 + AX + B, \tag{8}$$

and the discriminant is given by

$$\Delta_E = 4A^3 + 27B^2 \tag{9}$$

which we require to be non-zero. In this case, the formulae for addition of points on $E(\mathbb{F}_q)$ is also particularly simple. The point addition in this case, using the notation from before, proceeds in the following way.

1. If $P_1 = \mathcal{O}$ then $P_1 + P_2 = P_2$
2. If $P_2 = \mathcal{O}$ then $P_1 + P_2 = P_1$
3. If $y_1 = -y_2$ then $P_1 + P_2 = \mathcal{O}$

4. Else

$$P_1 + P_2 = (x_3, y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1)$$

where

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & , P_1 \neq P_2 \\ (3x_1^2 + A)/(2y_1) & , P_1 = P_2 \end{cases}. \tag{10}$$

Example 2. Let us consider still the curve from Example 1. We start by implementing point addition as a Sage function:

```
# add points
# assume p1 and p2 are tuples of the form (x,y)
# the point at infinity is represented by "infinity" in sage
def add_points(a1, a3, a2, a4, a6, p1, p2, K):
   if p1 == infinity:
      return p2
   elif p2 == infinity:
      return p1

   x1 = K(p1[0]); y1 = K(p1[1]); x2 = K(p2[0]); y2 = K(p2[1]);
   if x1 != x2:
      lamb = (y2 - y1)/(x2 - x1)
      mu = (y1*x2 - y2*x1)/(x2 - x1)
   else:
      if y1 == -y2:
         return infinity
      lamb = (3*x1^2 + 2*a2*x1 + a4 - a1*y1)/(2*y1 + a1*x1 +
          a3)
      mu = (-x1^3 + a4*x1 + 2*a6 - a3*y1)/(2*y1 + a1*x1 + a3)
   x3 = lamb^2 + a1*lamb - a2 - x1 - x2
   return (x3, -(lamb + a1)*x3 - mu - a3)
```

We now add the points $(1,5)$ and $(2,10)$:

```
sage: add_points(0,0,0,3,8,(1,5),(2,10),K)
(9, 7)
```

Consider a point $P \in E(\mathbb{F}_q)$. If we keep adding P to itself, we are in effect taking *multiples* of P, and we write for any $n \in \mathbb{Z}_{\mathsf{ord}(P)}$,

$$nP = \underbrace{P + P + \cdots + P}_{n \text{ times}}. \tag{11}$$

If we consider an elliptic curve over the reals, $E(\mathbb{R})$, there is a nice geometric interpretation of point addition that is worth mentioning. Consider points $P,Q \in E(\mathbb{R})$ and the line segment going through them. Unless $x_P = -x_Q$, i.e. the line segment is vertical, it will intersect $E(\mathbb{R})$ in a third point on the curve. Reflecting this point in the first axis yields another point $R \in E(\mathbb{R})$, and it turns out that $R = P + Q$. Note that for the special case where $x_P = x_Q$ we define $P + Q = \mathcal{O}$, while if $P = Q$, the line segment is a tangent in the point P which hits the curve in a second point which we reflect in the first axis to obtain $R = P + Q = 2P$.

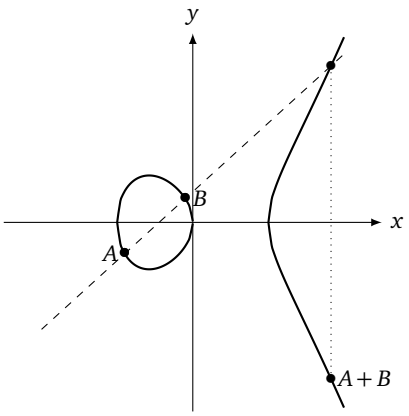

Fig. 1. Addition of points on $E(\mathbb{R})$

2.2 The Computational Aspect of Elliptic Curve Arithmetic

When elliptic curves are employed in cryptography, we care a great deal about their implementation, because to avoid becoming a bottleneck in the system, the performance of the implementation must be up to par. Ultimately, the most important computation, used everywhere in elliptic curve cryptography, is point multiplication, i.e. the computation of nP for $P \in E(\mathbb{F}_q)$ and $n \in \mathbb{Z}_{\mathsf{ord}(P)}$. For that reason, implementation of fast point multiplication has been the focus of optimization. In this section, we describe some of the most important observations that help us towards fast implementations of point multiplication. We focus on the cases where $\mathsf{char}(\mathbb{F}_q) = 2$ or $\mathsf{char}(\mathbb{F}_q) > 3$.

The Cost of Addition and Projective Coordinates. If we start by inspecting the case where $\mathsf{char}(\mathbb{F}_q) > 3$, we see from (10) that when $P_1 \neq P_2$, we need to perform one *inversion*, two *multiplications* and one squaring of elements in $\mathbb{F}_q$. We denote this cost by $1I + 2M + 1S$ (I for inversion, M for multiplication and S for squaring), and we never count addition nor multiplication by small constants such as 2 or 3, as their cost in comparison is negligible. On the other hand, if $P_1 = P_2$, i.e. we are doubling the point, we see from inspecting (10) that the cost is $1I + 2M + 2S$.

It has been estimated [1] that inversion in $\mathbb{F}_q$ is between 9 and 40 times slower than multiplication. For that reason, it is an interesting question whether it is possible to trade off the inversion required in (10) by doing more multiplications. As it turns out, the answer is positive, and what we need to do is to represent the curve equation and the points on the elliptic curve in a different way – in a different *coordinate system*. Several such systems exist, and in fact the one we have already introduced is called the *affine* system. In the following, we describe an alternate system called *projective* coordinates.

If we modify the curve equation to

$$Y^2Z = X^3 + AXZ^2 + BZ^3, \tag{12}$$

then a point $(X, Y, Z) \in \mathbb{F}_q^3$ satisfying the equation is a point on the elliptic curve in projective coordinates. It corresponds to the point $(X/Z, Y/Z)$ in affine coordinates when $Z \neq 0$. Let $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ be points on an elliptic curve in projective coordinates. The

point $P_3 = (x_3, y_3, z_3) = P_1 + P_2$ can be computed in the following way. When $P_1 \neq \pm P_2$, we have

$$\begin{aligned} u = y_2 z_1 - y_1 z_2, \quad v = x_2 z_1 - x_1 z_2, \quad w = u^2 z_1 z_2 - v^3 - 2v^2 x_1 x_2, \\ x_3 = vw, \quad y_3 = y(v^2 x_1 x_2 - w) - v^3 y_1 z_2, \quad z_3 = v^3 z_1 z_2, \end{aligned} \tag{13}$$

while if $P_1 = P_2$ we have

$$\begin{aligned} t = Az_1^2 + 3x_1^2, \quad u = y_1 z_1, \quad v = u x_1 y_1, \quad w = t^2 - 8v \\ x_3 = 2uw, \quad y_3 = t(4v - w) - 8y_1^2 u^2, \quad z_3 = 8u^3, \end{aligned} \tag{14}$$

and finally when $P_1 = -P_2$, we have $P_1 + P_2 = \mathcal{O}$.

By inspection of the formulae for point addition in projective coordinates, we see that when $P_1 \neq P_2$, point addition is accomplished at the cost of $12M + 2S$, while point doubling costs $7M + 5S$.

If we compare the cost for general addition in affine coordinates versus projective coordinates, we see that as long as an inversion in the underlying finite field costs at least as much as 12 multiplications, then we are better off performing the addition using projective coordinates. However, it must be mentioned that while converting from affine to projective coordinates comes for free, conversion in the other direction costs $1I + 4M$. With that said, if we are working in affine coordinates and need to perform point multiplication, it will generally be more efficient to first convert to projective coordinates, perform the point multiplication (which requires many point additions), and convert the result back to affine coordinates in the end. Table 1 gives an overview of the cost of point addition. For a more in-depth study, we refer to e.g. [5, Sections 3.2, 3.3] and [1, Sections 13.2, 13.3].

Table 1. Overview of cost for point addition in terms of inversions (I), multiplications (M) and squarings (S) in the underlying finite field, for affine, projective and mixed coordinates

	Affine	**Projective**
$\mathsf{char}(\mathbb{F}_q) > 3$		
General addition	1I + 2M + 1S	12M + 2S
Doubling	1I + 2M + 2S	7M + 5S
$\mathsf{char}(\mathbb{F}_q) = 2$		
General addition	1I + 2M + 1S	15M + 5S
Doubling	1I + 2M + 1S	5M + 5S

Computing nP: The Double-and-Add Algorithm. For computing the scalar multiplication nP for a point $P \in E(\mathbb{F}_q)$, a nice approach called the *double-and-add* algorithm exists. This can be considered as an analogue to the *square-and-multiply* algorithm in multiplicative groups for computing powers.

The idea is, that we first express the scalar n in its binary form,

$$n = n_{r-1} 2^{r-1} + n_{r-2} 2^{r-2} + \cdots n_1 2 + n_0, \qquad n_i \in \{0, 1\}. \tag{15}$$

Now, let $Q_i = 2^i P$. By the distributivity on $E(\mathbb{F}_q)$, we find that

$$nP = n_{r-1} Q_{r-1} + \cdots + n_0 Q_0. \tag{16}$$

These observations give immediately rise to Algorithm 1. The algorithm has r loops, and in each one we check if $n_i = 1$, in which case we need to add $2^i P$ (the current value of Q) to the resulting point R. Also, in each loop, we compute $2^{i+1}P$ and store it in Q by doubling the previous Q.

Algorithm 1: The double-and-add algorithm

Data: Point $P \in E(\mathbb{F}_q)$, integer n
1 Let $n = n_{r-1}2^{r-1} + \cdots n_1 2 + n_0, \qquad n_i \in \{0,1\}$
2 $R \leftarrow \mathcal{O}, Q \leftarrow P$
3 **for** $i = 0, \ldots, r-1$ **do**
4 **if** $n_i = 1$ **then**
5 $R \leftarrow R + Q$
6 **end**
7 $Q \leftarrow 2Q$
8 **end**
9 **return** R

If we consider a randomly chosen scalar n, we see that the double-and-add algorithm requires $\log n$ point doublings (line 7) and about $\frac{1}{2}\log n$ point additions (line 5), as a random n is expected to have Hamming weight $\frac{1}{2}\log n$.

Example 3. Continuing with the same curve as before, let us try to take different multiples of a point on $E(\mathbb{Z}_{13})$. First, we implement the double-and-add algorithm in Sage:

```
# double-and-add algorithm for point scalar multiplication
def point_scalar(a1, a3, a2, a4, a6, n, P, K):
    # determine reverse binary expansion of n
    nb = [int(x) for x in bin(n)[2:]][::-1]
    R = infinity
    Q = P
    for i in range(len(nb)):
        if nb[i] == 1:
            R = add_points(a1, a3, a2, a4, a6, R, Q, K) # R = R +
                Q
        Q = add_points(a1, a3, a2, a4, a6, Q, Q, K) # Q = 2Q
    return R
```

Let us use the code to compute $n \cdot (1,5)$ for $n = 0, \ldots, 9$:

```
sage: print [point_scalar(0,0,0,3,8,n,(1,5),K) for n in range
    (10)]
[+Infinity, (1, 5), (2, 10), (9, 7), (12, 2), (12, 11), (9, 6)
    , (2, 3), (1, 8), +Infinity]
```

We observe that $9 \cdot (1,5) = \mathcal{O}$, which means the point $(1,5)$ on the curve has order 9.

The double-and-add algorithm can be generalized to representing the scalar n in different ways. One alternative way is the *ternary* form, with which we are able to slightly decrease the expected complexity of the scalar multiplication. In this case, we write the scalar n as

$$n = u_k 2^k + u_{k-1}2^{k-1} + \cdots + u_1 k + u_0, \qquad u_i \in \{-1, 0, 1\}. \tag{17}$$

By allowing the coefficient -1 on a term 2^i in the expression of n, the description of n becomes more compact, and, as point *subtraction* is just as efficient as point addition for points on the elliptic curve, this results in slightly fewer expected additions and better performance in general. The algorithm for the ternary form is a direct generalization of Algorithm 1 which we do not give here. The procedure requires about $\log n$ doublings and $\frac{1}{3}\log n$ additions of points.

2.3 The Group Order: Number of Points on an Elliptic Curve

The number of points on an elliptic curve is denoted $\sharp E(\mathbb{F}_q)$ and also called the *order* of the elliptic curve. To asses the order of an elliptic curve, there is a well-known theorem by Hasse.

Theorem 1 (Hasse's theorem). *Let $\mathbb{F}_q$ be a finite field and let $E(\mathbb{F}_q)$ be an elliptic curve over $\mathbb{F}_q$. Then*

$$\sharp E(\mathbb{F}_q) = q + 1 - t, \qquad |t| \leq 2\sqrt{q}. \tag{18}$$

Example 4. Let us consider the curve from Example 1. We use Sage to count the points:

```
sage: points = points_on_curve(0,0,0,3,8,K)
sage: len(points) + 1
9
```

Thus, the curve has 9 points including $\mathcal{O}$, while Hasse's theorem says the number of points should be between 6 and 22 (rounding down and up, respectively). As another example, consider the curve defined by $E : Y^2 = X^3 + 4X + 6$ over $\mathbb{F}_p$ with prime $p = 1447$. We count the points again:

```
sage: points = points_on_curve(0,0,0,4,6,GF(1447))
sage: len(points) + 1
1495
```

In this case, Hasse's theorem says the number of points should satisfy $1371.92 \leq \sharp E(\mathbb{F}_{1447}) \leq 1524.08$.

As the curve order is very important in applications of elliptic curves, it is crucial that we can determine it efficiently. For this, two handy algorithms exist. The first, due to Schoof, computes $\sharp E(\mathbb{F}_q)$ in time $O(\log^8 q)$. The algorithm was improved by Elkies and Atkin who reduced the complexity to $O(\log^6 q)$ with the SEA algorithm.

3 Applications of Elliptic Curves

Elliptic curves have many applications, not just in cryptography, but in mathematics in general. In this section, we present a range of mathematical problems based on elliptic curves, and we show how these problems can be used to define cryptographic schemes. We also see an application of elliptic curves which is not strictly cryptography related, namely the use of such curves for factoring integers.

Definition 3 (The Discrete Logarithm Problem (DLP)). *Let p be a prime and let $h, g \in \mathbb{Z}_p$. The* discrete logarithm problem *asks: Given $h \equiv g^x \bmod p$, determine x.*

Definition 4 (The Elliptic Curve DLP (ECDLP)). *Let $\mathbb{F}_q$ be a finite field and let $E(\mathbb{F}_q)$ be an elliptic curve over $\mathbb{F}_q$. Let $P \in E(\mathbb{F}_q)$ and let $Q = nP$ for some positive integer n. The* elliptic curve discrete logarithm problem *is the problem of determining n when given P and Q. We denote $n = \log_P(Q)$.*

We note that there may not always be an n s.t. $Q = nP$ for any $P, Q \in E(\mathbb{F}_q)$. If such an n does exist (which it will, by construction, in applications of cryptographic relevance), then $Q = s(nP)$ where $s = \mathsf{ord}(P)$ in $E(\mathbb{F}_q)$, and $s \mid \sharp E(\mathbb{F}_q)$. Thus, we can assume that $n \in \mathbb{Z}_{\mathsf{ord}(P)}$.

3.1 Hardness of the ECDLP

The most important part about the ECDLP, is to understand how hard it is to solve, because if we use the assumed hardness of the problem for cryptography, we need to understand the hardness to say anything about how hard the resulting cryptographic scheme is to break. We attend to this issue in the following.

Special Groups: The MOV Algorithm. While the ECDLP is generally considered to be hard, there are cases where it can be solved somewhat efficiently. In this section we discuss a class called *supersingular curves* and the MOV algorithm, named after Menezes, Okamoto and Vanstone [12]. We start by giving the definition of point order below.

Definition 5 (Point order). *Let $m \geq 1$ be an integer. A point $P \in E$ which satisfies $mP = \mathcal{O}$ is said to have* order *m. The set of points of order m is denoted*

$$E[m] = \{P \in E \mid mP = \mathcal{O}\}. \tag{19}$$

We give the following proposition without proof.

Proposition 1. *Let $m \geq 1$ be an integer and let E be an elliptic curve over $\mathbb{F}_p$, and assume that $p \nmid m$. Then for all $j \geq 1$ there exists a k such that*

$$E(\mathbb{F}_{p^{jk}})[m] \cong \mathbb{Z}_m \times \mathbb{Z}_m. \tag{20}$$

Definition 6 (Embedding degree). *Let E be an elliptic curve over $\mathbb{F}_p$ and let $m \geq 1$ be an integer such that $p \nmid m$. We define the* embedding degree of E with respect to m *to be the smallest value of k such that*

$$E(\mathbb{F}_{p^k})[m] \cong \mathbb{Z}_m \times \mathbb{Z}_m. \tag{21}$$

Consider an elliptic curve $E(\mathbb{F}_p)$. Interestingly, for an embedding degree of k, the Weil pairing embeds the ECDLP on the elliptic curve $E(\mathbb{F}_p)$ into the DLP in the field $\mathbb{F}_{p^k}$. Consider a point $P \in E(\mathbb{F}_p)$ of order ℓ, where ℓ is a large prime, and let $Q \in E(\mathbb{F}_p)$ be a multiple of P. Also, let k be the embedding degree of $E(\mathbb{F}_p)$ with respect to ℓ. Then, if we know an efficient algorithm to solve the DLP in $\mathbb{F}_{p^k}$, then the MOV algorithm solves the corresponding ECDLP in $E(\mathbb{F}_p)$. We give a summary as Algorithm 2.

Algorithm 2: The MOV algorithm for solving ECDLP

Data: Curve $E(\mathbb{F}_p)$ and points P, Q, where P has order ℓ and E has embedding degree k w.r.t ℓ

1 $N \leftarrow \sharp E(\mathbb{F}_{p^k})$ `/* Note that ℓ | N since E has a point of order ℓ */`
2 **repeat**
3 Choose a random point $T \in E(\mathbb{F}_{p^k})$ where $T \notin E(\mathbb{F}_p)$
4 $T' \leftarrow (N/\ell)T$ `/* Determine point of order ℓ */`
5 **until** $T' = \mathcal{O}$
6 $\alpha \leftarrow e_\ell(P, T') \in \mathbb{F}^*_{p^k}$ `/* Compute the Weil pairing values */`
7 **if** $\alpha = 1$ **then** Go to line 2
8 $\beta \leftarrow e_\ell(Q, T') \in \mathbb{F}^*_{p^k}$
9 $n \leftarrow$ solution to DLP for α, β in $\mathbb{F}^*_{p^k}$
10 **return** n

The function e_ℓ used in Algorithm 2 is the *Weil pairing*, a concrete example of a *bilinear pairing* which we study more generally in Section 4. In this introductory note, we will not go further into the details of why the MOV algorithm works: the important thing to note is, that *if* the embedding degree of the curve with respect to the order of the target point P is not too large, and *if* one can solve the DLP in $\mathbb{F}_{p^k}^*$, then one can also solve the ECDLP in $E(\mathbb{F}_p)$.

But how large is *not too large*? If $k > (\ln p)^2$, then solving the ECDLP using the MOV algorithm is infeasible, and this is almost always the case for a randomly chosen elliptic curve [6, Section 5.9]. However, a class of curves with $\sharp E(\mathbb{F}_p) = p + 1$, called *supersingular curves*, have an embedding degree at most $k \leq 6$, so in this case solving the ECDLP is quite easy. Another class is the *anomalous curves* with $\sharp E(\mathbb{F}_p) = p$, where there are linear time algorithms for solving the ECDLP, so these curves must also be avoided for cryptography (see e.g. [14, 15]).

Black-box Groups. To solve the ECDLP for black-box groups, one can use the *baby-step giant-step* approach of Shanks, as presented in Algorithm 3. The idea is based on collisions and is a standard time/memory-tradeoff.

The algorithm proceeds as follows. The attacker chooses random values $j_1, \ldots, j_r$ and $k_1, \ldots, k_r$ from $\mathbb{Z}_{\mathsf{ord}(P)}$ and stores in one list L_1 the points $j_i P$ and in another list L_2 the points $k_i P + Q$ for $i = 1, \ldots, r$. When L_1 and L_2 have a point in common, it means there exists a pair (u, v) s.t. $j_u P$ is in L_2 and $k_v P + Q$ is in L_1, and we find

$$\begin{aligned} j_u P &= k_v P + Q \\ \Leftrightarrow Q &= (j_u - k_v)P, \end{aligned} \tag{22}$$

so $\log_P(Q) = j_u - k_v$.

Due to the birthday paradox, which should be familiar to the reader, we know that if the size of the two sets is slightly larger than $\sqrt{n}$, say $r \approx 3\sqrt{n}$, where n is the order of the curve, then there is a good chance that the algorithm will succeed in solving the ECDLP.

Algorithm 3: Collision algorithm for solving ECDLP

Data: Points $P, Q \in E(\mathbb{F}_q)$ s.t. $Q = nP$ for some n
$L_1, L_2 \leftarrow \emptyset$ **for** $i = 1, \ldots, r$ **do**
  Pick random values $j_i, k_i \in \mathbb{Z}_{\mathsf{ord}(P)}$
  $L_1 \leftarrow L_1 \cup \{j_i P\}$
  $L_2 \leftarrow L_2 \cup \{k_i P + Q\}$
end
if $L_1 \cap L_2 \neq \emptyset$ **then**
  Let u, v be s.t. $j_u P \in L_2$ and $k_v P + Q \in L_1$
  return $j_u - k_v$
end
return Failure

Recall that in multiplicative group $\mathbb{Z}_p$, a method called *index calculus* exists for solving the DLP in time $O(2^c \sqrt{\log p \log\log p})$ for some small constant c. The approach is based on determining a *factor base* consisting primes ℓ up to some bound B. Then, the DLP for $g^x \equiv \ell \bmod p$ is solved for all primes $\ell \leq B$. Next, one finds a value of k s.t. $hg^{-k} \bmod p$ is B-smooth, i.e. can be factored using primes up to B. It now follows that $\log_g(h)$ is equivalent to k plus a sum of scaled logarithms of the primes in the factor base modulo $p - 1$ (see e.g. [6, Section 3.8] for more details).

One might wonder if such a subexponential index calculus algorithm exists for solving the ECDLP, and the answer is negative. The fastest known methods to solve general ECDLP require $O(\sqrt{n})$ steps. It is, however, possible using Pollard's ρ or λ methods to obtain collision algorithms that are storage-free, as opposed to Algorithm 3.

3.2 Elliptic Diffie-Hellman Key Exchange

The reader should be familiar with the Diffie-Hellman key exchange, a protocol designed for two parties communicating over an insecure line, to exchange a shared secret in a secure manner. After the key exchange has been performed, the shared secret can be used by the communicating parties as a private key in a more efficient symmetric-key primitive such as AES-GCM. The Diffie-Hellman key exchange protocol is based on the hardness of the DLP in a multiplicative group. In this section, we describe a natural analogue which is based on the ECDLP.

First, Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E(\mathbb{F}_q)$ in which the ECDLP is hard, and a point $P \in E(\mathbb{F}_q)$. Both parties pick a secret integer from $\mathbb{Z}_{\mathsf{ord}(P)}$ and multiply this with the publicly known point P. The resulting point is transferred to the other party who multiplies their secret integer with the received point. Due to the commutativity of scalar multiplication, both parties end up computing the same shared secret. The protocol is summarized in Figure 2.

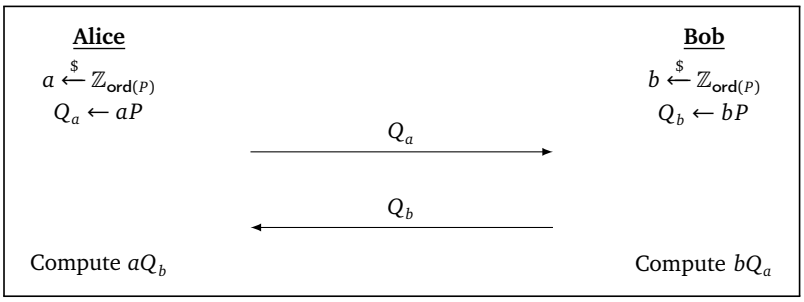

Fig. 2. The elliptic Diffie-Hellman protocol

Example 5. We show an example in Sage using the curve $E : Y^2 = X^3 + 3X + 8$ over the finite field $\mathbb{F}_{503}$. We show that the shared secret computed by Alice and Bob, is the same point on the curve.

```
sage: K = GF(503)
sage: points = points_on_curve(0,0,0,3,8,K) # get all points
sage: P = points[33] # choose some point to be P
sage: a = 33
sage: b = 8
sage: Qa = point_scalar(0,0,0,3,8,a,P,K)
sage: Qb = point_scalar(0,0,0,3,8,b,P,K)
sage: point_scalar(0,0,0,3,8,a,Qb,K)
(217, 87)
sage: point_scalar(0,0,0,3,8,b,Qa,K)
(217, 87)
```

As such, we see they determined the same, secret point $(217, 87)$.

In practice, when Alice and Bob use the elliptic Diffie-Hellman protocol, only the x coordinates of the points Q_a respectively Q_b are transmitted. This is done for efficiency reasons, and because each x coordinate has only two valid y coordinates which can be efficiently computed. Using this approach, Alice and Bob end up computing the same $\pm abP$, i.e. the points are identical up to the sign. After computing their points, the y coordinate is dropped to obtain the same x coordinate.

The security of the elliptic Diffie-Hellman problem, while tightly linked to the ECDLP, is defined as an analogue to the well-known *Diffie-Hellman problem* (DHP). We give this analogue as Definition 7.

Definition 7 (Elliptic curve Diffie-Hellman problem (ECDHP)). *Let $\mathbb{F}_q$ be a finite field and let $E(\mathbb{F}_q)$ be an elliptic curve over $\mathbb{F}_q$. Let $P \in E(\mathbb{F}_q)$. The* elliptic curve Diffie-Hellman problem *states: Given aP, bP and P for some $a, b \in \mathbb{Z}_{ord(P)}$, determine abP.*

It is easy to see that the ECDHP is polynomial-time reducible to ECDLP. In other words, if one has access to an oracle solving ECDLP in $E(\mathbb{F}_q)$, then one can solve ECDHP in $E(\mathbb{F}_q)$ with polynomially many calls to this oracle. In particular, one could use the oracle with aP and P to determine a, and then afterwards use a and bP to compute abP. It is an open problem whether the converse is true, i.e. that ECDLP is polynomial-time reducible to ECDHP, but it is generally believed to be so, and has also been proven in some cases.

3.3 Elliptic Curve ElGamal

The reader should be familiar with the ElGamal public-key cryptosystem, when defined over a cyclic multiplicative group. In this section, we show a variant of the ElGamal cryptosystem, which uses the additive structure of an elliptic curve, to obtain a public-key cryptosystem providing the same functionality.

Alice first randomly picks a secret integer s. She then computes $B = sP$ which becomes her public key. Now, Bob who wants to send a secret message (a point M on the curve) to Alice does so by first picking his own secret integer k. The first part of the ciphertext is a point on the curve $C_1 = kP$. The second part of the ciphertext is another point C_2 which is the sum of the message M and the point kB. The correctness of the scheme is given by

$$\begin{aligned} C_2 - sC_1 &= (M + kB) - s(kP) \\ &= M + ksP - skP \\ &= M. \end{aligned} \tag{23}$$

Example 6. Using the same curve as in Example 5, we give an example of encryption and decryption using the elliptic curve ElGamal cryptosystem.

```
sage: K = GF(503)
sage: points = points_on_curve(0,0,0,3,8,K)
sage: P = points[33]
sage: s = 73
sage: B = point_scalar(0,0,0,3,8,s,P,K)
sage: M = points[93]
sage: k = 34
sage: C1 = point_scalar(0,0,0,3,8,k,P,K)
sage: C2 = add_points(0,0,0,3,8,M,point_scalar(0,0,0,3,8,k,B,K
    ),K)
sage: sC1 = point_scalar(0,0,0,3,8,s,(C1[0],-C1[1]),K)
sage: Mp = add_points(0,0,0,3,8,C2,sC1,K)
sage: M, Mp
((85, 488), (85, 488))
```

As expected, the plaintext recovered by Alice, denoted $\mathtt{Mp}$, is equal to the plaintext M chosen by Bob. Note, that to obtain the point $-sC_1$, one computes $s \cdot (x, -y)$ where $C_1 = (x, y)$, as $-C_1 = (x, -y)$.

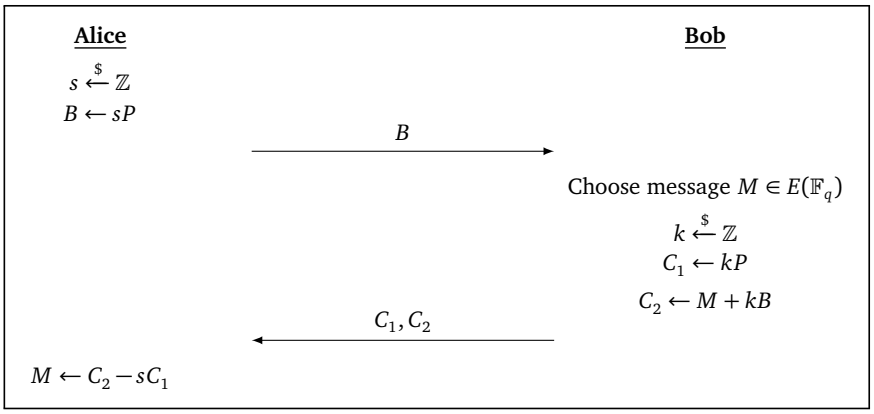

Fig. 3. The elliptic curve ElGamal cryptosystem

Unfortunately, elliptic curve ElGamal is made less attractive by the fact that in practice, one needs a way to map meaningful messages (e.g. in binary) to points on the curve. Also, by Hasse's theorem, we know that there are approximately q points on the curve, i.e. q possible messages. On the other hand, the ciphertext consists of two points of two coordinates each, meaning there is a 4-to-1 expansion from message length to ciphertext length.

To try to overcome the problem of the message expansion, it is tempting to let Bob only send the x coordinates of the points (C_1, C_2). However, to correctly decrypt, Alice needs the correct value of both y coordinates, as the points $M = C_2 - sC_1$ and $C_2 + sC_1$ are generally quite different. However, this can be solved by an encoding where Bob sends one additional bit for each of C_1 and C_2 to indicate which of the two possible y coordinates (for each point) is the correct one. This method is known as *point compression*. The two observations mentioned mean that elliptic curve ElGamal is not widely used in practice.

3.4 Lenstra's Elliptic Curve Factorization

In the middle of the 1980s, Hendrik Lenstra found a new application for elliptic curves in number theory, that had not previously been considered: factoring integers [10]. We consider a target integer N and would like to find a non-trivial divisor of it. If, for example, N is a modulus in the RSA encryption scheme (see e.g. [11, Section 8.2]), then solving this problem is computationally very hard. In fact, the security of RSA is based on exactly the hardness of this problem.

In this section, we describe the approach developed by Lenstra, which can successfully factor integers N of up to around 60 decimal digits, or when N has a prime factor of about 20 to 30 digits [16, Section 7.1].

While elliptic curves are defined over (finite) fields and not rings, the idea of Lenstra is to pretend that $\mathbb{Z}_N$ is a field, and construct an elliptic curve $E(\mathbb{Z}_N)$. Let us consider a point $P \in E(\mathbb{Z}_N)$. We start computing multiples of this point, $2P, 3P, \ldots$, and so on. We know from the formulae for the point addition (see Equation (10)), that when adding points, we need to compute an inverse

in the underlying field to determine λ. However, an element $a \in \mathbb{Z}_N$ only has a multiplicative inverse if $\gcd(a, N) = 1$. Thus, when computing multiples of P, it may happen that we encounter a situation where we need to determine $(x_2 - x_1)^{-1} \in \mathbb{Z}_N$, but no such element exists, because $1 < d = \gcd(x_2 - x_1, N) < N$. Let's say we find a j s.t. when we need to compute jP, we find that $1 < d < N$, and no inverse exists to determine the slope λ. What happened is, that for a factor d of N, we have $jP \equiv \mathcal{O} \bmod d$, but modulo the other factors of N, the point jP would be a finite point. In this case, we find that d is a non-trivial factor of N, and we have successfully accomplished our goal.

It may also happen that $jP \equiv \mathcal{O}$ modulo the other factors of N. In this case, the greatest common divisor d would equal N, and we would not find a factor. However, this is very unlikely to happen [16, Section 7.1]. However, such j satisfying the conditions mentioned do not occur too often. However, what we can do is switch to a different curve, still over $\mathbb{Z}_N$ and try over. If we try enough curves, it is likely that for one of them we will find such a j.

Algorithm 4: Lenstra's elliptic curve factorization

Data: Integer N to be factored
Result: Nontrivial factor $d \mid N$

$P \xleftarrow{\$} (a, b) \in \mathbb{Z}_N \times Z_N$
$A \xleftarrow{\$} \mathbb{Z}_N$
$B \leftarrow b^2 - a^3 - A \cdot a \bmod N$
Let $E(\mathbb{Z}_N)$ be the curve $Y^2 = X^3 + AX + B$
for $j = 2, 3, \ldots$ **do**
    $Q \leftarrow jP$
    $P \leftarrow Q$
    if *the computation of P fails* **then**
        if $d < N$ **then return** d;
        if $d = N$ **then** Go to 1;
    end
end

Example 7. We give an example where we use Lenstra's elliptic curve factorization algorithm to factor the number $N = 6887$. The example is taken from [6, Section 5.6].

```
sage: N = 6887
sage: P = (1512,3166)
sage: A = 14
sage: B = P[1]^2 - P[0]^3 - A * P[0]; B % N
19
```

Thus, we are working with the curve $E : Y^2 = X^3 + 14X + 19$. We start by computing multiples of P. We use Sage to check the values of $n!P$ for $n = 1, \ldots, 6$:

```
sage: print [point_scalar(0,0,0,14,19,factorial(n),P,K) for n
    in range(7)]
[(1512, 3166), (1512, 3166), (3466, 2996), (3067, 396), (6507,
     2654), (2783, 6278), (6141, 5581)]
```

If we try to compute $7!P$, we find that we get a division by zero modulo N at some point:

```
sage: point_scalar(0,0,0,14,19,factorial(7),P,K)
[...]
ZeroDivisionError: Inverse does not exist.
```

Aha! To find out where we ran into trouble, we proceed by setting $Q = 6!P$ and compute $2Q$ and $4Q$ to finally get $7!P = Q + 2Q + 4Q$:

```
sage: Q = point_scalar(0,0,0,14,19,factorial(6),P,K)
sage: Q2 = point_scalar(0,0,0,14,19,2,Q,K)
sage: Q4 = point_scalar(0,0,0,14,19,2,Q2,K)
sage: Q2, Q4
((5380, 174), (203, 2038))
sage: add_points(0,0,0,14,19,Q,Q2,K)
(984, 589)
sage: gcd(984-203,N)
71
```

What we saw is, that when we needed to add the points $(Q + 2Q)$ and $4Q$, we had to determine the inverse of $984 - 203 \bmod N$, however, this does not exist as $\gcd(984 - 203, N) = 71$, but instead we found that 71 is a factor of N.

We summarize the approach as Algorithm 4. Note, that the way the algorithm proceeds is by first picking a random point P and then constructs the curve such that the point lies on the curve. The reason we do this is, that if we first pick the curve and *then* pick a point, then checking if the point actually lies on the curve is hard without being able to factor N [6, Section 5.6].

4 Cryptography using Bilinear Pairings

Definition 8 (Bilinear pairing). *Let p be a prime and let $G_1 = \langle P \rangle$ be an additive group (typically an elliptic curve) of order p with the neutral element denoted $\mathcal{O}$ and let G_T be a multiplicative group of order p with neutral element denoted* 1.

A bilinear pairing *on (G_1, G_T) is a map*

$$\hat{e} : G_1 \times G_1 \to G_T \tag{24}$$

which satisfies

1. *bilinearity:*

$$\begin{aligned} \forall R, S, T \in G_1 : \hat{e}(R, S + T) = \hat{e}(R, S)\hat{e}(R, T) \qquad \textit{and} \\ \hat{e}(R + S, T) = \hat{e}(R, T)\hat{e}(S, T), \end{aligned} \tag{25}$$

2. *non-degeneracy: $\hat{e}(P, P) \neq 1$, and*
3. *computability: $\hat{e}$ can be efficiently computed.*

Examples of well-known pairings are the *Weil pairing*, named after André Weil, and the *Tate-Lichtenbaum pairing*, named after John Tate and Stephen Lichtenbaum. The description of the Weil and Tate-Lichtenbaum pairings are mathematically involved, and we do not go further into them here. For our further studies, we will need the properties of bilinear pairings alone, and assume their existence. We refer the reader interested in the specific instantiations to e.g. [3, 6, 16].

One of the interesting things about bilinear pairings is how they are related to the DLP. Consider the DLP in G_1. When G_1 is an elliptic curve, this naturally is the ECDLP, and we consider

this case henceforth. The ECDLP in G_1 would state: Given P and $Q = nP$ with $P, Q \in G_1$, determine n.

An interesting observation is, given a bilinear pairing on (G_1, G_T) there is a polynomial-time reduction from the ECDLP in G_1 to the DLP in G_T. Since $Q = nP$, it follows that

$$\begin{aligned} \hat{e}(P,Q) &= \hat{e}(P,nP) \\ &= \hat{e}(P,P)^n. \end{aligned} \tag{26}$$

Thus, $\log_P(Q) = \log_g(h)$ where $h = \hat{e}(P,Q)$ and $g = \hat{e}(P,P)$. Thus, if one can efficiently solve the DLP in G_T, then one can also efficiently solve the ECDLP in G_1.

From the introduction of bilinear pairings, another range of problems related to the DHP, which we can use for cryptographic applications arise. We present them, and some observations in the following.

Definition 9 (The bilinear DHP (BDHP)). *Let $\hat{e}$ be a bilinear pairing on (G_1, G_T) and let P be a generator of G_1. Let $a, b, c \in \mathbb{Z}_{ord(G_1)}$. The* bilinear Diffie-Hellman problem *asks: Given P, aP, bP and $cP \in G_1$, determine $\hat{e}(P,P)^{abc} \in G_T$.*

There is a result which relates the hardness of the BDHP to the DHP in G_1 and G_T.

Theorem 2. *Let $\hat{e}$ be a bilinear pairing on (G_1, G_T). Then the hardness of the BDHP on (G_1, G_T) implies the hardness of the DHP in* both *G_1 and G_T.*

Proof. To prove the statement, we show that if one can solve the DHP in either of G_1 or G_T, then one can solve the BDHP on (G_1, G_T). Assume first we can solve the DHP in G_1. Then, by assumption, we can compute abP in G_1 and we find that $\hat{e}(abP, cP) = \hat{e}(P,P)^{abc}$. If the DHP can be efficiently solved in G_T, then one could compute $g = \hat{e}(P,P)$, $g^{ab} = \hat{e}(aP, bP)$, $g^c = \hat{e}(P, cP)$ and finally $g^{abc} = \hat{e}(P,P)^{abc}$. □

In general, we believe that the BDHP is just as hard as DHP in G_1 or G_T. In the following section, we give an application of bilinear pairings in cryptography, whose security is based on the hardness of solving the BDHP.

4.1 Tripartite Diffie-Hellman

In Figure 4, we illustrate a key exchange protocol with three parties, Alice, Bob and Charlie. The protocol is due to Joux [8]. It uses a bilinear pairing $\hat{e}$ on (G_1, G_T) for which the BDHP is hard.

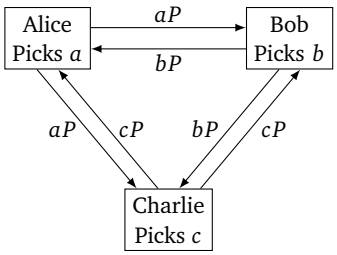

Fig. 4. Tripartite Diffie-Hellman key exchange

The protocol proceeds as follows. The three parties each pick a secret parameter $a, b, c \in \mathbb{Z}_p^*$. Each party multiplies the public point P by their secret value and broadcasts the resulting point

to the other two parties. Now, Alice computes the shared secret as $K = \hat{e}(bP, cP)^a = \hat{e}(P,P)^{abc}$ (and symmetrically for Bob and Charlie).

The important observation for Joux's tripartite Diffie-Hellman key exchange is, that a *passive* adversary faces the BDHP. In other words, an attacker who, by assumption, can *observe* the communication between Alice, Bob and Charlie (but not modify it!), needs to solve the BDHP if she is to obtain the shared secret K. Unfortunately, if the system is to be made secure against *active* adversaries, i.e. adversaries who by assumption can modify the communication between the parties, without them noticing, then another round of transfer of data is required. In that case, each party is better off with a regular Diffie-Hellman key exchange with two independent parties. An obvious question is whether this approach can be generalized to ℓ parties. Unfortunately, this would require a pairing mapping $G_1^{\ell-1}$ to G_T, and the existence of such pairings is an open problem!

4.2 Short Signatures

Another application of bilinear pairings which we highlight is that of short signatures. The motivation is the following. In most DLP-based signature schemes, such as ElGamal and the DSA, the signature is a *pair* of integers modulo some large prime p for which the DLP in $\mathbb{Z}_p^*$ is computationally hard. In the BLS scheme, named after Boneh, Lynn and Shacham [2], which we describe in the following, the signature is a single element of the group G_1 (again, typically this is a point on an elliptic curve), and the size is about $\log p$ bits (comparing to about $2\log p$ bits for aforementioned schemes).

The setup is the following. The scheme is composed of two parties: The *signee* and the *verifier*. The signee has a portion of data that she wants to sign off on, such that anyone else (in particular the verifier) can verify that she has done so. For example, a software company might want to sign a piece of software being released for download on the Internet, such that anyone having downloaded the software can verify that this software is in fact not malware placed by a malicious attacker.

To use BLS, two components are required: A bilinear pairing $\hat{e}$ on (G_1, G_T) for which the DHP is intractable in G_1 and a hash function $H : \{0,1\}^* \to G_1 \backslash \{\mathcal{O}\}$. The scheme proceeds as follows. The signee chooses a private key $a \in \mathbb{Z}_n^*$ where $n = \mathsf{ord}(G_1)$. Her public key is the elliptic curve point $A = aP \in G_1$. Now, when she wants to sign a particular portion of data m (assumed to be a binary string of non-negative length), she first computes the hash $M = H(m)$. The hash, M, is now a point on the curve G_1, and she can use the private key a to compute the signature as $S = aM \in G_1$.

Now, the verifier obtains a tuple (P, A, M, S), and wants to check if the signature S on the message M is valid. In other words, she wants to check if $S = aM$. Note that this is an instance of the DDHP in G_1. Fortunately, given the existence of the pairing $\hat{e}$, solving this is easy: The verifier accepts the signature *if and only if* she finds that $\hat{e}(P,S) = \hat{e}(M,S)$. Consider the problem an adversary faces when he wants to forge a signature on some M'. He must compute $S' = aM'$ given P,A and $M' = H(m')$ for some possibly malicious data m'. This problem is exactly the DHP in G_1, which presumable is computationally infeasible.

5 Lattice-Based Cryptography

In this section we introduce the mathematical concept of *lattices*, a tool which as we shall see, much like elliptic curves, can be used to define problems that are computationally intractable. As always, we use the mathematical structure to define cryptosystems whose security rely on the

hardness of those problems. First, however, we start off with a brief review of vector spaces, as these are tightly connected with lattices.

5.1 Review of Vector Spaces

The following definitions should be known to the reader already, but we give them here as well, as they may be useful to have handy during the rest of the treatment on lattices.

Definition 10 (Vector space). *A* vector space *V is a subset of $\mathbb{R}^m$ for which it holds that*

$$\forall \alpha_1, \alpha_2 \in \mathbb{R}, v_1, v_2 \in V : \alpha_1 v_1 + \alpha_2 v_2 \in V. \tag{27}$$

Definition 11 (Linear combination). *Let $v_1, \ldots, v_k \in V$. A* linear combination *of $v_1, \ldots, v_k$ is any vector of the form*

$$w = \alpha_1 v_1 + \ldots + \alpha_k v_k, \tag{28}$$

where $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$. The span *of $\{v_1, \ldots, v_k\}$ is*

$$\mathsf{span}\{v_1, \ldots, v_k\} = \{\alpha_1 v_1 + \cdots + \alpha_k v_k \mid \alpha_1, \ldots, \alpha_k \in \mathbb{R}\}. \tag{29}$$

Definition 12 (Independence). *A set of vectors $v_1, \ldots, v_k \in V$ are said to be* linearly independent *if the only solution to*

$$\alpha_1 v_1 + \cdots + \alpha_k v_k = 0 \tag{30}$$

is the trivial solution $\alpha_i = 0$ for $i = 1, \ldots, k$. A set of vectors that are not linearly independent are said to be linearly dependent.

Definition 13 (Basis). *A* basis *for a vector space V is a set of linearly independent vectors that span V.*

Definition 14 (Orthogonal basis). *A basis $V = \{v_1, \ldots, v_k\}$ is said to be* orthogonal *if and only if*

$$\forall i, j \in \{1, \ldots, k\}, i \neq j : v_i \cdot v_j = 0. \tag{31}$$

A basis is said to be orthonormal*, if in addition, $||v_i|| = 1$ for all $i = 1, \ldots, n$.*

Definition 15 (Euclidean norm). *The* Euclidean norm *or* length *or a vector $v = (v_1, \ldots, v_n)$ is determined as*

$$||v|| = \sqrt{v_1^2 + \cdots + v_n^2}. \tag{32}$$

Theorem 3 (Gram-Schmidt algorithm). *Let $v_1, \ldots, v_n$ be a basis for a vector space $V \subset \mathbb{R}^m$. Algorithm 5 (the* Gram-Schmidt algorithm*) determines an orthogonal basis $v_1^*, \ldots, v_n^*$ for V, i.e.*

$$\begin{aligned} &\mathsf{span}\{v_1, \ldots, v_n\} = \mathsf{span}\{v_1^*, \ldots, v_n^*\} \quad \text{and} \\ &\forall i \neq j : v_i^* \cdot v_j^* = 0. \end{aligned} \tag{33}$$

Algorithm 5: Gram-Schmidt algorithm

$v_1^* \leftarrow v_1$
for $i = 2, \ldots, n$ **do**
  for $j = 1, \ldots, i-1$ **do** $\mu_{i,j} \leftarrow (v_i \cdot v_j^*)/||v_j^*||^2$;
  $v_i^* \leftarrow v_i - \sum_{j=1}^{i-1} \mu_{i,j} v_j^*$
end

5.2 Lattices

We dive right into the topic of lattices by giving their definition in the following.

Definition 16 (Lattice). *Let $v_1, \ldots, v_n \in \mathbb{R}^n$ be linearly independent vectors. Then*

$$L = \{a_1 v_1 + a_2 v_2 + \cdots + a_n v_n \mid a_1, \ldots, a_n \in \mathbb{Z}\} \tag{34}$$

is called the lattice *spanned by $v_1, \ldots, v_n$.*

As such, an n-dimensional lattice is a subset of $\mathbb{R}^n$. Let L be an n-dimensional lattice and let $v_1, \ldots, v_n$ be a basis for L. Also, let $w_1, \ldots, w_n \in L$. Then for all $i = 1, \ldots, n$, we have $w_i = a_{i1} v_1 + \cdots + a_{in} v_n$. Let

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}. \tag{35}$$

The v_i can be expressed in terms of the w_i vectors if A^{-1} is an integer matrix, and for this to happen, it is required that $\det(A) = \pm 1$. Thus, we state the following.

Proposition 2. *Two bases for L are related by an $n \times n$ matrix over $\mathbb{Z}$ with determinant ± 1.*

Example 8. Let $L \subset \mathbb{R}^3$ be the lattice spanned by the basis $V = \{(2,1,3), (1,2,0), (2,-3,-5)\}$. Consider the matrix

$$U = \begin{pmatrix} 1 & 0 & 1 \\ 1 & -1 & 2 \\ 1 & 2 & 0 \end{pmatrix}. \tag{36}$$

We check with Sage that $\det(U) = \pm$, and we are able to form a new basis for L:

```
sage: V = Matrix([[2,1,3],[1,2,0],[2,-3,-5]])
sage: U = Matrix([[1,0,1],[1,-1,2],[1,2,0]])
sage: U.determinant()
-1
sage: U*V
[ 4 -2 -2]
[ 5 -7 -7]
[ 4  5  3]
```

Thus, $W = \{(4,-2,-2), (5,-7,-7), (4,5,3)\}$ is also a basis for L, and it is related to V by the matrix U.

Definition 17 (Fundamental domain). *Let $v_1, \ldots, v_n \in \mathbb{R}^n$ be a basis for a lattice L. The* fundamental domain *with respect to this basis is*

$$\mathscr{F}(v_1, \ldots, v_n) = \{t_1 v_1 + \cdots + t_n v_n \mid 0 \leq t_i \leq 1\}. \tag{37}$$

When the basis is understood, we denote the fundamental domain simply by $\mathscr{F}$.

Definition 18 (Determinant). *Let L be an n-dimensional lattice and let $\mathscr{F}$ be the fundamental domain for L. The n-dimensional volume of $\mathscr{F}$ is called the* determinant *of L, denoted* $\det(L)$.

A result relating the determinant of a lattice L to the lengths of the basis vectors is known as Hadamard's inequality, which we state in the following.

Theorem 4 (Hadamard's inequality). *Let $v_1, \ldots, v_n$ be any basis for a lattice L and let $\mathscr{F}$ be a fundamental domain for L. Then*

$$\det(L) = \mathrm{Vol}(\mathscr{F}) \leq ||v_1|| \cdots ||v_n||. \tag{38}$$

The closer the basis is to being orthogonal, the tighter the bound from Hadamard's inequality is, and for an orthogonal basis we have equality. The following proposition relates the determinant of a lattice to the determinant of its basis matrix.

Proposition 3. *Let $L \subset \mathbb{R}^n$ be a lattice of dimension n and let F be the basis matrix for L. Then* $\mathrm{Vol}(\mathscr{F}) = |\det(F)|$ *and all fundamental domains $\mathscr{F}$ have the same volume.*

Example 9. Consider the lattice L and basis V from Example 8. We implement the Euclidean norm in Sage:

```
# Euclidean norm. Assumes v is a vector of the form (v1,...,vn
    )
def norm(v):
   return sqrt(sum([x^2 for x in v]))
```

We compare the absolute value of the determinant of V with the product with the norms of the basis vectors to check Hadamard's inequality:

```
sage: abs(V.determinant())
36
sage: RR(prod(norm(x) for x in V))
51.5751878329105
```

We see that $|\det(L)| = 36$ while $\prod_{i=1}^{n} ||v_i|| \approx 52$, indicating that the basis is quite non-orthogonal.

Theorem 5 (Hermite's theorem). *Every lattice L of dimension n contains a vector $v \neq 0$ such that*

$$||v|| \leq \sqrt{n} \cdot \det(L)^{1/n}. \tag{39}$$

Hermite's constant γ_n is the smallest *value such that* every *n-dimensional lattice L contains some $v \neq 0$ where $||v||^2 \leq \gamma_n \cdot \det(L)^{2/n}$.*

Example 10. Continuing with the lattice from Example 8, we find that the lattice L contains a vector $v \neq 0$ such that $||v|| \leq 5.72$, as follows from the Sage code:

```
sage: RR(sqrt(3) * abs(V.determinant())^(1/3))
5.71910575798162
```

For large values of n, which is the case for cryptographic applications, we have

$$\frac{n}{2\pi e} \leq \gamma_n \leq \frac{n}{\pi e}. \tag{40}$$

Another result which we shall need, that also related to the determinant of a lattice, is the following theorem due to Minkowski.

Theorem 6 (Minkowski's theorem). *Let $L \subset \mathbb{R}^n$ be a lattice of dimension n and let $S \subset \mathbb{R}^n$ be a symmetric, bounded, convex set whose volume satisfies* $\mathrm{Vol}(S) > 2^n \det(L)$*. Then S contains a non-zero lattice vector $v \in L$. If also S is closed, then* $\mathrm{Vol}(S) \geq 2^n \det(L)$ *suffices.*

5.3 Hard Problems Using Lattices

In the following, we describe five problems based on lattices that are presumably computationally intractable (given a sufficiently high dimension). We also present some cases in which these problems turn out not to be hard. As we already have all the tools available, let us go ahead and define the problems.

Definition 19 (Shortest vector problem (SVP)). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice. The* shortest vector problem *(SVP) asks to find a non-zero vector $v \in L$ such that*

$$\neg\exists v' \in L : ||v'|| < ||v||. \tag{41}$$

Definition 20 (Closest vector problem (CVP)). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice. The* closest vector problem *(CVP) asks, given $w \in \mathbb{R}^n$ with $w \notin L$, to find a vector $v \in L$ such that*

$$\neg\exists v' \in L : ||v' - w|| < ||v - w||. \tag{42}$$

With the introduction of SVP and CVP, we want to note that CVP is considered to be *slightly harder* than SVP. This is because there often is a reduction from CVP to SVP in a slightly higher dimension. However, in full generality, both problems are considered to be extremely hard. The next two problems we introduce are generalizations of the SVP and CVP, in the sense that they allow a certain *slack* on the optimality of the solution found.

Definition 21 (Approximate shortest vector problem (apprSVP)). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice and let $\psi(n)$ be a function. The* approximate shortest vector problem *(apprSVP) asks to find a non-zero vector $v \in L$ such that*

$$||v|| \leq \psi(n) \cdot ||v_{shortest}||, \tag{43}$$

where $v_{shortest} \in L$ is a solution to the SVP.

Definition 22 (Approximate closest vector problem (apprCVP)). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice and let $\psi(n)$ be a function. The* approximate closest vector problem (apprCVP) *asks, given $w \in \mathbb{R}^n$, to find a vector $v \in L$ such that*

$$||v - w|| \leq \psi(n) \cdot ||v_{closest} - w||, \tag{44}$$

where $v_{closest} \in L$ is a solution to the CVP for w.

Finally, we introduce the last problem which asks to find a shortest basis for a lattice.

Definition 23 (Shortest basis problem (SBP)). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice. The* shortest basis problem (SBP) *asks to find a basis $v_1, \ldots, v_n$ for L which is shortest in some sense, for example to minimize*

$$\max_{1 \leq i \leq n} ||v_i|| \qquad \textit{or} \qquad \sum_{i=1}^{n} ||v_i||^2. \tag{45}$$

5.4 The Gaussian Heuristic

By combining Minkowski's theorem with Hermite's theorem and applying some advanced calculus, it is possible to obtain rather good estimates on the length of a shortest vector and distance to a closest vector. We give the results here, but for the derivations we refer to e.g. [6, Section 6.5.3].

Definition 24 (Gaussian expected shortest length). *Let $L \subset \mathbb{R}^n$ be an n-dimensional lattice. The* Gaussian expected shortest length *is defined as*

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}} \cdot \det(L)^{1/n}. \tag{46}$$

Example 11. The Guassian expected shortest length of the lattice L from Example 8 is

```
sage: RR(sqrt(3/(2*pi*e)) * 36^(1/3))
1.38385616386042
```

Now, let L be a randomly chosen n-dimensional lattice and consider some fixed $w \in \mathbb{R}^n$ where $w \notin L$. Then it holds that $||v|| \approx \sigma(L)$ and also $||v_{shortest} - w|| \approx \sigma(L)$. The measure of the Gaussian expected shortest length is a very useful tool, when we want to quantify the difficulty of the SVP or CVP for a particular lattice L.

5.5 Babai's Algorithm

It turns out that some of the presented problems are very easy to solve if one has at hand an orthogonal basis for the lattice. As such, let us consider a lattice L and a basis $v_1, \ldots, v_n$ for L which is orthogonal. Thus, it holds that $v_i \cdot v_j = 0$ for all $i, j \in \{1, \ldots, n\}$ where $i \neq j$. The length of any vector in L can be described as

$$||a_1 v_1 + a_2 v_2 + \cdots + a_n v_n||^2. \tag{47}$$

Due to the assumption of orthogonality, this can be written as

$$a_1^2||v_1||^2 + a_2^2||v_2||^2 + \cdots + a_n^2||v_n||^2. \tag{48}$$

As each of the a_i are integers, it follows immediately that the shortest vector in L is in the set $\{\pm v_1, \pm v_2, \ldots, \pm v_n\}$, because to minimize the length we set one of the a_i to either positive or negative one and the others to zero. Thus, it was very easy to solve the SVP in L when we have an orthogonal basis.

Let us consider the CVP for a vector $w \in \mathbb{R}^n$, where $w \notin L$. As $v_1, \ldots, v_n$ is a basis for L, we can write

$$w = t_1 v_1 + t_2 v_2 + \cdots + t_n v_n, \tag{49}$$

where each $t_i \in \mathbb{R}$. Now, letting $v = a_1 v_1 + \cdots + a_n v_n$ with $a_i \in \mathbb{Z}$, we have expressed some vector $v \in L$ and by the assumption of orthogonality of the basis we have

$$||v - w||^2 = (a_1 - t_1)^2||v_1||^2 + (a_2 - t_2)^2||v_2||^2 + \cdots + (a_n - t_n)^2||v_n||^2. \tag{50}$$

To minimize the distance between v and w we should minimize each term $(a_i - t_i)^2$, which we do by setting $a_i = \lfloor t_i \rceil$ for all $i = 1, \ldots, n$. Again, we saw that with an orthogonal basis, it was very easy to solve the CVP for the lattice L.

An idea, known as Babai's algorithm, which we describe in the following, is based on these observations. The idea is to use some lattice basis which is "reasonably orthogonal" to try to solve

the CVP. Let us consider a lattice L with a basis $v_1, \ldots, v_n$ and a target vector $w \in \mathbb{R}^n$. Let $\mathscr{F}$ be the fundamental domain for L expressed using $v_1, \ldots, v_n$. Now, w can be expressed as

$$w = \mathscr{F} + v, \tag{51}$$

i.e. as a translation of the fundamental domain by some vector $v \in L$. Thus,

$$w = v + \epsilon_1 v_1 + \epsilon_2 v_2 + \cdots + \epsilon_n v_n, \qquad 0 \leq \epsilon_i < 1. \tag{52}$$

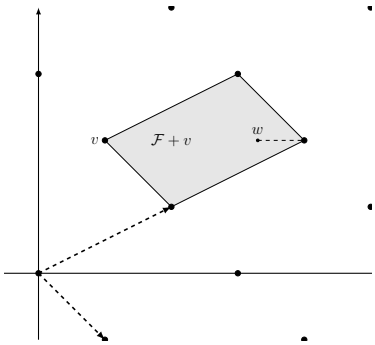

Fig. 5. Babai's algorithm with a sufficiently orthogonal basis in two dimensions. Due to the high orthogonality of the basis vectors (dashed), we see that the vertex in the translated fundamental domain is also the closest lattice point.

If we consider an example where L is a 2-dimensional lattice, Figure 5 shows that, when the basis vectors are reasonably orthogonal, the translated fundamental domain $\mathscr{F} + v$ contains the vector w in a way such that the vertex of $\mathscr{F} + v$ which is closest to w is also likely to be the closest vector in L. As such, we may guess that the closest vector is given by

$$v_{closest} = v + \lfloor \epsilon_1 \rceil v_1 + \lfloor \epsilon_2 \rceil v_2 + \cdots + \lfloor \epsilon_n \rceil v_n. \tag{53}$$

If, on the other hand, the basis for L is highly non-orthogonal, Babai's algorithm is unlikely to work. We illustrate this again for a 2-dimensional lattice in Figure 6, where we observe that the points in the translated fundamental domain $\mathscr{F} + v$ are likely to be far away from the target vector w. We remark that for higher dimensions, this inaccuracy of the algorithm becomes even more severe.

We summarize Babai's algorithm as Algorithm 6. In general, given a sufficiently orthogonal basis, Babai's algorithm solves *some* version of apprCVP.

Example 12. Consider the lattice $L \subset \mathbb{R}^2$ spanned by the basis $v_1 = (137, 312)$ and $v_2 = (215, -187)$. This is an example from [6, Section 6.6]. We consider the target vector $w = (53172, 81743)$ and want to find the vector $v \in L$ closest to w. First, we determine w in terms of the basis, by inverting the basis matrix and multiplying with w:

```
sage: V = Matrix([[137,312],[215,-187]])
sage: w = vector([53172,81743])
sage: [RR(x) for x in w * V.inverse()]
[296.852274566069, 58.1545971369702]
```

Thus, we find that $w \approx 297v_1 + 58v_2$. We determine that lattice vector:

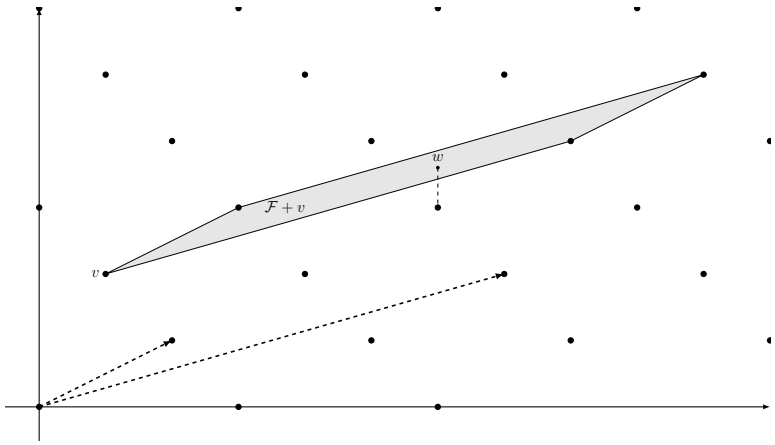

Fig. 6. Babai's algorithm with a highly non-orthogonal basis in two dimensions. Due to the non-orthogonality of the basis vectors (dashed), we see that the closest lattice point is not one of the vertices of the translated fundamental domain.

Algorithm 6: Babai's algorithm

Data: Lattice L with basis $v_1, \ldots, v_n$ and vector $w \in \mathbb{R}^n$
Result: Vector v as proposed solution to CVP for w

Write $w = t_1 v_1 + \cdots + t_n v_n$ with $t_i \in \mathbb{R}$
for $i = 1, \ldots, n$ **do**
  $a_i \leftarrow \lfloor t_i \rceil$
end
$v \leftarrow a_1 v_1 + \cdots + a_n v_n$
return v

```
sage: v = 297*V[0] + 58*V[1]; v
(53159, 81818)
sage: RR(norm(w-v))
76.1183289359403
```

We see that indeed the distance $||w-v||$ is only about 76.12, which is quite short. To check just how orthogonal the basis is, we compare $\det(L) = |\det(V)|$ with $||v_1|| \cdot ||v_2||$, and verify that indeed Hadamard's inequality is very tight:

```
sage: abs(V.determinant())
92699
sage: RR(prod([norm(x) for x in V]))
97096.2353647143
```

5.6 GGH

In the previous section, we saw that if one can find a sufficiently orthogonal basis for a lattice L, then it is likely that one can solve the CVP, a problem which is otherwise considered very hard. The idea of Goldreich, Goldwasser and Halevi with their paper from CRYPTO 1997 [4] was to use this fact as a trapdoor to define a lattice-based encryption scheme.

For Alice and Bob to communicate using GGH, Alice must first pick a private and public keys. For her private key, she chooses a *good* (i.e. highly orthogonal) basis $v_1, \ldots, v_n$ for an n-dimensional lattice L, and for her public key she chooses a *bad* (i.e. highly non-orthogonal) basis $w_1, \ldots, w_n$ for the same lattice L. Such a basis can be computed from $v_1, \ldots, v_1$ by a change of basis using a matrix from $SL_n(\mathbb{Z})$.

With a hold of Alice's public key $w_1, \ldots, w_n$, Bob will choose as his message a small vector $m \in \mathbb{Z}^n$. He also picks randomly a small error vector r. He now computes the ciphertext

$$c = mW + r \tag{54}$$

and sends this to Alice. Since the error vector r is small, Alice can obtain mW by applying Babai's algorithm using her private key, the good basis $v_1, \ldots, v_n$. Finally, she can obtain the message as $m = mWW^{-1}$. The GGH encryption scheme is summarized in Figure 7.

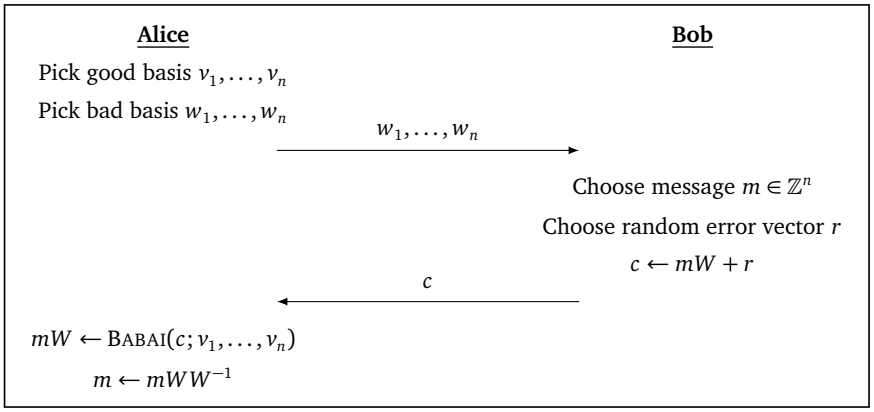

Fig. 7. The GGH encryption scheme. The basis matrix for $w_1, \ldots, w_n$ is denoted W and $\text{BABAI}(c; v_1, \ldots, v_n)$ is a procedure for using Babai's algorithm to solve CVP for target c using basis $v_1, \ldots, v_n$.

Example 13. The following example is from [6, Section 6.8]. We use a lattice L of dimension $n = 3$. Alice picks a basis V which is good, and uses a matrix U to transform it into a bad basis W. Bob picks the message $m = (86, -35, -32)$ and error vector $r = (-4, -3, 2)$, and we see that Alice is able to recover this in the end:

```
sage: V = Matrix([[-97,19,19],[-36,30,86],[-184,-64,78]])
sage: U = Matrix
    ([[4327,-15447,23454],[3297,-11770,17871],[5464,-19506,29617]])

sage: U.determinant()
-1
sage: m = vector([86,-35,-32])
sage: r = vector([-4,-3,2])
sage: W = U*V; W
[-4179163 -1882253   583183]
[-3184353 -1434201   444361]
[-5277320 -2376852   736426]
sage: c = m*W + r; c
(-79081427, -35617462, 11035473)
```

```
sage: v = sum([round(wV[i]) * V[i] for i in range(3)]); v
(-79081423, -35617459, 11035471)
sage: v * W.inverse()
(86, -35, -32)
```

Some care must be taken with respect to the small error vector r. In particular, it is unsafe to use two different values of r for the same message m, or vice versa. For that reason, r should be taken as a hash of the message m. The GGH system was initially conjectured secure for $n > 300$. However, this was before the famous LLL lattice reduction algorithm which we will discuss in Section 5.10. At CRYPTO 1999, Nguyen showed [13] that the original GGH system can be transformed into a modified system in which CVP is easier. This allowed him to solve the posed GGH challenge problems for n up to 350. With $n > 400$, the public key is roughly 128 KB in size [6, Section 6.7].

5.7 NTRU

In this section, we finally describe the probably most well-known lattice-based encryption scheme, NTRU. The design is due to Hoffstein, Pipher and Silverman in 1996. Two very appealing parts of NTRU, compared to RSA, is that NTRU is significantly faster. Also, there are no known attacks on NTRU using quantum computers, as opposed to RSA.

While the security of the NTRU encryption system turns out to be connected to hard lattice-based problems, the description of the algorithm itself does not rely on lattices. Rather, it uses what is known as *convolutional polynomials rings*. We start out by describing these in the following.

Convolutional Polynomial Rings Recall that a *ring* R is a set together with two binary operations $+$ and $\star$ such that R together with $+$ is an Abelian group and

1. $\forall a, b, c \in R : (a \star b) \star c = a \star (b \star c)$,
2. $\forall a \in R : \exists e \in R : a \star e = e \star a = a$,
3. $\forall a, b, c \in R : (a + b) \star c = a \star c + b \star c$, and
4. $\forall a, b, c \in R : a \star (b + c) = a \star b + a \star c$.

Next, we specify rings we will need for the description of the NTRU encryption system.

Definition 25 (Convolution polynomial ring). *Fix a positive integer N. The* ring of convolution polynomials of order N *is the quotient ring*

$$R = \mathbb{Z}[x]/(x^N - 1). \tag{55}$$

The ring of convolution polynomials modulo q of order N *is the quotient ring*

$$R_q = \mathbb{Z}_q[x]/(x^N - 1). \tag{56}$$

In both R and R_q, addition works as usual. We add polynomials, and in the case of R_q we reduce the coefficients modulo q. For multiplication in R, we define an operator $\star$, such that $a(x) \star b(x) = c(x)$, where

$$c_k = \sum_{i+j \equiv k \bmod N} a_i b_j \tag{57}$$

is the coefficient on the term x^k in $c(x)$ for $k = 0, \ldots, N-1$. In R_q we define the same operation, but each c_k is reduced modulo q.

Example 14. Let us let $N = 7$ and $a(x) = 2x + 6x^2 - 3x^3 - 2x^6$ and $b(x) = 5 - x^3 + 4x^4$. We write a Sage function to compute $a(x) \star b(x)$:

```
# star product for conv. poly. rings
# assume a and b are polynomials rep. as lists of equal length
def star(a,b):
   N = len(a)
   return [sum([a[i] * b[(k-i)%N] for i in range(N)]) for k in
       range(N)]
```

We now use it to compute $c(x) = a(x) \star b(x)$:

```
sage: star([0,2,6,-3,0,0,-2],[5,0,0,-1,4,0,0])
[-12, 10, 32, -23, -2, 2, 17]
```

We see that $c(x) = -12 + 10x + 32x^2 - 23x^3 - 2x^4 + 2x^5 + 17x^6 \in \mathbb{Z}[x]/(x^7 - 1)$.

There is a simple mapping of a polynomial $a(x) \in R$ to R_q obtained by reducing the coefficients of $a(x)$ modulo q. This mapping is a ring homomorphism, and in particular it holds that

$$\begin{aligned} (a(x) + b(x)) \bmod q &= (a(x) \bmod q) + (b(x) \bmod q), \quad \text{and} \\ (a(x) \star b(x)) \bmod q &= (a(x) \bmod q) \star (b(x) \bmod q). \end{aligned} \tag{58}$$

In order to map a polynomial $a(x) \in R_q$ to R, we define the *centered lift*.

Definition 26 (Centered lift). *Let $a(x) \in R_q$. The* centered lift *of $a(x)$ to R is the map $CL : R_q \to R$ where $CL(a(x))$ is the unique polynomial $a'(x) \in R$ which satisfies $a'(x) \bmod q = a(x)$, and where the coefficients of $a'(x)$ satisfy $-q/2 < a'_i \leq q/2$.*

The following proposition is helpful when using NTRU, as we shall see, because it tells us for a fixed N and q, when a polynomial in R_q has an inverse.

Proposition 4. *Let q be a prime. Then $a(x) \in R_q$ has a multiplicative inverse* if and only if $\gcd(a(x), x^N - 1) = 1$ *in $\mathbb{Z}_q$. This inverse $a(x)^{-1} \in R_q$ can be computed using Euclid's extended algorithm.*

Finally, we define *ternary polynomials* which are polynomials with a particular structure that we need to describe the NTRU encryption system.

Definition 27 (Ternary polynomials). *For any positive integers d_1 and d_2, we define the* ternary polynomials *as*

$$\mathscr{T}(d_1, d_2) = \left\{ a(x) \in R \;\middle|\; \begin{array}{r} a(x) \text{ has } d_1 \text{ coefficients equal to } 1 \\ a(x) \text{ has } d_2 \text{ coefficients equal to } -1 \\ a(x) \text{ has all other coeffients equal } 0 \end{array} \right\} \tag{59}$$

Encryption using NTRU. To use NTRU, we start by picking a fixed integer $N \geq 1$ and two moduli p and q as well as a positive integer d. We then consider the convolution polynomial rings R, R_p and R_q as defined above. To map a polynomial $a(x) \in R$ to either R_p or R_q, we reduce the coefficients modulo p respectively q. To map in the other direction, we use the centered lift. For the parameters, we require that N and p be prime, and that $\gcd(N, q) = \gcd(p, q) = 1$. We also require that $q > (6d + 1)p$.

To use NTRU, the communicating parties need to first agree on parameters (N, p, q, d), satisfying the constraints described. First, Alice picks randomly two polynomials

$$f(x) \in \mathscr{T}(d + 1, d) \quad \text{and} \quad g(x) \in \mathscr{T}(d, d), \tag{60}$$

where $f(x)$ should be invertible in both R_p and R_q (if a chosen $f(x)$ turns out not to be, which can be determined using Proposition 4, she chooses a new one). She now computes the inverses

$$F_q(x) = f(x)^{-1} \in R_q \quad \text{and} \quad F_p(x) = f(x)^{-1} \in R_p. \tag{61}$$

Now she computes

$$h(x) = F_q(x) \star g(x) \in R_q. \tag{62}$$

Alice's private key consists of the pair $(f(x), F_p(x))$ and her public key is $h(x)$. If she does not want to store the full private key, she can store just $f(x)$ and compute $F_p(x)$ when required.

Now, Bob chooses a message which will be a polynomial $m(x) \in R$, for which all coefficients lie in the range $-p/2$ and $p/2$, such that $m(x)$ is the centered lift of some polynomial in R_p. He also chooses a random polynomial $r(x) \in \mathscr{T}(d,d)$ and computes

$$c(x) \equiv p \cdot h(x) \star r(x) + m(x) \bmod q \in R_q. \tag{63}$$

The ciphertext is now the polynomial $c(x) \in R_q$.

To decrypt the ciphertext, Alice computes first

$$a(x) \equiv f(x) \star e(x) \bmod q. \tag{64}$$

She now computes the centered lift $a'(x) = CL(a(x)) \in R$ and computes

$$m(x) \equiv F_p(x) \star a'(x) \bmod p. \tag{65}$$

If the parameters were chosen to satisfy the aforementioned constraints, and in particular if $q > (6d+1)p$, then the $m(x)$ obtained by Alice equals that of Bob. We do not give a proof here, but refer the interested reader to [6, Section 6.10]. We note that the condition $q > (6d+1)p$ means that decryption *never fails*. However, even with a smaller value of q, one can keep the probability of decryption failure very small, and trade off this probability with storing a smaller q parameter.

As was the case for GGH, also NTRU is a probabilistic encryption system. As such, Bob should never repeat $m(x)$ using a different $r(x)$, or vise versa. Again, this means that one typically takes $r(x)$ has a hash of $m(x)$.

As already mentioned, one of the motivations behind NTRU is its speed. The most time-consuming operation in NTRU is the $\star$ operation. In general, a convolution product $a(x) \star b(x)$ costs N^2 multiplications. However, the polynomials $f(x), g(x)$ and $r(x)$ are ternary, and as such, any convolution product with them can be computed using no multiplications (as coefficients are either 1 or -1), and cost only about $\frac{2}{3}N^2$ additions and subtractions. The NTRU encryption and decryption take $O(N^2)$ steps, where each step is very fast.

Example 15. We give an example of NTRU encrypting using $N = 7, p = 3, q = 41$ and $d = 2$. We let $f(x) = x^6 - x^4 + x^3 + x^2 - 1$ and $g(x) = x^6 + x^4 - x^2 - x$. First, we compute $F_q(x)$ and $F_p(x)$:

```
sage: N = 7
sage: p = 3
sage: q = 41
sage: d = 2
sage: Rbase.<a> = PolynomialRing(ZZ)
sage: Rpbase.<b> = PolynomialRing(GF(p))
sage: Rqbase.<c> = PolynomialRing(GF(q))
sage: R.<x> = Rbase.quotient(a^N - 1)
sage: Rp.<y> = Rpbase.quotient(b^N - 1)
```

```
sage: Rq.<z> = Rqbase.quotient(c^N - 1)
sage: Fq = (c^6 - c^4 + c^3 + c^2 - 1).inverse_mod(c^N - 1);
    Fq
8*c^6 + 26*c^5 + 31*c^4 + 21*c^3 + 40*c^2 + 2*c + 37
sage: Fp = (b^6 - b^4 + b^3 + b^2 - 1).inverse_mod(b^N - 1);
    Fp
b^6 + 2*b^5 + b^3 + b^2 + b + 1
```

Thus,

$$\begin{aligned} F_q(x) &= 8x^6 + 26x^5 + 31x^4 + 21x^3 + 40x^2 + 2x + 37 \quad \text{and} \\ F_p(x) &= x^6 + 2x^5 + x^3 + x^2 + x + 1. \end{aligned} \tag{66}$$

We now compute $h(x) = F_q(x) \star g(x) \in R_q$ (note that $g(x) \in R_q$ is expressed by the list `[0,40,40,0,1,0,1]`

```
sage: star([37,2,40,21,31,26,8],[0,40,40,0,1,0,1])
[1383, 1871, 1607, 1719, 2503, 2090, 2357]
sage: [x % q for x in _]
[30, 26, 8, 38, 2, 40, 20]
```

So we find that $h(x) = 20x^6+40x^5+2x^4+38x^3+8x^2+26x+30$. Let's say Bob wants to send the message $m(x) = -x^5+x^3+x^2-x+1$ to Alice using the random polynomial $r(x) = x^6-x^5+x-1$. He computes the ciphertext

```
sage: r = x^6 - x^5 + x - 1; p*r
3*x^6 - 3*x^5 + 3*x - 3
sage: star([-3,3,0,0,0,-3,3], [30,26,8,38,2,40,20])
[24, -78, 162, -204, 168, -144, 72]
sage: [l % q  for l in _]
[24, 4, 39, 1, 4, 20, 31]
sage: m = -x^5 + x^3 + x^2 - x + 1
sage: 31*x^6 + 20*x^5 + 4*x^4 + x^3 + 39*x^2 + 4*x + 24 + m
31*x^6 + 19*x^5 + 4*x^4 + 2*x^3 + 40*x^2 + 3*x + 25
```

Thus, we find $c(x) = 31x^6 + 19x^5 + 4x^4 + 2x^3 + 40x^2 + 3x + 25$. To decrypt, Alice computes first $f(x) \star c(x)$:

```
sage: c = 31*x^6 + 19*x^5 + 4*x^4 + 2*x^3 + 40*x^2 + 3*x + 25
sage: f = x^6 - x^4 + x^3 + x^2 - 1
sage: [l % q for l in star(c.list(), f.list())]
[40, 1, 40, 40, 33, 10, 1]
```

We centerlift the result to get $a(x) = x^6 + 10x^5 - 8x^4 - x^3 - x^2 + x - 1$. Finally, she computes $F_p(x) \star a(x) \bmod p$:

```
sage: Fp = x^6 + 2*x^5 + x^3 + x^2 + x + 1
sage: a = x^6 + 10*x^5 - 8*x^4 - x^3 - x^2 + x - 1
sage: [l % p for l in star(Fp.list(), a.list())]
[1, 2, 1, 1, 0, 2, 0]
```

By centerlifting the result modulo p, we find the original message $m(x) = -x^5 + x^3 + x^2 - x + 1$.

5.8 Mathematical Problems Underlying NTRU

In NTRU, there is a special relationship between the polynomials. Namely,

$$f(x) \star h(x) = g(x) \bmod q, \tag{67}$$

where $f(x) \in \mathcal{T}(d+1,d)$ and $g(x) \in \mathcal{T}(d,d)$. Thus, $f(x)$ and $g(x)$ have small coefficients, while $h(x)$ is expected to have random-looking coefficients. This relationship can be potentially used by an attacker. Thus, the problem of recovering the NTRU private key boils down to the following problem.

Definition 28 (The NTRU key recovery problem). *Given $h(x)$, find ternary polynomials $f(x)$ and $g(x)$ satisfying $f(x) \star h(x) = g(x) \bmod q$.*

We remark that a solution to the NTRU key recovery problem is not unique. If $(f(x), g(x))$ is one solution, then $(x^k \star f(x), x^k \star g(x))$ is also a solution for any k such that $0 \leq k < N$. Such a polynomial $x^k \star f(x)$ is called a *rotation* of $f(x)$, because the coefficients have been cyclically rotated by k positions. Decryption in NTRU with a rotated key $x^k \star f(x)$ results in a rotated message $x^k \star m(x)$.

First, we consider how hard it would be for an attacker to attempt to recover the NTRU private key by brute-force search. The attacker can verify if she has found the private key $f(x)$ by checking if $f(x) \star h(x) \bmod q$ is a ternary polynomial (as it is extremely unlikely that this is the case for any $f(x)$ other than the private key or any rotation hereof – for an argument explaining this, see e.g. [6, Section 6.10]). In general, an element of $\mathcal{T}(d_1, d_2)$ can be specified by first choosing d_1 coefficients to equal 1 and then select d_2 of the remaining $N - d_1$ coefficients to equal -1. Thus,

$$\begin{aligned} \sharp\mathcal{T}(d_1,d_2) &= \binom{N}{d_1}\binom{N-d_1}{d_2} \\ &= \frac{N!}{d_1!d_2!(N-d_1-d_2)!}. \end{aligned} \tag{68}$$

This number is maximized when d_1 and d_2 are both approximately $N/3$. An attacker must randomly try polynomials in $\mathcal{T}(d+1,d)$ until she finds a decryption key. However, there are N rotations of the correct private key that will also work, so there are N working candidates in $\mathcal{T}(d+1,d)$. As such, it will take an attacker about $\frac{\sharp\mathcal{T}(d+1,d)}{N}$ attempts until some rotation of $f(x)$ is found.

Example 16. Consider the NTRU parameters $(N,p,q,d) = (251,3,257,83)$. We compute the expected number of keys an attacker has to try to find the correct key.

```
def T(N,d1,d2):
    return factorial(N) / (factorial(d1) * factorial(d2) *
        factorial(N - d1 - d2))

sage: N, p, q, d = 251, 3, 257, 83
sage: 1/N * binomial(N, d+1) * binomial(N - (d+1), d)
74595695058813537027492584550215094243400691930422854 \
   094129733704750462518001993773148631711276311976969218940250000

sage: RR(log(_,2))
381.598895191654
```

Thus, we see that an attacker must try $\sharp\mathcal{T}(d+1,d)/N \approx 2^{381.6}$ keys before the correct one is found.

An observation by Odlyzko means that an attacker can use a standard time-memory-tradeoff in a collision search to lower the time complexity in turn for higher memory complexity. In

particular, the attacker will brute-force search over pairs of polynomials

$$f_1(x) = \sum_{0 \le i < N/2} a_i x^i \quad \text{and} \quad f_2(x) = \sum_{N/2 \le i < N} a_i x^i, \tag{69}$$

where it holds that $f_1(x) + f_2(x) \in \mathcal{T}(d+1, d)$. She computes now

$$f_1(x) \star h(x) \bmod q \quad \text{and} \quad -f_2(x) \star h(x) \bmod q \tag{70}$$

and puts them into bins depending on their coefficients. Now, when a polynomial from each list lands in the same bin, then

$$(f_1(x) + f_2(x)) \star h(x) \bmod q \tag{71}$$

has small coefficients, and hence $f_1(x)+f_2(x)$ is a decryption key. For more details, we refer to [7]. The effect of this collision algorithm is that the complexity more or less becomes $\sqrt{\sharp\mathcal{T}(d+1,d)}$.

5.9 NTRU and Lattices

In this section we give a new view on NTRU, namely in the context of lattices. As it turns out, the NTRU key recovery problem can be formulated as a SVP in a certain type of lattices.

Definition 29 (NTRU lattice). *Let*

$$h(x) = h_0 + h_1 x + h_2 x^2 + \cdots + h_{N-1} x^{N-1} \tag{72}$$

be an NTRU public key. The NTRU lattice L_h^{NTRU} *associated to* $h(x)$ *is the* $2N$*-dimensional lattice spanned by the rows of the matrix*

$$M_h^{\text{NTRU}} = \left(\begin{array}{cc|ccc} 1 & 0 \cdots & 0 & h_0 & h_1 \cdots & h_{N-1} \\ 0 & 1 \cdots & 0 & h_{N-1} & h_0 \cdots & h_{N-2} \\ \vdots & \vdots \ddots & \vdots & \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & 1 & h_1 & h_2 \cdots & h_0 \\ \hline 0 & 0 \cdots & 0 & q & 0 \cdots & 0 \\ 0 & 0 \cdots & 0 & 0 & q \cdots & 0 \\ \vdots & \vdots \ddots & \vdots & \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & 0 & 0 & 0 \cdots & q \end{array}\right). \tag{73}$$

We note that M_h^{NTRU} is composed of four block matrices. Starting in the upper left corner and going clockwise, we have the identity matrix; a cyclical permutation of the $h(x)$ coefficients; the identity matrix multiplied by q; and the zero matrix. Thus, we abbreviate the matrix as

$$M_h^{\text{NTRU}} = \begin{pmatrix} I & h \\ 0 & qI \end{pmatrix}. \tag{74}$$

Let us consider polynomials $a(x), b(x) \in R$ and associate them with a $2N$-dimensional vector

$$(a, b) = (a_0, \ldots, a_{N-1}, b_0, \ldots, b_{N-1}) \in \mathbb{Z}^{2N}. \tag{75}$$

In Propositions 5 and 6, we make it clear just how NTRU is related to the NTRU lattice, and in particular how the NTRU key recovery problem is associated with hard lattice problems. We do not give any proofs here, but refer the reader to [6, Section 6.11].

Proposition 5. *Suppose $h(x)$ was constructed using NTRU with polynomials $f(x)$ and $g(x)$, such that $f(x) \star h(x) \equiv g(x) \bmod q$. Let $u(x) \in R$ be the polynomial s.t. $f(x) \star h(x) \equiv g(x) + q \cdot u(x)$. Then*

$$(f, -u) M_h^{\text{NTRU}} = (f, g). \tag{76}$$

Proposition 6. *Let (N, p, q, d) be NTRU parameters where we assume $d \approx N/3$ and $q \approx 6d \approx 2N$. Let L_h^{NTRU} be the lattice associated with the private key (f, g). Then*

1. $\det(L_h^{\text{NTRU}}) = q^N$,
2. $||(f, g)|| \approx \sqrt{4d} \approx \sqrt{4N/3} \approx 1.155\sqrt{N}$,
3. *The Gaussian heuristic says the shortest non-zero vector in L_h^{NTRU} has length $\sigma(L_h^{\text{NTRU}}) \approx \sqrt{\frac{Nq}{\pi e}} \approx 0.484N$.*

Thus, when N is very large it is likely that a short vector in L_h^{NTRU} equals (f, g) or some rotation of it. Furthermore, $\frac{||(f,g)||}{\sigma(L)} \approx \frac{2.39}{\sqrt{N}}$, so (f, g) is a factor of $O(1/\sqrt{N})$ shorter than predicted by the Gaussian heuristic.

It follows directly from Proposition 6 that if an attacker can solve SVP in the NTRU lattice, then she can recover the private key $f(x)$ for the system. In general, if she can solve apprSVP within a factor N^ϵ where $\epsilon < 1/2$, then the shortest vector will likely work as a decryption key [6, Section 6.11].

The LLL algorithm, which we discuss in Section 5.10, runs in polynomial time and can solve apprSVP to within a factor of 2^N, but if N is large then LLL does not find very small vectors in L_h^{NTRU}. A generalization called BKZ-LLL is able to find very small vectors, and solves apprSVP to within a factor $\beta^{2N/\beta}$, where β is a parameter of the algorithm. However, the running time is exponential in β. The operating characteristics of lattice reduction algorithms such as LLL and BKZ-LLL are not very well understood, so the security of lattice-based cryptosystems are typically determined experimentally. One takes a sequence of parameters (N, q, d) where N increases, and certain rations involving all parameters are kept constant. Then, one runs BKZ-LLL using increasing β parameter until a shortest vector is found. With enough data points, one can find a fitting linear function

$$\log(\text{running time}) = AN + B, \tag{77}$$

and extrapolate from this an idea about how long a private key recovery using lattice reduction would take for larger values of N.

5.10 The LLL Lattice Reduction Algorithm

We have seen how the security of the GGH and NTRU cryptosystems rely on the hardness of solving apprCVP and/or apprSVP in lattices. In this section we describe LLL, named after Lenstra, Lenstra and Lovász, which solves these problems to within a factor of C^n where C is a small constant and n is the dimension of the lattice. Thus, the LLL algorithm comes close to solving CVP and SVP when n is small, but does not succeed when n is large. At the end of the day, the best reason we would believe a lattice-based cryptosystem to be secure is, if the lattice reduction algorithms such as LLL would not be able to solve apprSVP or apprCVP to within a factor of, say, $O(\sqrt{n})$.

Say we are given a basis $B = \{v_1, \ldots, v_n\}$ for a lattice L. The goal of the LLL algorithm is to transform this basis into a *better* basis, either in the sense to make the basis vectors *shorter* or to make them *more orthogonal*. Recall Hadamard's inequality, which states $\det(L) = \text{Vol}(\mathscr{F}) \leq \prod_{i=1}^n ||v_i||$, where the more orthogonal basis means a tighter bound.

We start off by creating a Gram-Schmidt orthogonal basis $B^* = \{v_1^*, \ldots, v_n^*\}$ which is orthogonal to the vector space spanned by B. Note that B^* is *not* a basis for L, however $\det(L) = \prod_{i=1}^{n} ||v_i^*||$, as stated in the following theorem which we give without proof.

Theorem 7. *Let $B = \{v_1, \ldots, v_n\}$ be a basis for a lattice L and let $B^* = \{v_1^*, \ldots, v_n^*\}$ be the associated Gram-Schmidt orthogonal basis. Then*

$$\det(L) = \prod_{i=1}^{n} ||v_i^*||. \tag{78}$$

Proof. We refer to [6, Section 6.12]. □

Definition 30 (Orthogonal complement). *Let V be a vector space and let $W \subset V$ be a subspace of V. The* orthogonal complement *of W in V is*

$$W^{\perp} = \{v \in V \mid \forall w \in W : v \cdot w = 0\}. \tag{79}$$

Note, that the orthogonal complement $W^{\perp}$ is also a subspace of V. We use the concept of the orthogonal complement together with the Gram-Schmidt orthogonal basis to define what it means for a basis to be *LLL reduced*.

Definition 31 (LLL reduced basis). *Let $B = \{v_1, \ldots, v_n\}$ be a basis for a lattice L and let $B^* = \{v_1^*, \ldots, v_n^*\}$ be the associated Gram-Schmidt orthogonal basis. The basis B is said to be* LLL reduced *if it satisfies the two following conditions:*

1. *Size condition:*
$$\forall 1 \leq j < i \leq n : |\mu_{i,j}| = \frac{|v_i \cdot v_j^*|}{||v_j^*||^2} \leq \frac{1}{2}, \quad \textit{and} \tag{80}$$
2. *Lovász condition:*
$$\forall 1 < i \leq n : ||v_i^*||^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) ||v_{i-1}^*||^2. \tag{81}$$

The important result of Lenstra, Lenstra and Lovász given in [9] is, that an LLL reduced basis can be computed in polynomial time, and furthermore it has very desirable properties, as made precise in Theorem 8. The LLL algorithm is summarized in Algorithm 7, where, at each step, $v_1^*, \ldots, v_n^*$ denotes the Gram-Schmidt orthogonal basis obtained from the *current* values of $v_1, \ldots, v_n$.

Theorem 8. *Let L be a lattice of dimension n. Any LLL reduced basis $v_1, \ldots, v_n$ for L has the following properties,*

$$\begin{aligned} \prod_{i=1}^{n} ||v_i|| &\leq 2^{n(n-1)/4} \cdot \det(L) \quad \textit{and} \\ \forall 1 \leq j \leq i \leq n : ||v_j|| &\leq 2^{(i-1)/2} \cdot ||v_i^*||, \end{aligned} \tag{82}$$

where $v_1^*, \ldots, v_n^*$ is the associated Gram-Schmidt orthogonal basis. Furthermore, the initial vector v_1 satisfies

$$||v_1|| \leq 2^{(n-1)/4} \cdot |\det(L)|^{1/n} \quad \textit{and} \quad ||v_1|| \leq 2^{(n-1)/2} \min_{0 \neq v \in L} ||v||. \tag{83}$$

Algorithm 7: LLL lattice reduction algorithm

Data: A basis $\{v_1, \ldots, v_n\}$ for a lattice L
Result: An LLL reduced basis

$k \leftarrow 2$
$v_1^* \leftarrow v_1$
while $k \leq n$ **do**
  for $j = k-1, \ldots, 1$ **do**
    $v_k \leftarrow v_k - \lfloor \mu_{k,j} \rceil v_j$; /* Reduce size of v_k */
  end
  if $||v_k^*||^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) ||v_{k-1}^*||^2$ **then**
    $k \leftarrow k+1$; /* Lovász condition satisfied */
  else
    Swap v_{k-1} and v_k
    $k \leftarrow \max\{k-1, 2\}$
  end
end
return $\{v_1, \ldots, v_n\}$

Example 17. We give an example of the LLL lattice reduction algorithm for a 6-dimensional lattice spanned by the basis matrix

$$V = \begin{pmatrix} 19 & 2 & 32 & 46 & 3 & 33 \\ 15 & 42 & 11 & 0 & 3 & 24 \\ 43 & 15 & 0 & 24 & 4 & 16 \\ 20 & 44 & 44 & 0 & 18 & 15 \\ 0 & 48 & 35 & 16 & 31 & 31 \\ 48 & 33 & 32 & 9 & 1 & 29 \end{pmatrix}. \tag{84}$$

First, we implement the LLL algorithm in Sage. Note that Sage has already functionality for computing the Gram-Schmidt orthogonal basis and the μ matrix, which we make use of.

```
# LLL lattice reduction algorithm. Input V is basis matrix for
    L.
def LLL(V):
   n = (V.dimensions())[0]
   k = 1
   while k < n:
      W, mu = V.gram_schmidt()
      for j in range(k-1, -1, -1):
         V[k] = V[k] - round(mu[k][j]) * V[j]

      W, mu = V.gram_schmidt()
      if norm(W[k])^2 >= ((3/4) - mu[k][k-1]^2) * norm(W[k-1])
          ^2:
         k = k + 1
      else:
         tmp = V[k-1]
         V[k-1] = V[k]
         V[k] = tmp
         k = max([k-1, 2])
   return V
```

We now use this function to compute the LLL reduced basis for V and check that $\det(V) = \pm\det(LLL(V))$. We also check the norms of the basis vectors in the LLL reduced basis.

```
sage: V = Matrix
    ([[19,2,32,46,3,33],[15,42,11,0,3,24],[43,15,0,24,4,16], \
    [20,44,44,0,18,15],[0,48,35,16,31,31],[48,33,32,9,1,29]])
sage: LLL(V)
[ 15  42  11   0   3  24]
[  0  16  -1 -15 -18  22]
[  7 -12  -8   4  19   9]
[-10 -24  21 -15  -6 -11]
[-20   4  -9  16  13  16]
[  6   7  20  21  -8  12]
sage: V.determinant(), LLL(V).determinant()
(777406251, -777406251)
sage: [RR(norm(v)) for v in LLL(V)]
[26.7394839142419,
 35.9165699921359,
 45.1774279923061,
 42.0475920832573,
 34.3220046034610,
 33.6749164809655]
```

We see that the first vector in the LLL reduced basis has length $||v|| \approx 26.74$.

The proof of Theorem 8 and the complete understanding of the LLL algorithm is beyond this note. However, we try to give some intuition into the algorithm. First, note that the LLL algorithm consists of three major parts: size reduction, the Lovász condition and the swapping of v_{k-1} and v_k for some values of k. The purpose of the size reduction is, quite obviously, to obtain a reduced basis with as small basis vectors as possible. By subtracting integer multiples of other basis vectors from v_k, we obtain a different basis which spans the same lattice. Instead of making all basis vectors smaller at once, we do them in stages (i.e. from v_1 up to v_k each time). This is because the ordering step which is taken later (provided the Lovász condition is not satisfied), will affect this resizing through changes to $\mu_{i,j}$. The goal of the LLL algorithm is to produce a list of short basis vectors of increasing size. Letting L_ℓ, where $1 \leq \ell \leq n$, be the lattice spanned by $v_1, \ldots, v_\ell$, each L_ℓ is a sublattice of $L_n = L$. By employing the swapping step, the LLL algorithm tries to find an ordering of the (reduced) basis vectors, such as to minimize the determinant $\det(L_\ell)$ for each sublattice L_ℓ of L.

While this does not fully explain why the algorithm works, it gives some intuition as to the steps performed, and why the result is an LLL reduced basis. One can prove this, and also that the running time is polynomial. If efficiently implemented, it terminates after at most $O(n^6(\log T)^3)$ basic operations, where $T = \max_{1\leq i\leq n} ||v_i||$. The most important part to take away from the LLL algorithm is evident from Theorem 8, namely that it returns an LLL reduced basis for which $||v_1|| \leq 2^{(n-1)/2} \min_{0\neq v\in L} ||v||$. In other words, it solves apprSVP to within a factor $2^{(n-1)/2}$. To solve apprCVP, one can directly apply first LLL reduction to obtain a *reasonably* orthogonal basis, and then apply Babai's algorithm to find the correct closest vector within a factor C^n where C is some small constant.

References

1. Roberto M. Avanzi, Henri Cohen, Christophe Doche, Gerhard Frey, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*.

2. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
3. Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.
4. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.
5. Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
6. J. Hoffstein, J. Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematics. Springer, 2008.
7. Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Ntru cryptosystems technical report #004, version 2: A meet-in-the-middle attack on an ntru private key. `https://www.securityinnovation.com/uploads/Crypto/NTRUTech004v2.pdf`. Accessed on May 8, 2015.
8. Antoine Joux. A one round protocol for tripartite diffie-hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, ANTS-IV, pages 385–394, London, UK, UK, 2000. Springer-Verlag.
9. A.K. Lenstra, Jr. Lenstra, H.W., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
10. Jr. Lenstra, H. W. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):pp. 649–673, 1987.
11. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 1996.
12. Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.
13. Phong Q. Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 1999.
14. Igor A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Math. Comput.*, 67(221):353–356, 1998.
15. Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *J. Cryptology*, 12(3):193–196, 1999.
16. L.C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Discrete Mathematics and Its Applications. CRC Press, 2008.