Cryptography
Week #10:
Elliptic Curve and Discrete Lattice Cryptography

Rogério Reis, rogerio.reis@fc.up.pt
DCC FCUP

December, 16th 2022

Elliptic Curve Cryptography (*ECC*)

The introduction of elliptic curve cryptography (ECC) in 1985 revolutionized the way we do public-key cryptography. ECC is more powerful and efficient than alternatives like RSA and classical Diffie– Hellman (ECC with a 256-bit key is stronger than RSA with a 4096-bit key), but it's also more complex.

Although first introduced in 1985, ECC wasn't adopted by standardization bodies until the early 2000s, and it wasn't seen in major toolkits until much later: OpenSSL added ECC in 2005, and the OpenSSH secure connectivity tool waited until 2011. But modern systems have few reasons not to use ECC.

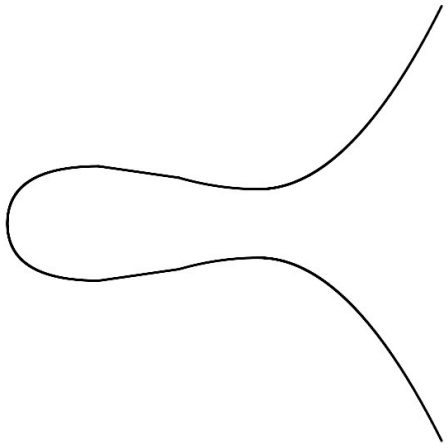
What are Elliptic Curves?

Definition (Elliptic Curve)

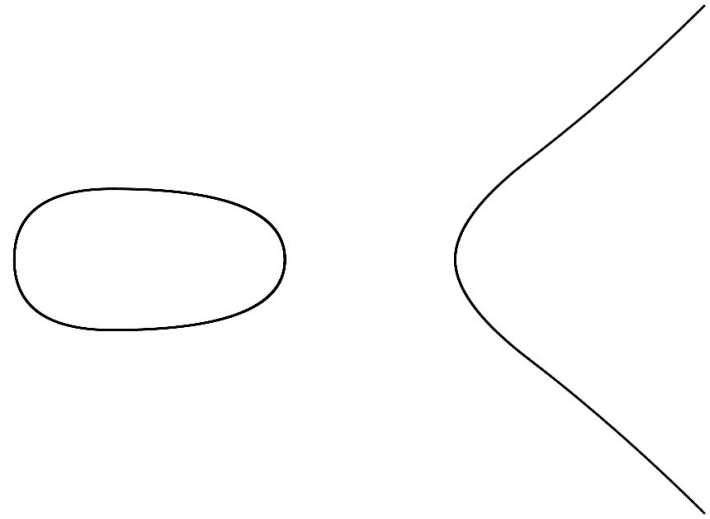
An elliptic curve is the set of solutions to an equation of the form

$$Y^2 = X^3 + AX + B.$$

Equations of this type are called Weierstrass equations.



$E_1 : Y^2 = X^3 - 3X + 3$

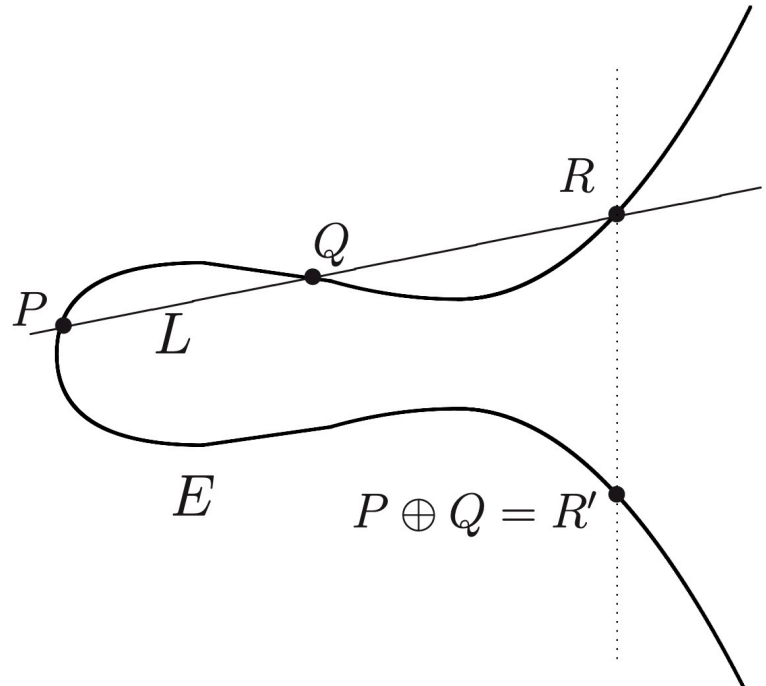


$E_2 : Y^2 = X^3 - 6X + 5$

Addition of two points in a EC

Let P and Q be two points on an elliptic curve E. We start by drawing the line L through P and Q. This line L intersects E at three points, namely P, Q, and one other point R. We take that point R and reflect it across the x-axis (i.e., we multiply its Y-coordinate by -1) to get a new point R'. We write

$$P \oplus Q = R'.$$



An example (1)

Let E be the elliptic curve

$$E : Y^2 = X^3 - 15X + 18.$$

The points $P = (7, 16)$ and $Q = (1, 2)$ are on the curve E. The line L connecting them is given by the equation

$$L : Y = \frac{3}{7}X - \frac{1}{3}.$$

Using this equation in E we get

$$\begin{aligned}
\left(\frac{7}{3}X - \frac{1}{3}\right)^2 &= X^3 - 15X + 18 \\
\frac{40}{9}X^2 - \frac{14}{9} + \frac{1}{9} &= X^3 - 15X + 18 \\
0 &= X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9}.
\end{aligned}$$

We need to find the roots of this cubic polynomial. In general, finding the roots of a cubic is difficult. However, in this case we already know two of the roots, namely $X = 7$ and $X = 1$, since we know that P and Q are in the intersection $E \cap L$. It is then easy to find the other factor,

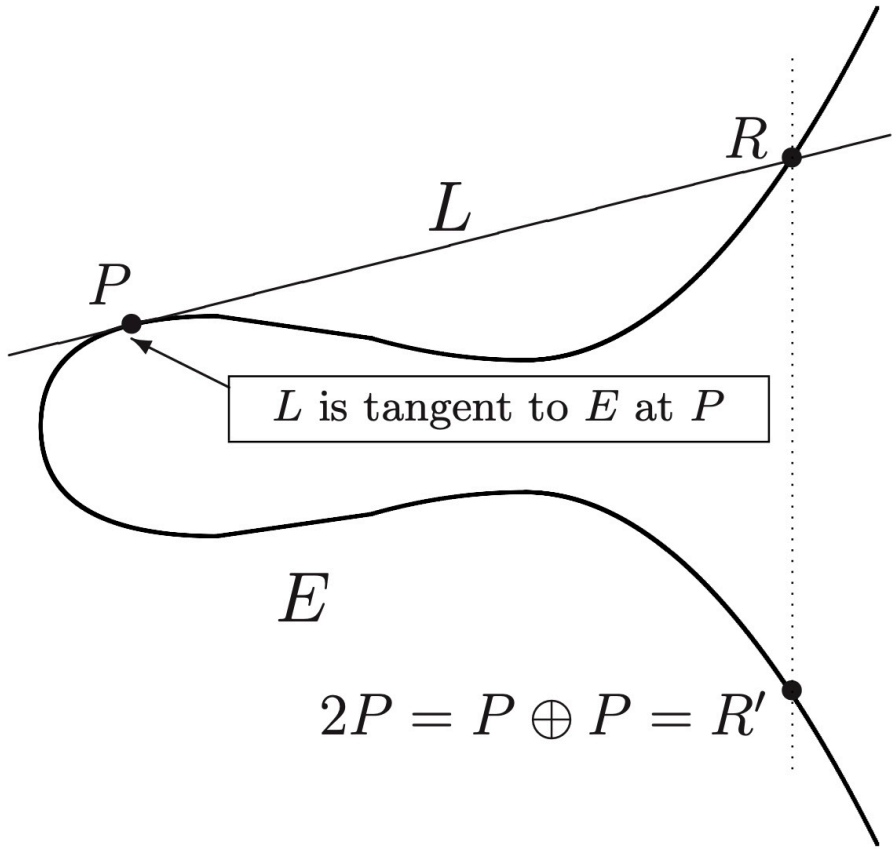
$$X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9} = (X - 7)(X - 1)\left(X + \frac{23}{9}\right).$$

Computing the corresponding value of Y we get $R = (-\frac{23}{9}, \frac{170}{27})$. And thus

$$P \oplus Q = \left(-\frac{23}{9}, -\frac{170}{27}\right).$$

What if $P = Q$?

Imagine what happens to the line L connecting P and Q if the point Q slides along the curve and gets closer and closer to P. In the limit, as Q approaches P, the line L becomes the tangent line to E at P.



Example (2)

Using the previous example let us compute $P \oplus P$.

$$2Y\frac{dY}{dX} = 3X^3 - 15, \quad \text{so} \quad \frac{dY}{dX} = \frac{3X^2 - 15}{2Y}.$$

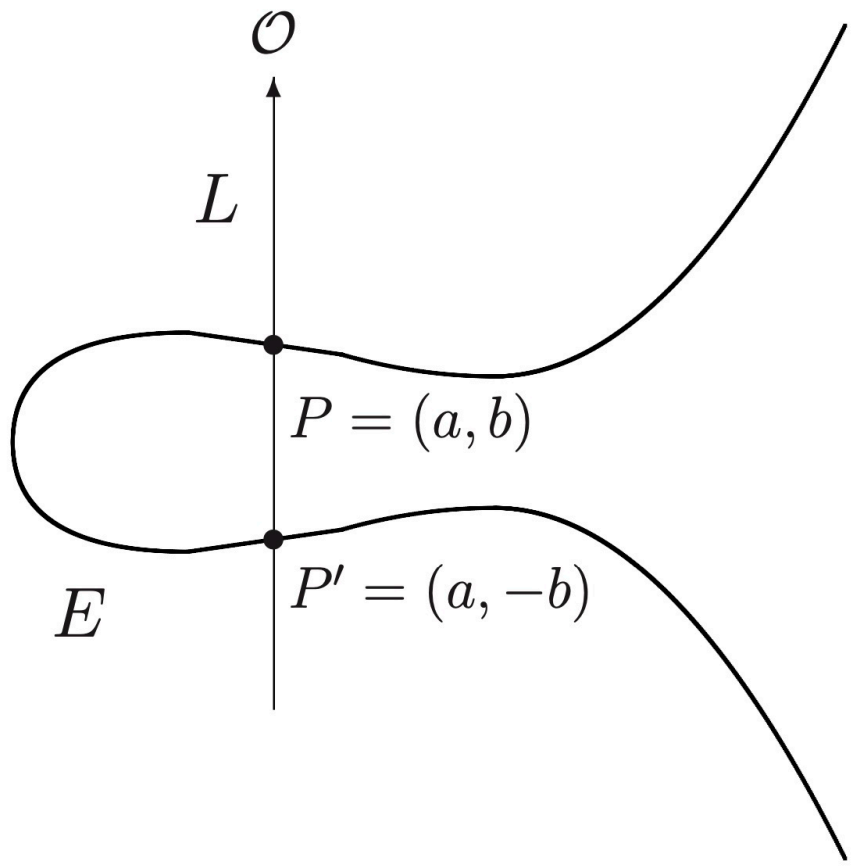
Using the coordinates of $P = (7, 16)$ gives slope $\lambda = 33$, so the tangent line to E at P is given by the equation

$$L : Y = \frac{33}{8}X - \frac{103}{8}.$$

$$\begin{aligned}
\left(\frac{33}{8}X - \frac{103}{8}\right)^2 &= X^3 - 15X + 8 \\
X^3 - \frac{1089}{64}X^2 + \frac{2919}{32}X - \frac{9457}{64} &= 0 \\
(X-7)^2\left(X - \frac{193}{64}\right) &= 0.
\end{aligned}$$

Thus, $2P = P \oplus P = (\frac{193}{64}, \frac{223}{512})$.

What about $(a, b) \oplus (a, -b)$?



Simply add a point to every Elliptic Curve: $P \oplus P' = \mathcal{O}$.

Theorem

Let E be an elliptic curve. The addition law on E has the following properties:

i) $P + \mathcal{O} = \mathcal{O} + P = P \ (\forall P)$

ii) $P + (-P) = \mathcal{O} \ (\forall P)$

iii) $(P + Q) + R = P + (Q + R) \ (\forall P, Q, R)$

iv) $P + Q = Q + P \ (\forall P, Q)$

Thus and Elliptic Curve is an Abelian group.

Elliptic Curve Addition Algorithm

Let

$$E : Y^2 = X^3 + AX + B$$

be an elliptic curve and let P_1 and P_2 be points on E.

1. If $P_1 = \mathcal{O}$, then $P_1 + P_2 = P_2$.
2. Otherwise, if $P_2 = \mathcal{O}$, then $P_1 + P_2 = P_1$.
3. Otherwise, let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.
4. If $x_1 = x_2$ and $y_1 = -y_2$, then $P_1 + P_2 = \mathcal{O}$.
5. Otherwise,

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2, \\ \frac{3x_1^2 + A}{2y_1} & \text{if } P_1 = P_2. \end{cases} \qquad \begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

$P_1 + P_2 = (x_3, y_3)$.

Multiplication (by an integer)

$$nP = \underbrace{P + P + P + \cdots + P}_{n \text{ copies}}.$$

To compute kP efficiently the naive technique of adding P by applying the addition law k–1 times is far from optimal. For example, if k is large (of the order of, say, 2^{256}) as it occurs in elliptic curve–based crypto schemes, then computing k–1 additions is downright infeasible.

But there's a trick: you can gain an exponential speed-up by adapting the technique discussed in "Fast Exponentiation Algorithm" to compute $x^e \pmod{n}$. For example, to compute $8P$ in three additions instead of seven using the naive method, you would first compute $P_2 = P + P$,then $P_4 = P_2 + P_2$, and finally $P_4 + P_4 = 8P$.

Elliptic curves over finite fields

In order to apply the theory of elliptic curves to cryptography, we need to look at elliptic curves whose points have coordinates in a finite field $\mathbb{F}_p$. We simply define an elliptic curve over $\mathbb{F}_p$ to be an equation of the form

$$E : Y^3 = X^2 + AX + B \quad \text{with } A, B \in \mathbb{Z}_p \text{ and } 4A^3 + 27B^2 \neq 0$$

and

$$E(\mathbb{F}_p) = \{\langle x, y\rangle \in \mathbb{F}_p \mid y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

Example (3)

Consider the elliptic curve

$$E : Y^2 = X^3 + 3X + 8 \quad \text{over the field } \mathbb{F}_{13}.$$

We can find the points of $E(\mathbb{F}_{13})$ by substituting in all possible values $X = 0, 1, 2, \ldots, 12$ and checking for which X values the quantity $X^3 + 3X + 8$ is a square modulo 13. For example, putting $X = 0$ gives 8, and 8 is not a square modulo 13. Next we try $X = 1$, which gives $1 + 3 + 8 = 12$. It turns out that 12 is a square modulo 13; in fact, it has two square roots,

$$52 \equiv 12 \pmod{13} \text{ and } 82 \equiv 12 \pmod{13}.$$

This gives two points $\langle 1, 5 \rangle$ and $\langle 1, 8 \rangle$ in $E(\mathbb{F}_p)$.

Continuing in this fashion, we end up with a complete list,

$$E(\mathbb{F}_{13}) = \{\mathcal{O}, \langle 1,5\rangle, \langle 1,8\rangle, \langle 2,3\rangle, \langle 2,10\rangle, \langle 9,6\rangle, \langle 9,7\rangle, \langle 12,2\rangle, \langle 12,11\rangle\}.$$

Thus $E(\mathbb{F}_{13})$ consists of nine points.

	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$\mathcal{O}$	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$(1,5)$	$(1,5)$	$(2,10)$	$\mathcal{O}$	$(1,8)$	$(9,7)$	$(2,3)$	$(12,2)$	$(12,11)$	$(9,6)$
$(1,8)$	$(1,8)$	$\mathcal{O}$	$(2,3)$	$(9,6)$	$(1,5)$	$(12,11)$	$(2,10)$	$(9,7)$	$(12,2)$
$(2,3)$	$(2,3)$	$(1,8)$	$(9,6)$	$(12,11)$	$\mathcal{O}$	$(12,2)$	$(1,5)$	$(2,10)$	$(9,7)$
$(2,10)$	$(2,10)$	$(9,7)$	$(1,5)$	$\mathcal{O}$	$(12,2)$	$(1,8)$	$(12,11)$	$(9,6)$	$(2,3)$
$(9,6)$	$(9,6)$	$(2,3)$	$(12,11)$	$(12,2)$	$(1,8)$	$(9,7)$	$\mathcal{O}$	$(1,5)$	$(2,10)$
$(9,7)$	$(9,7)$	$(12,2)$	$(2,10)$	$(1,5)$	$(12,11)$	$\mathcal{O}$	$(9,6)$	$(2,3)$	$(1,8)$
$(12,2)$	$(12,2)$	$(12,11)$	$(9,7)$	$(2,10)$	$(9,6)$	$(1,5)$	$(2,3)$	$(1,8)$	$\mathcal{O}$
$(12,11)$	$(12,11)$	$(9,6)$	$(12,2)$	$(9,7)$	$(2,3)$	$(2,10)$	$(1,8)$	$\mathcal{O}$	$(1,5)$

How many points can we expect in a EC?

Theorem (Hasse)

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

$$|E(Fp)| = p + 1 - tp \quad \textit{with } t_p \textit{ satisfying } |t_p| \leq 2\sqrt{p}.$$

The *ECDLP* Problem

We defined before *DLP*: that of finding the number y given some base number g, where $x = g^y \pmod{p}$ for some large prime number p.

Cryptography with elliptic curves has a similar problem: the problem of finding the number k given a base point P where the point $Q = kP$. This is called the elliptic curve discrete logarithm problem, or *ECDLP*.

One important difference between *ECDLP* and the classical *DLP* is that *ECDLP* allows you to work with smaller numbers and still enjoy a similar level of security.

One way of solving *ECDLP* is to find a collision between two outputs, $c_1P + d_1Q$ and $c_2P + d_2Q$. The points P and Q in these equations are such that $Q = kP$ for some unknown k, and c_1, d_1, c_2, and d_2 are the numbers you will need in order to find k.
A collision occurs when two different inputs produce the same output. Therefore, in order to solve ECDLP, we need to find points where the following is true: $c_1P + d_1Q = c_2P + d_2Q$.
In order to find these points, we replace Q with the value kP, and we have the following:

$$c_1P + d_1kP = (c_1 + d_1k)P = (c_2 + d_2k)P = c_2P + d_2kP.$$

Thus $c_1 + d_1k = c_2 + d_2k$ modulo the number of the points in the curve, **which is not a secret**.

Thus

$$\begin{aligned} d_2 k - d_1 k &= c_1 - c_2 \\ k(d_2 - d1) &= c_1 - c_2 \\ k &= \frac{c_1 - c_2}{d_2 - d_1} \end{aligned}$$

In practice, elliptic curves extend over numbers of at least 256 bits, which makes attacking elliptic curve cryptography by finding a collision impractical because doing so takes up to 2^{128} operations (the cost of finding a collision over 256-bit numbers.

Diffie–Hellman Key Agreement over Elliptic Curves

The elliptic curve version of *DH* is identical to that of classical *DH* but with different notations. In the case of *ECC*, for some fixed point G, Alice picks a secret random number d_A, computes $P_A = d_A G$ (the point G multiplied by d_A), and sends P_A to Bob. Bob picks a secret random d_B, computes the point $P_B = d_B G$, and sends it to Alice. Then both compute the same shared secret, $d_A P_B = d_B P_A = d_A d_B G$. This method is called *elliptic curve Diffie–Hellman*, or *ECDH*.

Signing with Elliptic Curves

The standard algorithm used for signing with *ECC* is *ECDSA*, which stands for *elliptic curve digital signature algorithm*. This algorithm has replaced *RSA* signatures and classical *DSA* signatures in many applications. It is, for example, the only signature algorithm used in *Bitcoin* and is supported by many *TLS* and *SSH* implementations.

As with all signature schemes, *ECDSA* consists of a signature generation algorithm that the signer uses to create a signature using their private key and a verification algorithm that a verifier uses to check a signature's correctness given the signer's public key. The signer holds a number, d, as a private key, and verifiers hold the public key, $P = dG$. Both know in advance what elliptic curve to use, its order (n, the number of points in the curve), as well as the coordinates of a base point, G.

ECDSA Signature Generation

In order to sign a message, the signer first hashes the message with a cryptographic hash function such as *SHA-256* or *BLAKE2* to generate a hash value, h, that is interpreted as a number between 0 and $n - 1$. Next, the signer picks a random number, k, between 1 and n–1 and computes kG, a point with the coordinates $\langle x, y \rangle$. The signer now sets $r = x \pmod{n}$ and computes $s = (h + rd)/k \pmod{n}$, and then uses these values as the signature $\langle r, s \rangle$. The length of the signature will depend on the coordinate lengths being used. For example, when you're working with a curve where coordinates are 256-bit numbers, r and s would both be 256 bits long, yielding a 512-bit-long signature.

ECDSA Signature Verification

In order to verify an *ECDSA* signature $\langle r, s \rangle$ and a message's hash, h, the verifier first computes $w = 1/s$, the inverse of s in the signature, which is equal to $k/(h + rd) \pmod{n}$, since s is defined as $s = (h + rd)/k$. Next, the verifier multiplies w with h to find u according to the following formula:

$$wh = hk(h + rd) = u.$$

Then multiplies w with r to find v:

$$wr = rk(h + rd) = v.$$

Having u and v, the verifier computes the point Q:

$$Q = uG + vP$$

where P is the signer's public key, which is equal to dG, and the verifier only accepts the signature if the x coordinate of Q is equal to the value r from the signature.

This process works because, as a last step, we compute the point Q by substituting the public key P with its actual value dG:

$$uG + vdG = (u + vd)G.$$

When we replace u and v with their actual values, we obtain the following:

$$\begin{aligned} u + vd &= hk(h + rd) + drk/(h + rd) \\ &= (hk + drk)/(h + rd) \\ &= k(h + dr)/(h + rd) = k. \end{aligned}$$

This tells us that $(u + vd)$ is equal to the value k, chosen during signature generation, and that $uG + vdG$ is equal to the point kG.

ECDSA vs *RSA* Signatures

When comparing *RSA* and *ECC's* signature algorithms, recall that in *RSA* signatures, the signer uses their private key d to compute a signature as $y = xd \pmod{n}$, where x is the data to be signed and y is the signature. Verification uses the public key e to confirm that $y^e \pmod{n}$ equals x — a process that's clearly simpler than that of *ECDSA*.

RSA's verification process is often faster than *ECC's* signature generation because it uses a small public key e. But *ECC* has two major advantages over *RSA*:

- shorter signatures
- signing speed.

Encrypting with Elliptic Curves

Although elliptic curves are more commonly used for signing, you can still encrypt with them. But you'll rarely see people do so in practice due to restrictions in the size of the plaintext that can be encrypted: you can fit only about 100 bits of plaintext, as compared to almost 4000 in *RSA* with the same security level.

One simple way to encrypt with elliptic curves is to use the integrated encryption scheme (*IES*), a hybrid asymmetric–symmetric key encryption algorithm based on the Diffie–Hellman key exchange. Essentially, *IES* encrypts a message by generating a Diffie–Hellman key pair, combining the private key with the recipient's own public key, deriving a symmetric key from the shared secret obtained, and then using an authenticated cipher to encrypt the message.

When used with elliptic curves, *IES* relies on *ECDLP's* hardness and is called *elliptic-curve integrated encryption scheme* (*ECIES*). Given a recipient's public key, P, *ECIES* encrypts a message, M, as follows:

1. Pick a random number, d, and compute the point $Q = dG$, where the base point G is a fixed parameter. Here, $\langle d, Q \rangle$ acts as an ephemeral key pair, used only for encrypting M.
2. Compute an ECDH shared secret by computing $S = dP$.
3. Use a key derivation scheme (*KDF*) to derive a symmetric key, K, from S.
4. Encrypt M using K and a symmetric authenticated cipher, obtaining a ciphertext, C, and an authentication tag, T.

The *ECIES* ciphertext then consists of the ephemeral public key Q followed by C and T. Decryption is straightforward: the recipient computes S by multiplying R with their private exponent to obtain S, and then derives the key K and decrypts C and verifies T.

Choosing a Curve

When making your selection of the curve, be sure to choose coefficients a and b in the curve's equation $y^2 = x^3 + ax + b$ carefully; otherwise, you may end up with an insecure curve.

- The order of the group should not be a product of small numbers; otherwise solving *ECDLP* becomes much easier.
- Adding points $P + Q$ requires a specific addition formula when Q = P. Unfortunately, treating this case differently from the general one may leak critical information if an attacker is able to distinguish doublings from additions between distinct points. Some curves are secure because they use a single formula for all point addition. (When a curve does not require a specific formula for doublings, we say that it admits a unified addition law.)

- If the creators of a curve don't explain the origin of a and b, they may be suspected of foul play because you can't know whether they may have chosen weaker values that enable some yet-unknown attack on the cryptosystem.

How Things Can Go Wrong

Elliptic curves have their downsides due to their complexity and large attack surface. Their use of more parameters than classical Diffie–Hellman brings with it a greater attack surface with more opportunities for mistakes and abuse, and possible software bugs that might affect their implementation. Elliptic curve software may also be vulnerable to side-channel attacks due to the large numbers used in their arithmetic. If the speed of calculations depends on inputs, attackers may be able to obtain information about the formulas being used to encrypt.

ECDSA with Bad Randomness

ECDSA signing is randomised, as it involves a secret random number k when setting $s = (h + rd)/k \pmod{n}$. However, if the same k is reused to sign a second message, an attacker could combine the resulting two values, $s1 = (h1 + rd)/k$ and $s2 = (h2 + rd)/k$, to get $s1 - s2 = (h1 - h2)/k$ and then $k = (h1 - h2)/(s1 - s2)$. When k is known, the private key d is easily recovered by computing the following:

$$(ks1 - h1)/r = ((h1 + rd) - h1)/r = rd/r = d.$$

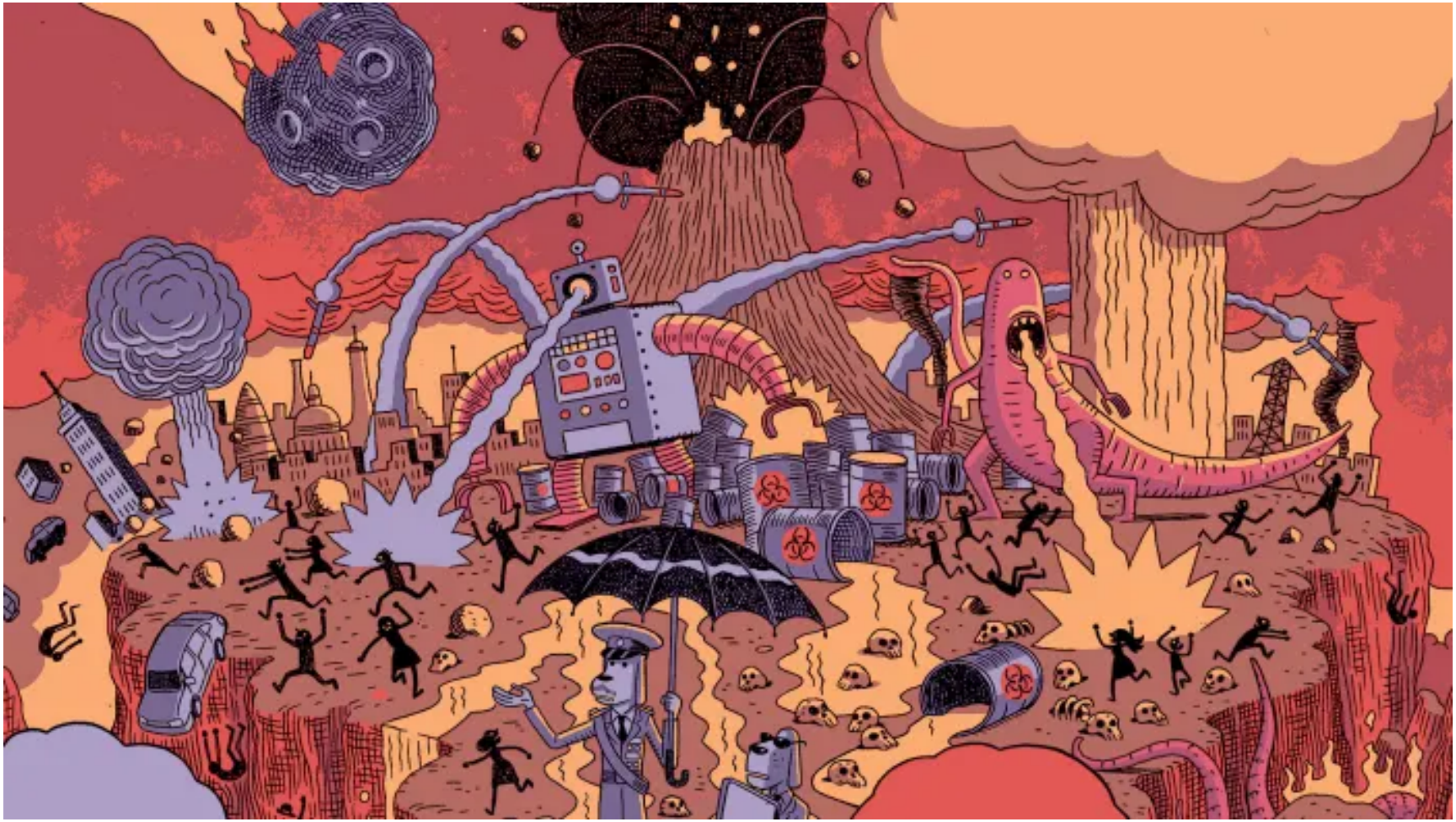
Breaking *ECDH* Using Another Curve

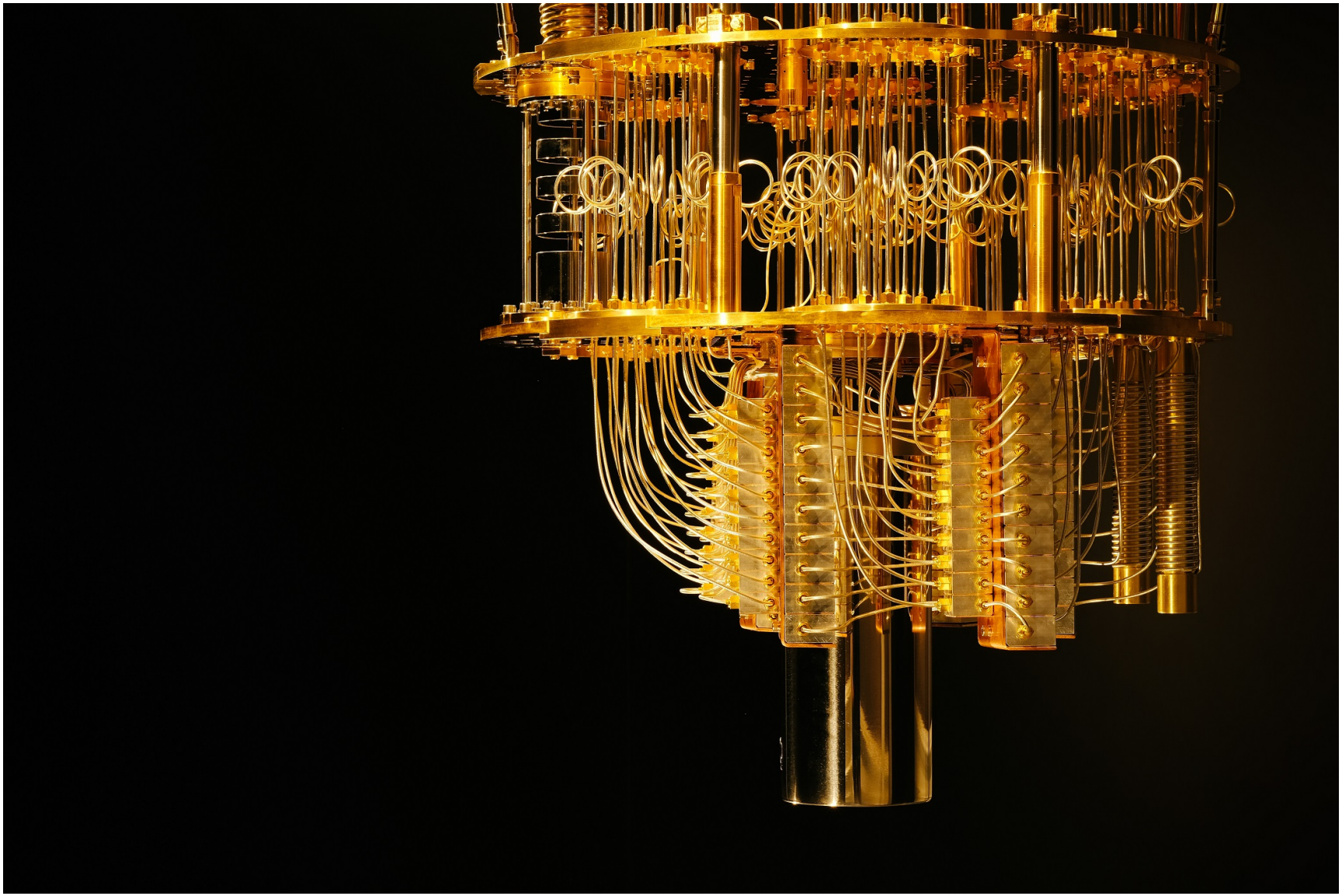
Say that Alice and Bob are running *ECDH* and have agreed on a curve and a base point, G. Bob sends his public key $d_B G$ to Alice. Alice, instead of sending a public key $d_A G$ on the agreed upon curve, sends a point on a different curve, either intentionally or accidentally. Unfortunately, this new curve is weak and allows Alice to choose a point P for which solving *ECDLP* is easy. She chooses a point of low order, for which there is a relatively small k such that $kP = O$.

Now Bob, believing that he has a legitimate public key, computes what he thinks is the shared secret $d_B P$, hashes it, and uses the resulting key to encrypt data sent to Alice. The problem is that when Bob computes $d_B P$, he is unknowingly computing on the weaker curve. As a result, because P was chosen to belong to a small subgroup within the larger group of points, the result $d_B P$ will also belong to that small subgroup, allowing an attacker to determine the shared secret $d_B P$ efficiently if they know the order of P.

One way to prevent this is to make sure that points P and Q belong to the right curve by ensuring that their coordinates satisfy the curve's equation. Doing so would prevent this attack by making sure that you're only able to work on the secure curve.

...and now for something completely different!





Discrete Lattices Cryptography

Definition (Discrete Lattice)

Let $v_1, \ldots, v_n \in \mathbb{R}^m$ be a set of linearly independent vectors. The lattice L generated by $v_1, \ldots, v_n$ is the set of linear combinations of $v_1, \ldots, v_n$ with coefficients in $\mathbb{Z}$,

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

Proposition

Any two bases for a lattice L are related by a matrix having integer coefficients and determinant equal to ± 1.

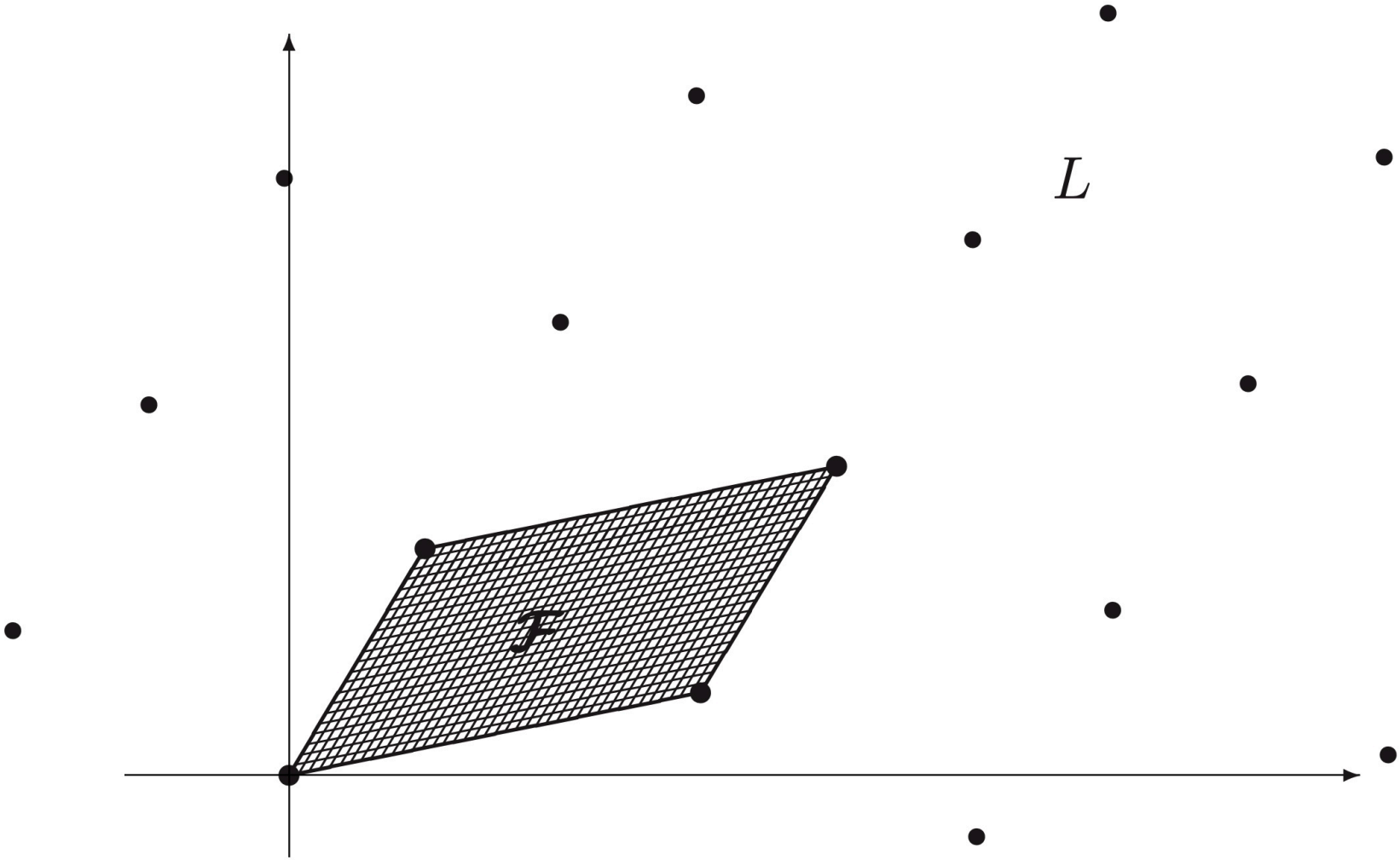
Definition (Integral Lattice)

An integral (or integer) lattice is a lattice all of whose vectors have integer coordinates. Equivalently, an integral lattice is an additive subgroup of $\mathbb{Z}^m$ for some $m \geq 1$.

Definition (Fundamental Domain)

Let L be a lattice of dimension n and let $v_1, v_2, \ldots, v_n$ be a basis for L. The *fundamental domain* (or fundamental parallelepiped) for L corresponding to this basis is the set

$$\mathcal{F}(v_1, \ldots, v_n) = \{t_1 v_1 + t_2 v_2 + \cdots + t_n v_n \mid 0 \leq t_i < 1\}$$


L
F

Proposition (Hadamard's Inequality)

Let L be a lattice, take any basis $v_1, \ldots, v_n$ for L, and let F be a fundamental domain for L. Then

$$det\, L = Vol(F) \leq ||v_1||\, ||v_2|| \cdots ||v_n||.$$

The shortest vector problem and the closest vector problem

The Shortest Vector Problem (*SVP*): Find a nonzero vector $v \in L$ that minimizes the Euclidean norm $||v||$.

The Closest Vector Problem (*CVP*): Given a vector $w \in \mathbb{R}^m$ that is not in L, find a vector $v \in L$ that minimizes $||w - v||$.

Approximate Shortest Vector Problem (*apprSVP*): Let $\psi(n)$ be a function of n. In a lattice L (dimension n), find a nonzero vector that is no more than $\psi(n)$ times longer than a shortest nonzero vector. In other words, if v_s is a shortest nonzero vector in L, find a nonzero vector $v \in L$ satisfying

$$||v|| \leq \psi(n)||v_s||.$$

Approximate Closest Vector Problem (*apprCVP*): This is the same as *apprSVP*, but now we are looking for a vector that is an approximate solution to *CVP*, instead of an

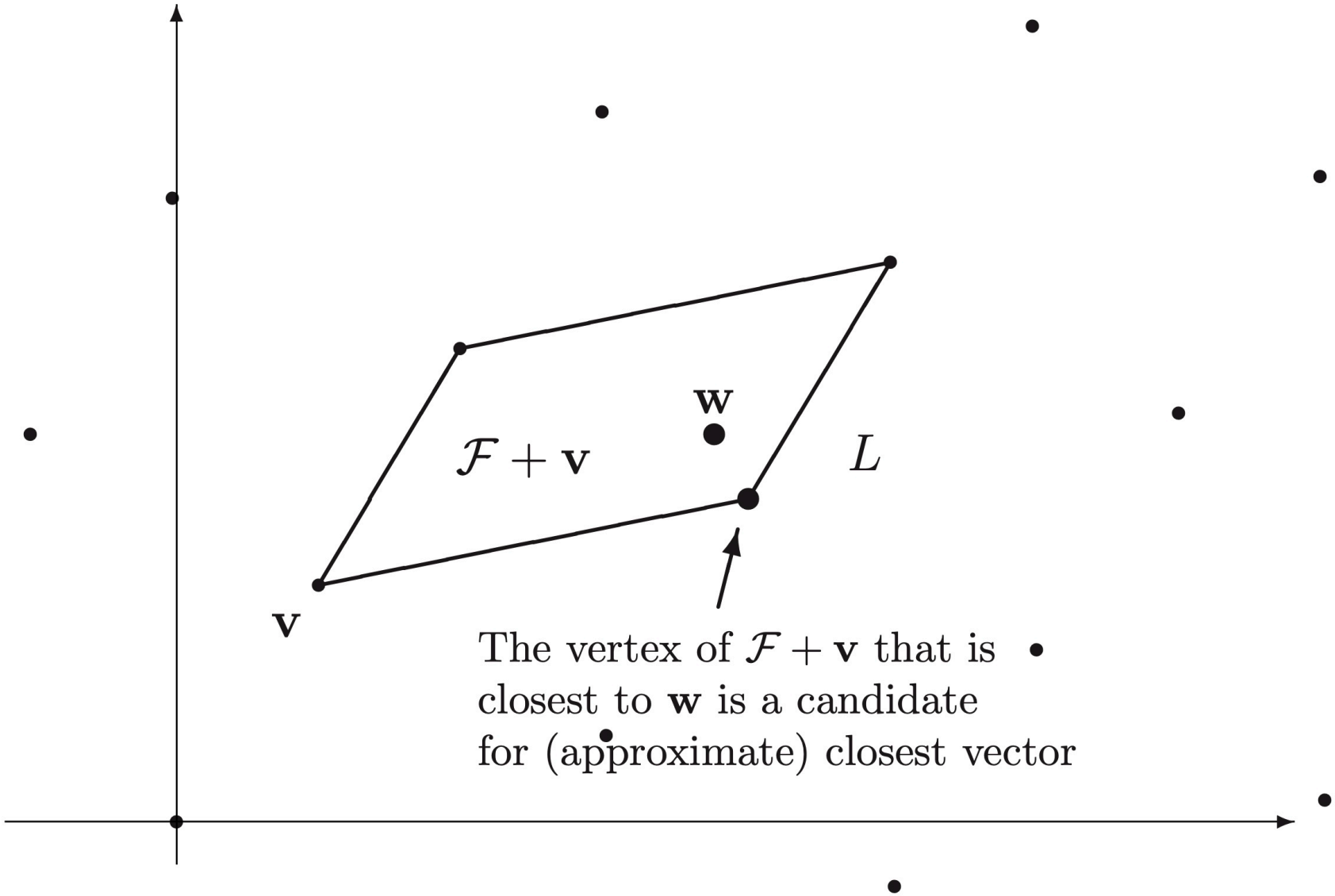
Babai's algorithm to solve *apprCVP*

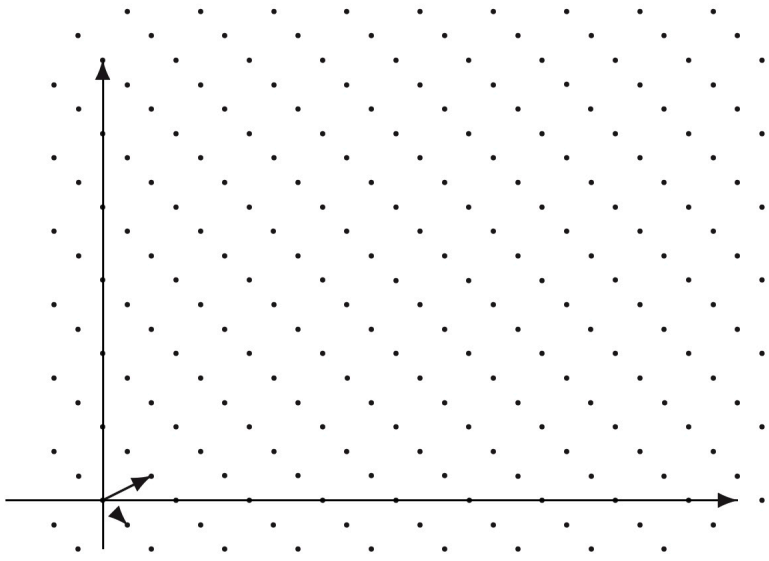
Theorem (Babai's Closest Vertex Algorithm)

Let $L \subset \mathbb{R}^n$ be a lattice with basis $v_1, \ldots, v_n$, and let $w \in \mathbb{R}^n$ be an arbitrary vector. If the vectors in the basis are sufficiently orthogonal to one another, then the following algorithm solves CVP.

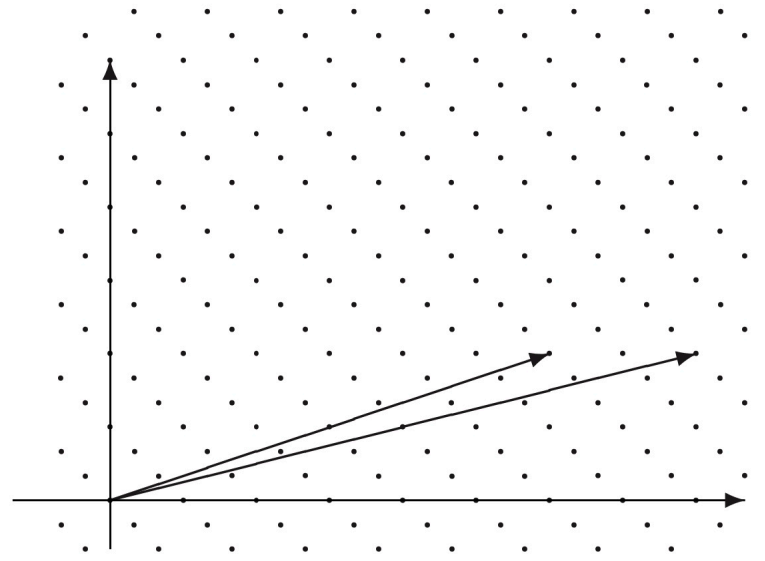
1. *Write $w = t_1 v_1 + t_2 v_2 + \cdots + t_n v_n$ with $t_1, \ldots, t_n \in \mathbb{R}$.*
2. *Set $a_i = \lfloor t_i \rceil$ for $i = 1, 2, ..., n$.*
3. *Return the vector $v = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n$.*

In general, if the vectors in the basis are reasonably orthogonal to one another, then the algorithm solves some version of apprCVP, but if the basis vectors are highly nonorthogonal, then the vector returned by the algorithm is generally far from the closest lattice vector to w.


w
$\mathcal{F} + \mathbf{v}$
L
v
The vertex of $\mathcal{F} + \mathbf{v}$ that is
closest to $\mathbf{w}$ is a candidate
for (approximate) closest vector



A "Good Basis"



A "Bad Basis"

Example (4)

Let $L \subset \mathbb{R}^2$ be the lattice given by the basis

$$v_1 = (137, 312) \text{ and } v_2 = (215, -187).$$

We are going to use Babai's algorithm to find a vector in L that is close to the vector

$$w = (53172, 81743).$$

The first step is to express w as a linear combination of v_1 and v_2 using real coordinates. We do this using linear algebra. Thus we need to find $t_1, t_2 \in \mathbb{R}$ such that

$$w = t_1 v_1 + t_2 v_2.$$

This gives the two linear equations

$$53172 = 137t_1 + 215t_2 \text{ and } 81743 = 312t_1 - 187t_2,$$

or,

$$(53172, 81743) = (t_1, t_2)\begin{pmatrix} 137 & 312 \\ 215 & -187 \end{pmatrix}.$$

We find that $t_1 \approx 296.85$ and $t_2 \approx 58.15$. Babai's algorithm tells us to round t_1 and t_2 to the nearest integer and then compute

$$v = \lfloor t_1 \rceil v_1 + \lfloor t_2 \rceil v_2 = 297(137, 312) + 58(215, -187) = (53159, 81818).$$

Then v is in L and v should be close to w. This is to be expected, since the vectors in the given basis are fairly orthogonal to one another, as is seen by the fact that the Hadamard ratio

$$\mathcal{H}(v_1, v_2) = \left(\frac{\det L}{||v_1||.||v_2||}\right)^{\frac{1}{2}} \approx \left(\frac{92699}{340.75 \times 284.95}\right)^{\frac{1}{2}} \approx 0.977$$

is reasonable close to 1.

We now try to solve the same closest vector problem in the same lattice, but using the new basis

$v_1' = (1975, 438) = 5v_1 + 6v_2$ and $v_2' = (7548, 1627) = 19v_1 + 23v_2$.

The system of linear equations

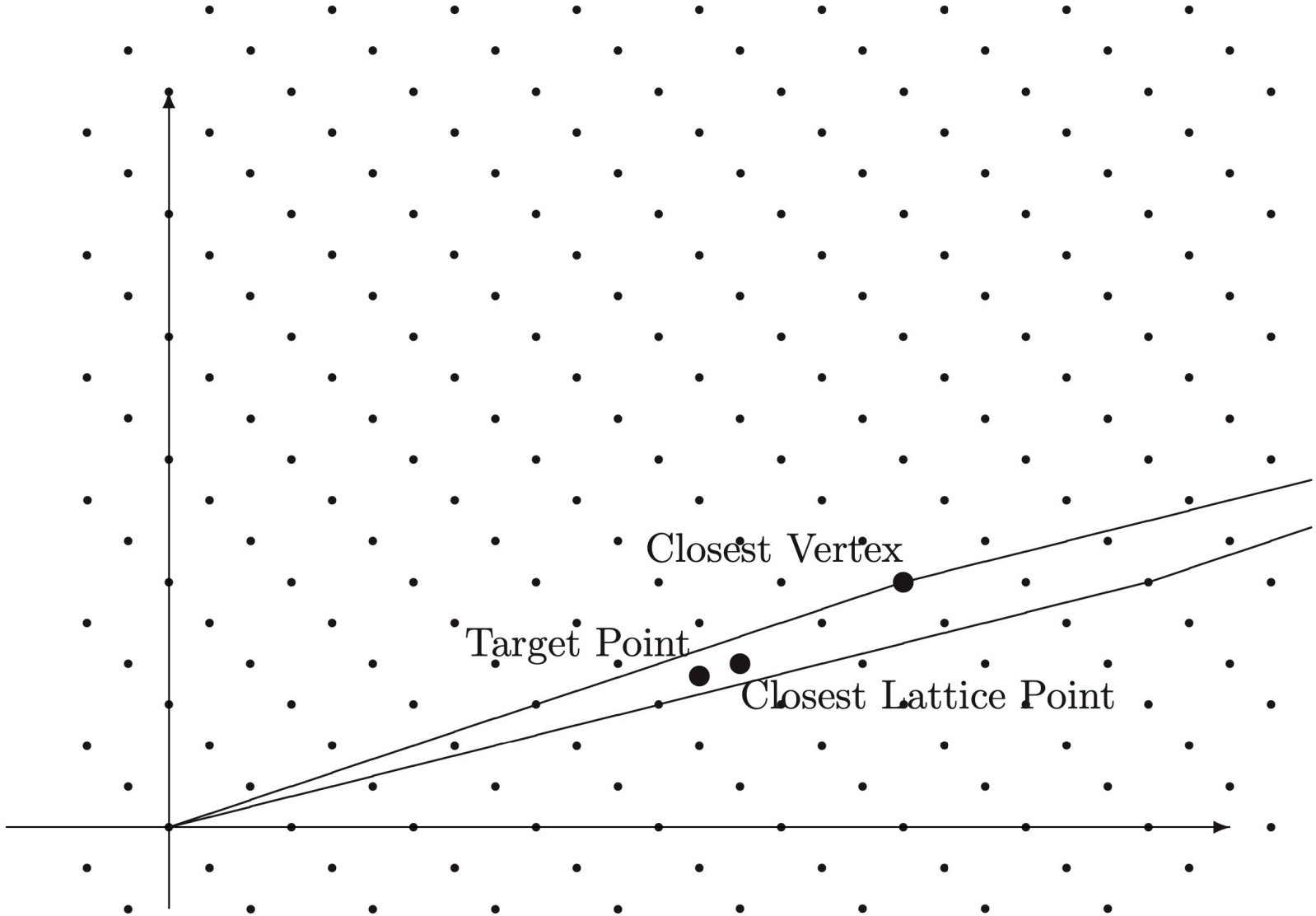
$$(53172, 81743) = (t_1, t_2) \begin{pmatrix} 1975 & 438 \\ 7548 & 1627 \end{pmatrix}$$

has the solution $(t_1, t_2) \approx (5722.66, -1490.34)$, so we set

$$v' = 5723v_1' - 1490v_2' = (56405, 82444).$$

Then $v' \in L$, but v' is not particularly close to w, since

$$||v' - w|| \approx 3308.12.$$


Closest Vertex
Target Point
Closest Lattice Point

PKC using Discrete Lattices

1. Choose a near orthogonal set of linear independent vectors, in $\mathbb{Z}^n$, that will be vectorial base B of the Lattice L.
2. Choose a matrix M that transforms B in a quite "skewed" vectorial base $B' = MB$. This new base will be Alice's **public key**.
3. Alice's **private key** will be M.
4. If Bob wants to share some secret s with Alice, interprets s as coordinates in base B', thus getting a point x in the lattice.
5. Randomly generates some (small) noise Δx and obtains a point $x' = x + \Delta x$ living in $\mathbb{R}^n$, that sends to Alice.
6. Alice can use M to write express x' in B coordinates and apply Babbai's algorithm to recover x.


DID ANYONE WATER THE TREE ?
I DID
I DID
I DID
I DID
I DID
DOG CHRISTMASES
Crazy-Jokes.com