Applications of elliptic curves in cryptography and computational number theory

Kamal Khuri-Makdisi

Applications of elliptic curves in cryptography and computational number theory

Kamal Khuri-Makdisi
Center for Advanced Mathematical Sciences
American University of Beirut
kmakdisi@aub.edu.lb

Lectures given during CIMPA summer school
Algebraic geometry and arithmetic of curves
Beirut, Lebanon, July 2004

– Typeset by FoilTEX –

Introduction: what is cryptography?

Given: a (very large) finite set of messages $\mathcal{M}$, or more generally a finite set of **plaintext** messages $\mathcal{M}_P$ and a finite set of **ciphertext** messages $\mathcal{M}_C$. Most often we'll take $\mathcal{M} = \mathcal{M}_P = \mathcal{M}_C$.

Question: Find "practical" **encoding** and **decoding** maps $E : \mathcal{M}_P \to \mathcal{M}_C$ and $D : \mathcal{M}_C \to \mathcal{M}_P$ such that $D \circ E = \mathrm{id}$.

Then two people, traditionally called Alice and Bob, can communicate in privacy even if their messages are being intercepted (e.g., over the internet!). To send a message $m \in \mathcal{M}_P$, Alice sends $m' = E(m) \in \mathcal{M}_C$, and Bob decodes m' by applying D to it. This assumes that Alice and Bob have decided on D and E, which they usually keep private between themselves. The cryptographer (Cathy) only sees the message m', but in principle does not know how to apply D. A reasonable model is to assume that Alice and Bob have sent many messages m, encoded and decoded using E, D, and that Cathy has seen all the corresponding coded forms m'. In addition, Cathy may have found out by other

means **some** of the original plaintext messages m, in addition to knowing the corresponding ciphertexts.

What can one ask for in a cryptographic system (abbreviated **cryptosystem**) to allow practical and secure communication?

Basic Requirements:

- E must be injective
- A computer must be able to carry out both D and E in reaonable time
- Cathy should not be able to figure out how to do E, or even to guess part of m from a knowledge of m'. Typically, Cathy will have intercepted many different m', and will know a number of earlier sent pairs (m, m'). But it must remain computationally **infeasible** for Cathy to determine m from a newly sent m'.

Remark: different levels of security require different levels of infeasibility: e.g., if a company is discussing a merger that has to be kept secret for the next two

weeks only, then it's okay if Cathy can figure out how to decrypt messages using six months of work on powerful computers.

Example: Julius Caesar's cipher: $\mathcal{M} = \mathbf{Z}/26\mathbf{Z}$, with $\overline{0}$ ='A', $\overline{1}$ ='B', etc.; then define $E(m) = m + \overline{3}$, and $D(m') = m' - \overline{3}$. Thus

E('A') = 'D'	D('A') = 'X'
E('B') = 'E'	D('B') = 'Y'
E('C') = 'F'	D('C') = 'Z'
E('D') = 'G'	D('D') = 'A'
...	...

And we can encode messages letter-by-letter, so E('VENIVIDIVICI') = 'YHQLYLGLYLFL'. This code is of course quite easy to break using frequency analysis, even if we use more complicated bijections $D, E : \mathcal{M} \to \mathcal{M}$. This is one reason why $\mathcal{M}$ must be a large set.

Problem: If $M = |\mathcal{M}|$ is very large, then it is infeasible to allow D (and its inverse E) to be an arbitrary permutation. E.g., if $M = 10^{100}$, then there

are $M! \sim 10^{10^{102}}$ possible permutations. No practical notation or algorithm can describe all elements of the symmetric group S_M for M this large.

Solution: Restrict to a small class of E and D, indexed by a **key** $k \in \mathcal{K}$, where the set of keys $\mathcal{K}$ is still large, but elements of $\mathcal{K}$ can be described succinctly. Example: if $|\mathcal{K}| = 10^{100}$ then a key can be described using 100 digits or about 300 bits. It is possible to use different keys for encryption than for decryption. So, say we have encryption keys $\mathcal{K}$, and decryption keys $\hat{\mathcal{K}}$, which describe distinct encryption/decryption maps ($k \in \mathcal{K}$, $\hat{K} \in \hat{\mathcal{K}}$):

$$E_k : \mathcal{M} \to \mathcal{M}, \qquad D_{\hat{K}} : \mathcal{M} \to \mathcal{M}.$$

We assume we also have a good way to generate pairs $(k, \hat{K})$ for which $D_{\hat{K}} = E_k^{-1}$. Such a pair is called an **encryption/decryption pair**.

Private key cryptography

Here $\mathcal{K} = \hat{\mathcal{K}}$, and $k = \hat{K}$. Thus D_k is the inverse of E_k. Alice and Bob agree on a particular key k, which they keep secret between themselves. They then use E_k and D_k to encrypt and decrypt messages.

Basic cryptographic premise: Cathy knows how the system works but doesn't know the key. So she knows the map $k \mapsto E_k, D_k$, and typically has found out several pairs (m, m') where $m' = E_k(m)$ but should have a hard time determining k. Hence E_k and D_k should depend on k in a fairly complicated way.

Example: Let $\mathcal{M} = \mathbf{Z}/M\mathbf{Z}$, $\mathcal{K} = (\mathbf{Z}/M\mathbf{Z})^* \times \mathbf{Z}/M\mathbf{Z}$. For $k = (a, b)$, with a invertible modulo M and b any integer modulo m, define $E_{(a,b)}(m) = am + b$, and $D_{(a,b)}(m') = a^{-1}(m - b)$. We have thus restricted our permutations to the small set of affine transformations of $\mathbf{Z}/M\mathbf{Z}$. Here $|\mathcal{K}| \sim M^2 \ll M!$. This system is not secure, since knowledge of just two pairs of the form (m, m') usually allows one to find the key $k = (a, b)$.

Advantage of private key cryptosystem: Encryption and decryption of long messages can be done very quickly with modern private key cryptosystems.

Disadvantages of private keys:

- If n people need to communicate privately, then need $n(n-1)/2$ keys, one for each pair of people. Difficult to manage security with so many keys floating around the different computers.
- Each given pair of people (Alice and Bob) must somehow agree on a choice of k. They cannot send the key over the internet (Cathy is watching). They need to meet in person or have some kind of secure communications channel — impractical.

Public key cryptography

Main cryptographic application of number theory and algebraic curves, particularly elliptic curves at the moment.

Setup: Each user U (e.g., Alice or Bob) generates an encryption/decryption pair $(k_U, \hat{K}_U) \in \mathcal{K} \times \hat{\mathcal{K}}$, and publishes k_U in a directory while keeping $\hat{K}_U$ secret. Then if Bob wants to send a message m to Alice, he sends her $m' = E_{k_A}(m)$ which she can decrypt using her private key. If he wants to sign the message, he can send $E_{k_A}(D_{\hat{K}_B}(m))$, which Alice decrypts using $D_{\hat{K}_A}$ to get $n = D_{\hat{K}_B}(m)$. She then applies E_{k_B} to recover m. The point is that only Bob can produce an n which satisfies $E_{k_B}(n) = m$.

Important: It should be computationally infeasible to find $\hat{K}_U$ (more precisely, to find the map $D_{\hat{K}_U}$) from a knowledge of k_U. It is a bit surprising that this can be done at all (modulo some plausible but unproven conjectures).

Advantages of public key systems: Far fewer keys have to circulate; n users need only n keys. Also, no secret keys ever need to be exchanged over the internet.

Disadvantages:

- Existing public key cryptosystems are much slower than private key cryptosystems. One standard solution is to use a public key cryptosystem just to agree on a private key for future communication that same day.
- No formal proofs have been found to guarantee the difficulty of breaking public key cryptosystems (or of any private ones, for that matter!). One can make heuristic arguments why systems are secure, if one is willing to believe that certain problems (like factoring large integers) are computationally hard.
- In practice, the main source of insecurity in cryptosystems seems to be the human side! A poor choice of key, or a faulty implementation, or an easily bribed insider, are much easier (and apparently more common) ways for cryptographic systems to be broken.

Example: the RSA public key cryptosystem

Remark: RSA (Rivest-Shamir-Adleman) does not involve algebraic curves, but we discuss it because:

- It introduces the topic of efficient computations in number theory.
- It gives practical motivation for the mathematical question of either factoring a very large number (say around 10^{300}), or proving it to be prime. This question is of course interesting for its own sake.
- We will later see how elliptic curves can help answer the above question in practice.

Implementing RSA: First one sets up a private/public key pair. Each user U (somehow?) finds two very large prime numbers $p = p_U$ and $q = q_U$, both of the order of 10^{150}, say. The basic premise is that **factoring is hard** — so if someone knows the product $N = pq \sim 10^{300}$ but does not know p and q, then that person cannot factor N

in reasonable time. Now user U chooses an integer $e = e_U$ that is relatively prime to $p-1$ and $q-1$. This is typically done by choosing e to be a large prime number.

The public key: is the pair $k = k_U = (N, e) = (N_U, e_U)$.

The private key: is the pair $\hat{K} = \hat{K}_U = d = d_U \in \mathbf{Z}$, chosen such that $de \equiv 1 \bmod (p-1)(q-1)$. User U knows p and q, and therefore can find d by using the extended Euclidean algorithm on e and $(p-1)(q-1)$. This yields $d, \ell \in \mathbf{Z}$ such that $de + \ell(p-1)(q-1) = 1$.

The message space: We theoretically use $\mathcal{M} = (\mathbf{Z}/N\mathbf{Z})^*$, the group of invertible integers modulo N. But in practice we do calculations in the ring $\mathbf{Z}/N\mathbf{Z}$, since the probability of stumbling across an element $x \in \mathbf{Z}/N\mathbf{Z}$ that is divisible by p or by q is less than $1/p + 1/q$ which is miniscule.

Encoding and decoding:

- $E_k(m) \equiv m^e \bmod N$,
- $D_{\hat{K}}(m') \equiv (m')^d \bmod N$.

Reason this works: The group $(\mathbf{Z}/N\mathbf{Z})^*$ has order $\varphi(N) = (p-1)(q-1)$, so if m is an integer relatively prime to N, then $m^{(p-1)(q-1)} \equiv 1 \bmod N$. Thus $(m')^d \equiv m^{de} \equiv m^1 = m \bmod N$. Recall that the chance of stumbling across an m that is NOT relatively prime to N is so small as to be negligible. If that happens, though, we can factor N and break the cryptosystem.

Questions: many of the operations listed above may seem difficult to do if the numbers are all large. We deal eith them as follows:

- How can we find large primes p and q? The prime number theorem says essentially that a large number p is prime with "probability" $1/\log p$. Thus since $p \sim 10^{150}$, the probability is $1/150 \log 10 \approx 1/350$. So somehow we must test around 350 large numbers on average to find one p, and we then repeat the process to find q. We will discuss how to test primality later.
- Doing arithmetic with large numbers (up to $\delta = 300$ digits in our example) is not too bad for a computer — either adapt the pencil and

paper algorithms from elementary school, which works well for addition and subtraction but takes time $O(\delta^2)$ for multiplication, OR use fancier techniques which do multiplication and division in time $O(\delta^{1+\epsilon})$.

- What about the impossibly large powers m^e, $(m')^d$?
  - We only care about $m^e \bmod N$. So of course we NEVER calculate the integer m^e, since $m^e \sim (10^{300})^{10^{300}}$ which has some 10^{302} digits!
  - So do every calculation modulo N, and reduce each intermediate result modulo N. So we never see numbers bigger than $N^2 \sim 10^{600}$. Much better.
  - Wait — we cannot hope to get the eth power by multiplying $(m \bmod N)(m \bmod N)\ldots(m \bmod N)$. That would take e factors which is too many since $e \sim 10^{300}$. Instead we use **fast exponentiation**.

Fast exponentiation

- Works for any group G where we can implement the group operation. We then find m^e for $m \in G$, even if e is large.
- Basic idea: write e in binary notation, and use repeated squaring to get powers of the form $m, m^2, m^4, m^8, \ldots$ which get combined according to the binary digits of e.
- Example: $G = (\mathbf{Z}/1000\mathbf{Z})^*$, $m = \overline{3}$, and $e = 147$. In other words, **find** $\overline{3}^{147}$.
  - The binary expansion is $147 = 10010011_2 = 128 + 16 + 2 + 1$.
  - By repeated squaring, we find $\overline{3}^2 = \overline{9}$, $\overline{3}^4 = \overline{81}$, $\overline{3}^8 = \overline{561}$, $\overline{3}^{16} = \overline{721}$, . . . , $\overline{3}^{128} = 961$.
  - Put these together: $\overline{3}^{147} = \overline{3}^{128} \cdot \overline{3}^{16} \cdot \overline{3}^2 \cdot \overline{3}^1 = \overline{961} \cdot \overline{721} \cdot \overline{9} \cdot \overline{3} = \overline{787}$.

First remarks on testing primality

If p is large, say with hundreds of digits, it is not practical to try to divide p by all potential divisors (less than $\sqrt{p}$). Of course we do check some small divisors anyhow, so that p is not divisible by 2 or 3, or by any number up to, e.g., a million.

In practice: Check first that p is **probably prime**. Very roughly, we check that Fermat's little theorem $a^{p-1} \equiv 1 \bmod p$ holds for many random a (where we had better have $(a, p) = 1$). This is done **using fast exponentiation**. If we ever get $a^{p-1} \not\equiv 1$, we know that p cannot be prime. But there is a chance that many a's will falsely pass this test.

More precisely: one uses the Miller-Rabin test, which also tests that $a^{(p-1)/2} \equiv \pm 1 \bmod p$, and further similar identities with powers of $a \bmod p$. If p is composite, then fewer than $1/4$ of the choices of a can pass the Miller-Rabin test. So if p passes the Miller-Rabin test for many randomly chosen a's, we are willing to bet that p is prime.

Proving primality: This takes more work, and is the topic of a later lecture. One can use the recent

breakthrough (2002) of Agarwal, Kayal, and Saxena, a deterministic polynomial-time algorithm (in the number of digits of p). For the sizes of numbers we consider, there are other methods that are faster in practice. We will see a probabilistic test using elliptic curves later in these lectures.

The discrete logarithm problem

Premise: Public key cryptosystems can be built from large finite abelian groups, based on the belief that the Discrete Logarithm Problem (DLP) in a well-chosen group G is intractable.

The DLP: Let $g, h \in G$. Then either find $a \in \mathbf{Z}$ such that $h = g^a$, or show that no such a exists. We write $a = \log_g h$. The value of a actually belongs to $\mathbf{Z}/N\mathbf{Z}$, where N is the order of the element $g \in G$.

Remark: In many cases we know that G is cyclic, and we choose g to be a generator. Thus $N = |G|$. In this case a exists, but finding it seems difficult in general if N is very large. If G is not cyclic, we often just work in the cyclic subgroup generated by g.

Careful: One has to avoid the case where N is a product of small primes, since then there exists an algorithm due to Silver-Pohlig-Hellman to find the discrete logarithm quickly. Otherwise, for "black box" groups, the best general methods known take time $O(\sqrt{N})$ (probabilistic) or $O(\sqrt{N}\log N)$ (deterministic). In a moment we will review one such method, called "Baby-step-giant-step."

Examples of groups used in cryptography:

- Let p be a prime. Then $(\mathbf{Z}/p\mathbf{Z})^*$ is a cyclic group of order $p-1$. We can look for a generator g, called a **primitive root** modulo p (finding g is difficult, unless we somehow manage to factor $p-1$, which we hope has some large prime factors to avoid Silver-Pohlig-Hellman). The discrete logarithm in this group is still believed to be hard, but ingenious algorithms exist that are much better than those for "black box" groups.

- If q is a power of a prime, let $\mathbf{F}_q$ be the finite field with q elements. (Example: $\mathbf{F}_p = \mathbf{Z}/p\mathbf{Z}$ if p is prime.) Given an elliptic curve E defined over $\mathbf{F}_q$, the $\mathbf{F}_q$-rational points $E(\mathbf{F}_q)$ form a finite group with $q+1-2\sqrt{q} \leq |E(\mathbf{F}_q)| \leq q+1+2\sqrt{q}$ **(Hasse's theorem)**. This is usually not cyclic, but we can often work in a large cyclic subgroup of $E(\mathbf{F}_q)$. No algorithm is currently known for the DLP on an elliptic curve that improves on what is known for a "black box" group.

- Fancier: let C be an algebraic curve of genus g over $\mathbf{F}_q$. Then the Jacobian $J(C)$ is a

g-dimensional algebraic group (see Oesterlé's lectures), and its group of $\mathbf{F}_q$-rational points generalizes the above example. If q is fixed and g grows, the DLP for these groups becomes easier, while the group law becomes more difficult to implement. Cryptographic applications seem limited to rather small g, say $g \leq 6$.

- A **non-example**: let G be the **additive** group $\mathbf{Z}/N\mathbf{Z}$. Then the discrete logarithm problem becomes the equation $ag \equiv h \bmod N$, which is trivial to solve for a (or to find that no solution exists).

Toy example of DLP: Let $G = (\mathbf{Z}/101\mathbf{Z})^*$, so $|G| = 100 = 2^2 5^2$. One can check that $g = \overline{2}$ is a generator, since $g^{100/2} = \overline{2}^{50} = \overline{100}$, and $g^{100/5} = \overline{95}$, so the order of g is a factor of $|G| = 100$ but not a factor of 20 or of 50.

- $\log_{\overline{2}} \overline{8} = 3$ (or 103, or 203, or -97, etc.).

- $\log_{\overline{2}} \overline{3} =$? Tricky way: notice from the above that $\overline{2}^{50} = -\overline{1}$, and $\overline{2}^{20} = -\overline{6}$. Then $\overline{2} \cdot \overline{3} = \overline{2}^{50+20}$, and the discrete logarithm is 69.

- The above trick generalizes to "index calculus methods" in any $(\mathbf{Z}/p\mathbf{Z})^*$. Idea: look for powers $g^b h^c \bmod p$ whose residues modulo p factor easily, and combine identities to find the discrete logarithm. These are much better than the black box DLP algorithms such as Baby-step-giant-step described below.

A trickier example of DLP: Let $p = 172316432754274362361$, and consider the elliptic curve E over $\mathbf{F}_p$, where $E : y^2 = x^3 + \overline{3141}x + \overline{5926}$.

- $|E(\mathbf{F}_p)| = 172316432762555079388 = 2^2 \cdot 13 \cdot 140534491 \cdot 23579816809$.

- The "randomly chosen" point $P = (\overline{2718}, \overline{7303544926054677840})$ generates the group.

- Question: What is the discrete logarithm $\log_P Q$, where $Q = (\overline{271828}, \overline{5326516977756444 2543})$? This can be found in a small multiple of $\sqrt{23579816809} \approx 150000$ steps using a combination of Silver-Pohlig-Hellman and Baby-

step-giant-step, so it isn't unreasonable. The answer turns out to be 134712877515817113540.

Black box ways to do DLP

We illustrate Baby-step-giant-step (BSGS) and Silver-Pohlig-Hellman (SPH) on the example of $\log_{\overline{2}} \overline{3}$ in $(\mathbf{Z}/101\mathbf{Z})^*$. Our goal: find a with $\overline{2}^a = \overline{3}$. We want to avoid trying out each of $a = 0, 1, \ldots, 99$ until we reach a solution. This will take expected time $O(N)$ where $N = 100$.

BSGS: Write $a = 10i + j$, where $0 \leq i, j < 10$. (For general N, we replace 10 by the first integer $> \sqrt{N}$.) Then we want to solve $\overline{2}^{10i} = \overline{3} * \overline{2}^{-j}$. We calculate the 20 numbers corresponding to each choice of i and j:

- Giant steps: $\overline{2}^0 = \overline{1}, \overline{2}^{10} = \overline{14}, \ldots, \overline{2}^{60} = \overline{87}, \ldots$
- Baby steps: $\overline{3} * \overline{2}^0 = \overline{3}, \overline{3} * \overline{2}^{-1} = \overline{52}, \ldots, \overline{3} * \overline{2}^{-8} = \overline{73}, \overline{3} * \overline{2}^{-9} = \overline{87}$.

These two lists have size $10 = O(\sqrt{N})$. We find the common element ($\overline{87}$) and conclude that $\overline{2}^{60} = \overline{3} * \overline{2}^{-9}$, hence that $a = 69$.

SPH: We factor $N = 100 = 2^2 5^2$. Our strategy:

- Find $a \bmod 2$, and use this to find $a \bmod 4$.
- Find $a \bmod 5$, and use this to find $a \bmod 25$.
- By the Chinese Remainder Theorem, combine $a \bmod 4$ and $a \bmod 25$ to obtain $a \bmod 100$.

The method is to write $a = 4b_2 + 2b_1 + b_0 = 25c_2 + 5c_1 + c_0$, where $b_0, b_1 \in \{0, 1\}$ and $c_0, c_1 \in \{0, 1, 2, 3, 4\}$. We then find the unknown b_0, b_1, c_0, c_1 by a clever exhaustive search (can also use BSGS if N has large prime factors). We illustrate for c_0 and c_1:

- Precomputation: $(\overline{2}^0, \overline{2}^{20}, \overline{2}^{40}, \overline{2}^{60}, \overline{2}^{80}) = (\overline{1}, \overline{95}, \overline{36}, \overline{87}, \overline{84})$.
- Observe that if $\overline{3} = \overline{2}^{5k+c_0}$, then $\overline{3}^{20} = \overline{2}^{20c_0}$. So we can find $c_0 \bmod 5$ by consulting the above table. Since $\overline{3}^{20} = \overline{84}$, we get $20c_0 = 80$ and $c_0 = 4$.
- Now take $\overline{2}^{25c_2+5c_1} = \overline{3} \cdot \overline{2}^{-c_0} = \overline{57}$ since we know $c_0 = 4$. By a similar argument we have

$\overline{2}^{20c_1} = \overline{57}^4 = \overline{87}$. Thus $c_1 = 3$ and $5c_1 + c_0 = 19$. Thus $a \equiv 19 \bmod 25$.

- End of the argument: a similar calculation gives $b_0 = 1, b_1 = 0$ so $a \equiv 1 \bmod 4$. Combining, we get $a \equiv 69 \bmod 100 = |G|$, so we can take $a = 69$.

Cryptography based on DLP

Assume: G is a cyclic group with generator g. The order $|G|$ should be large, and not divisible by too many small primes, in order that the DLP in G be considered intractable. Remark: this is a necessary but perhaps not sufficient condition for the following cryptosystems to be secure!

Diffie-Hellman key exchange: Alice and Bob need to agree on which key to use in their private-key cryptosystem, but can only communicate across an open channel where Cathy is listening. **Method:** A, B agree on the group G and generator g. Cathy knows this, but can't solve DLP on G. Then A privately chooses a random integer $a < g$,and sends the group element g^a to B. Similarly, B chooses a random b, and sends g^b to A. Their common secret information is $k = g^{ab}$, which they can use as their key.

- Alice knows a and g^b, so computes $k = (g^b)^a$.
- Bob knows b and g^a, so computes $k = (g^a)^b$.

- Cathy knows g^a and g^b, but cannot solve for a and b. It is unclear how she can compute g^{ab}. Maybe some method exists that allows Cathy to find g^{ab} without solving a DLP. (Partial results suggest that the answer is no.)
- Note that A does not know b, and does not have to try to find it, since A knows a.

ElGamal public-key cryptosystem: Essentially, a different Diffie-Hellman key exchange for each message. The space of messages is $\mathcal{M} = G$, and Alice has a private key $k_A = a < |G|$, and a public key $\hat{K}_A = g^a$. If Bob wants to send a message $m \in G$ to Alice, he chooses a random number $r < |G|$, and computes $h = g^r$ and $n = (\hat{K}_A)^r m$. Bob then sends the pair (h, n) to Alice.

- Alice knows $h = g^r$ and $n = g^{ar}m$, and can compute $m = h^{-a}n$. Note that Alice never finds out what r is, but can decode due to her knowing the private key a.
- Cathy cannot find m from n without somehow finding g^{ar}. But she only knows g^a and g^r.

ElGamal digital signature: This uses the same public-private key pair $k_A = a$, $\hat{K}_A = g^a$. Write $M = |G|$. We fix a simple bijection $f : G \to \mathbf{Z}/M\mathbf{Z}$, sending $h \in G$ to $f(h) \in \mathbf{Z}/M\mathbf{Z}$. This time, the messages belong to $\mathbf{Z}/M\mathbf{Z}$, and A signs $m \in \mathbf{Z}/M\mathbf{Z}$ by giving a pair $(h, s) \in G \times \mathbf{Z}/M\mathbf{Z}$ such that

$$(\hat{K}_A)^{f(h)} h^s = g^m.$$

Anyone can verify this signature.

- A can find the signature as follows: choose a random r and let $h = g^r$. Then the powers of g in the above equation are $af(h) + rs \equiv m \bmod M$. It is easy for A to solve for s using an inverse of $r \bmod M$ obtained by the Euclidean algorithm. Note that A uses her knowledge of the private key a.
- If Cathy fixes a choice of h and wants to forge the signature by finding the correct s, she needs to solve a DLP somewhere. If she starts from a fixed s, she needs to solve a strange nonlinear equation for h.

- Nobody knows (rather, has admitted to knowing) how to choose s and h simultaneously in some other way to forge a signature.

Remark: The ElGamal digital signature algorithm requires us to know the order $|G|$ of the group. We also need to know $|G|$ to make sure that the DLP in G is safe from the Silver-Pohlig-Hellman attack. But finding $|G|$ can be far from obvious.

- Example: how many points lie on the elliptic curve $E : y^2 = x^3 + x + 1$? More precisely, given the finite field $\mathbf{F}_p$, we can view E as being defined over $\mathbf{F}_p$. (Actually, if $p = 2$ or $p = 31$ the equation gives a singular curve, so it is not elliptic.)
- $E(\mathbf{F}_5) = \{P_\infty, (0,1), (0,4), (2,1), (2,4), (3,1), (3,4), (4,2), (4,3)\}$, so $|E(\mathbf{F}_5)| = 9$.
- But what about $E(\mathbf{F}_{12532716264317})$? Answer: this curve has 12532721750444 points. Not obtained by enumerating all the points!
- General result mentioned above: if E is any elliptic curve over the finite field $\mathbf{F}_q$, then $|E(\mathbf{F}_q)| =$

$q+1-t$ with $|t| \leq 2\sqrt{q}$. This a theorem of Hasse, and is analogous to the Riemann Hypothesis for the function field $\mathbf{F}_q(E)$.

- In the example above, $t = -5486126$, while $2\sqrt{p} \approx 7080315.3$. We can use the limited range of possible ts to direct our computation of the number of points on E.

- Two main polynomial-time approaches exist to counting points on E over a large finite field:
  - (i) the Schoof-Elkies-Atkin algorithm — works generally
  - (ii) Satoh's algorithm (also similar work by Kedlaya) — better for finite fields like $\mathbf{F}_{2^{500}}$ of small characteristic.

Factoring large integers

Let N be a very large number (say $N \sim 10^{300}$), which we know is composite because it has failed a primality test. How do we find factors of N?

Here are some general ways:

- Trial division: try all integers $j \leq \sqrt{N}$ to see if N is divisible by j. This takes **exponential time** since the input size is the number of digits of N, which is $O(\log N)$. In practice we do test all j up to a moderate bound, say a few million.

- Remark: a polynomial time algorithm means its running time is $O\Big((\log N)^c\Big) = O(\exp(c \log\log N))$ for some constant c. An exponential time algorithm takes time $O(N^c) = O(\exp(c \log N))$.

- The number field sieve takes (heuristic) time $O(\exp(c(\log N)^{1/3}(\log\log N)^{2/3}))$. This is "one-third of the way" between polynomial and exponential time. (For some ranges of N, the

simpler quadratic sieve is still better, even though its asymptotics are slower — "halfway" between polynomial and exponential time.)

The above running times are given in terms of the size of N, and allow for the worst case when the composite number N is the product of two prime numbers both approximately $\sqrt{N}$. Most composite N have a significantly smaller prime factor p. Trial division will then find this factor in time $O(p)$. But the number field sieve and quadratic sieve are insensitive to the size of p. We will describe two more methods that depend on the size of this smallest prime factor p, in preparation for the **elliptic curve factorization method**.

- A heuristic $O(\sqrt{p})$ algorithm: Pollard's rho method. Say for simplicity that $N = pq$, where p is significantly smaller than N — e.g., $N \sim 10^{300}$ but $p < 10^{20}$. Trial division to find p takes $O(10^{20})$ steps, too long. But the rho method takes $O(10^{10})$ steps, quite reasonable.

- Description of rho method: work in $\mathbf{Z}/N\mathbf{Z} \cong \mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/q\mathbf{Z}$. Take a nice nonlinear function

like $f(x) = x^2 + 1 \bmod N$ which "works on each coordinate independently." I.e., say $a \in \mathbf{Z}/N\mathbf{Z}$ corresponds to $(b, c) \in \mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/q\mathbf{Z}$, where $b \equiv a \bmod p$ and $c \equiv a \bmod q$. Then we mean that $f(a)$ corresponds to $\big(f(b), f(c)\big)$. Now start from a random $a \in \mathbf{Z}/N\mathbf{Z}$, and compute $f(a), f(f(a)), \ldots, f^M(a)$ where $M = O(\sqrt{p})$. Recall that $f^M(a)$ corresponds to the pair $(f^M(b), f^M(c))$. We can reasonably expect that for some $i, j \leq M$, we have $f^i(b) \equiv f^j(b) \bmod p$ because the values of the iterated function $f^\ell(b)$ **inside the smaller set** $\mathbf{Z}/p\mathbf{Z}$ eventually have to repeat themselves. But for that particular i, j we usually have $f^i(c) \not\equiv f^j(c) \bmod q$. **Conclusion** the GCD $(f^i(a) - f^j(a), N)$ gives us the factor p.

- The $(p-1)$ method: this works if N has a factor p such that $p-1$ is highly composite. We say that $p-1$ is B-**smooth** if $p - 1 = 2^a 3^b 5^c \ldots B^z$ with a largest prime factor B (which need not appear since we can have $z = 0$). Now p is unknown, so $a, b, c, \ldots, z$ are unknown. However, since $2^a \leq p - 1 < N$, we know that $a < (\log N)/(\log 2)$. Similarly, $b < (\log N)/(\log 3)$, and so forth. Thus

define

$$S = 2^{[\log N/\log 2]}3^{[\log N/\log 3]}\cdots B^{[\log N/\log B]}.$$

This guarantees that S is a multiple of $p-1$, so for any number x we have $x^S \equiv 1 \bmod p$.

- Description of the $(p-1)$ method: Again for simplicity say $N = pq$ with $p-1$ a B-smooth number. If $x \in \mathbf{Z}/N\mathbf{Z} \cong \mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/q\mathbf{Z}$, then x corresponds to $(x \bmod p, x \bmod q)$ and x^S corresponds to $(x^S \bmod p, x^S \bmod q) = (1 \bmod p, x^S \bmod q)$. Usually, $x^S \not\equiv 1 \bmod q$, so we can recover p as the GCD $(x^S - 1, N)$. Here the power $x^S \bmod N$ is as usual computed by fast exponentiation.

coordinates, $P_1 = [x_1 : y_1 : 1]$ and the identity element is the point at infinity $P_\infty = [0:1:0]$.

- The order of $E(\mathbf{Z}/p\mathbf{Z})$ is known to be at most $p+1+2\sqrt{p} < N$. So if $|E(\mathbf{Z}/p\mathbf{Z})|$ is B-smooth, then its order is a factor of

$$S = 2^{[\log N/\log 2]}3^{[\log N/\log 3]}\cdots B^{[\log N/\log B]},$$

and so $S \cdot P_1 \equiv P_\infty \bmod p$. In projective coordinates this means that $P_S = S \cdot P_1 = [X_S : Y_S : Z_S]$ with $Z_S \equiv 0 \bmod p$. We usually have $P_S \not\equiv P_\infty \bmod q$, so $Z_S \not\equiv 0 \bmod q$. Thus we can find the factor p as the GCD (Z_S, N).

- We will show below that the expected time for the elliptic curve method to find the factor p of N is **(roughly)** $O(\big(\exp(\sqrt{2}\cdot(\log p)^{1/2}(\log\log p)^{1/2}\big)$. This is "halfway between" polynomial and exponential time in the size of the smallest prime factor p of N. For most large N, this is the best way to factor since usually $p \ll N^c$. However, for the $N = pq$ used in RSA cryptography, where $p \sim q \sim \sqrt{N}$, the number field sieve (and quadratic sieve) are still asymptotically better.

The elliptic curve factorization method

Keep the simplifying assumption $N = pq$.

Reinterpretation of $p-1$ method: The multiplicative group $(\mathbf{Z}/N\mathbf{Z})^*$ decomposes as $(\mathbf{Z}/p\mathbf{Z})^* \times (\mathbf{Z}/q\mathbf{Z})^*$, and the first factor has order $p-1$ which we hope is B-smooth for a moderately sized B. The **disadvantage** is that p is a fixed factor of N, so we have no choice in the order $p-1$.

How to get many more choices: Replace the group $(\mathbf{Z}/N\mathbf{Z})^*$ by the group of points on an elliptic curve: $E(\mathbf{Z}/N\mathbf{Z}) \cong E(\mathbf{Z}/p\mathbf{Z}) \times E(\mathbf{Z}/q\mathbf{Z})$. Now the first factor has order $|E(\mathbf{Z}/p\mathbf{Z})| = p+1-t$ with $|t| \le 2\sqrt{p}$. Varying the elliptic curve will vary t, and there is a good chance that $p+1-t$ will be smooth.

Outline of Elliptic Curve factorization:

- Find a random elliptic curve E defined over $\mathbf{Z}/N\mathbf{Z}$, and a random point $P_0 \in E(\mathbf{Z}/N\mathbf{Z})$. Specifically, choose a random $x_1, y_1, a \in \mathbf{Z}/N\mathbf{Z}$, and determine $b \in \mathbf{Z}/N\mathbf{Z}$ such that $y_1^2 = x_1^3 + ax_1 + b$. Then the elliptic curve is $E : y^2 = x^3 + ax + b$, and $P_1 = (x_1, y_1)$. In projective

Asymptotics of elliptic curve method

Here is a heuristic justification of the rough expected time $O(\big(\exp(\sqrt{2}\cdot(\log p)^{1/2}(\log\log p)^{1/2}\big)$.

Principle: A number of the order of B^μ has a chance of about $\mu^{-\mu}$ of being B-smooth.

Since we want to find a factor p, we should try elliptic curves until $|E(\mathbf{F}_p)|$ is B-smooth. How should we choose B?

- $|E(\mathbf{F}_p)| \sim p$, so if we write $p = B^\mu$, we need to try about μ^μ elliptic curves.

- Trying ONE elliptic curve involves about $B\log N/\log B$ group operations. Reason: for each prime $q \le B$, we raise our point to the power q^{ℓ_q}, where $\ell_q \approx \log N/\log q$. So we raise to a power that is about N; this takes about $\log N$ group operations by fast arithmetic for each q. But the number of prime q's less than B is around $B/\log B$.

- Thus, choose μ with $B = p^{1/\mu}$ so as to minimize $\mu^\mu B\log N/\log B$. Note that $\log N$ is fixed.

So, let us choose μ to minimize

$$\Phi(\mu) = \frac{p^{1/\mu}}{\log p^{1/\mu}} \cdot \mu^{\mu} = \frac{p^{1/\mu}}{\log p} \cdot \mu^{\mu+1}.$$

Easier: minimize $\log \Phi$ after dropping the fixed $\log p$ from the denominator. Thus minimize

$$\Psi(\mu) = \frac{1}{\mu} \log p + (\mu + 1) \log \mu,$$

whose derivative is

$$\frac{d\Psi}{d\mu} = \frac{-1}{\mu^2} \log p + \log \mu + 1 + \frac{1}{\mu} \sim \frac{-1}{\mu^2} \log p + \log \mu$$

which is close to zero when $\mu^2 \log \mu \approx \log p$. As a "zeroth" approximation, μ is about $(\log p)^{1/2}$, up to a factor of order $\log \log p$. So we can well approximate $\log \mu \approx (\frac{1}{2}) \log \log p$. Substituting above, we get the more precise approximation

$$\mu \approx \sqrt{\frac{\log p}{\log \mu}} \approx \sqrt{\frac{2 \log p}{\log \log p}}.$$

End of the argument: We now know that the optimal value of $\Psi = \log(\Phi \log p)$ is

$$\Psi = \frac{1}{\mu} \log p + (\mu + 1) \log \mu \approx \sqrt{2 \log p \log \log p}.$$

The difference of $\log \log p$ between Ψ and $\log \Phi$ is hence negligible. Moreover, the actual time needed to factor is the time it takes to evaluate $\Phi \log N$ group operations in an elliptic curve modulo N, which takes time $O(\Phi (\log N)^c)$ for some c. This power of $\log N$ also contributes the negligible amount $c \log \log N$ to the **logarithm** of the amount of time needed, and so the amount of time actually needed is $O(\exp\big((1+\epsilon) \log \Phi\big))$.

Final answer: the time needed is

$$O(\exp(\big((1+\epsilon)\sqrt{2 \log p \log \log p}\big)).$$

Primality proving

Given a number N which has resisted trial division by moderately small numbers, and which has passed a number of Rabin-Miller tests (variations on $a^{(N-1)/2} \equiv \pm 1 \bmod N$ for several a), we are fairly certain that N is prime. But how can we be completely sure?

- The engineer's philosophical reason: if N were composite, then the probability of a given a passing the Miller-Rabin test is at most $1/4$, and is usually much, much less. So if N passes the test for 100 randomly chosen a, we can bet with odds of $4^{100} \approx 1.6 \times 10^{60}$ to 1 that N is prime. This is much better odds than the chance of a random fault in our computer! Besides, each Miller-Rabin test runs very quickly (time $O((\log N)^{1+\epsilon})$ with fancy techniques for multiplication modulo N).

- The mathematician's point of view: we still want a complete proof that N is prime, even if this takes longer. The AKS primality test, in its fastest modification, takes expected time $O((\log N)^{4+\epsilon})$ to prove primality probabilistically

(working deterministically, the exponent becomes 7.5). But it isn't yet the fastest method in practice for primes of the size used in cryptography, which have about 300 digits.

- Other methods exist, which give a **certificate of primality** that can be checked rather quickly, even if finding the certificate takes more time. We will discuss such a method using elliptic curves, which finds the certificate in heuristic probabilistic time $O((\log N)^{4+\epsilon})$, but with a better constant than AKS.

A simple primality certificate: Say we know the factorization of $N - 1 = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$. (Need a certificate of primality for each p_i, obtained recursively!) Now look for $a \in \mathbf{Z}/N\mathbf{Z}$ such that:

- $(a, N) = 1$,
- $a^{N-1} \equiv 1 \bmod N$,
- $a^{(N-1)/p_i} \not\equiv 1 \bmod N$ for all p_i.

THEN N is prime.

Conversely: if N is prime, then choosing a to be any generator of $(\mathbf{Z}/N\mathbf{Z})^*$ works. Such an a is called a primitive root modulo N.

Why this is a certificate: We have $a \in (\mathbf{Z}/N\mathbf{Z})^*$. Write d for the order of a in this group; thus d is the smallest possible number for which $a^d \equiv 1 \bmod N$. Now $N-1$ is a multiple of d, but $(N-1)/p_i$ cannot be a multiple of d for any i. The only choice is that $d = N-1$, so a has order $N-1$ in $(\mathbf{Z}/N\mathbf{Z})^*$, but $(\mathbf{Z}/N\mathbf{Z})^* \subset \mathbf{Z}/N\mathbf{Z} - \{0\}$ cannot have more than $N-1$ elements. Thus $(\mathbf{Z}/N\mathbf{Z})^* = \mathbf{Z}/N\mathbf{Z} - \{0\}$, and N is prime.

A slight variation: Say that we factor $N-1 = q\ell$ where q is a rather large prime, $q > \sqrt{N}-1$. Now say we find a such that

- $(a, N) = 1$,
- $a^{N-1} \equiv 1 \bmod N$,
- $(a^{(N-1)/q} - 1, N) = 1$.

Then N is prime because if p is any prime factor of N, then the order of a in $(\mathbf{Z}/p\mathbf{Z})^*$ is a multiple of

q, and hence $p - 1 \geq q > \sqrt{N} - 1$; thus all prime factors p of N satisfy $p > \sqrt{N}$, so N is prime.

Difficulty with both methods: We would have to be lucky enough to either factor $N-1$ completely, or ensure that $N-1$ has a large prime factor q. On the other hand, if N is prime, then it should be easy to come across a generator a of $(\mathbf{Z}/N\mathbf{Z})^*$ by random search.

Avoiding this difficulty: Just like for factoring, the solution is to be able to choose other groups than just $(\mathbf{Z}/N\mathbf{Z})^*$. We again work with elliptic curves defined over $\mathbf{Z}/N\mathbf{Z}$.

Basic idea: Say we find an elliptic curve E and a point $P \in E(\mathbf{Z}/N\mathbf{Z})$ such that P "has order somewhat larger than $\sqrt{N}$." Then N cannot have any prime factor $p \leq \sqrt{N}$, because the image of P modulo p generates a subgroup of $E(\mathbf{Z}/p\mathbf{Z})$ of order bigger than $\sqrt{N}$, which is too large compared to p. In fact, Hasse's theorem says that $|E(\mathbf{Z}/p\mathbf{Z})| \leq p + 1 + 2\sqrt{p} \leq (N^{1/4}+1)^2$.

More precise version: Suppose we find a prime $q > (N^{1/4}+1)^2$, an elliptic curve E defined over

$\mathbf{Z}/N\mathbf{Z}$, a point $P \in E(\mathbf{Z}/N\mathbf{Z})$, and a multiple m of our prime q, all satisfying

- $[m]P = [0:1:0]$ in projective coordinates,
- $[m/q]P = [a:b:c]$ with $(c, N) = 1$ in projective coordinates.

Then P has order a multiple of q in $E(\mathbf{Z}/N\mathbf{Z})$, or more generally when projected to any $E(\mathbf{Z}/p\mathbf{Z})$, and so N is prime.

Strategy to make certificates of primality: Start with N probably prime, and find q, E, P, m as above with $q \sim N^{1/2}$. The certificate that N is prime is the data q, E, P, m PLUS a certificate of the primality of q, obtained recursively. Since q is considerably smaller than N, this process terminates quickly enough. The **hard part** of the above strategy is in finding m and q.

First way (less practical but easier to explain): try many choices of E with a given point P, and use Schoof's algorithm to count $m_E = |E(\mathbf{Z}/N\mathbf{Z})|$ for each E. Then look for an $m = m_E$ which has

a large (probably) prime factor q. This is done by factoring out as much of m as possible, using, say, trial division and elliptic curve factorization.

Second way (more practical): Choose q, m first (!!) and look for an elliptic curve with $|E(\mathbf{Z}/N\mathbf{Z})| = m$. This is done by finding an elliptic curve over $\overline{\mathbf{Q}}$ with **complex multiplication**, and reducing "modulo N."

Complex multiplication for elliptic curves over $\mathbf{C}$: Then $E \cong \mathbf{C}/L$ for a lattice $L \subset \mathbf{C}$. A homomorphism $\varphi : E \to E$ is determined by a number $\alpha \in \mathbf{C}$ such that $\alpha L \subset L$. The map is:

$$\begin{array}{ccc} \mathbf{C} & \xrightarrow{\cdot\alpha} & \mathbf{C} \\ \downarrow & & \downarrow \\ \mathbf{C}/L & \xrightarrow{\varphi=[\alpha]} & \mathbf{C}/L \end{array}$$

I.e., $\varphi(z \bmod L) = \alpha z \bmod L$.

- $\mathrm{End}(E) = \{\alpha \in \mathbf{C} \mid \alpha L \subset L\}$ is the endomorphism ring of E.
- $\mathrm{End}(E)$ contains the multiplication-by-m maps

$[m]$ for $m \in \mathbf{Z}$. But occasionally it can be larger. In that case we say that E has **complex multiplication**.

- Some examples:
  - If $L = \mathbf{Z} + \mathbf{Z} \cdot i$, then the corresponding $\mathrm{End}\, E$ is $\mathbf{Z}[i]$.
  - If $L = \mathbf{Z} \cdot 2 + \mathbf{Z} \cdot 5i$, then $\mathrm{End}\, E = \mathbf{Z}[10i]$.
  - If $L = \mathbf{Z} \cdot 2 + \mathbf{Z} \cdot (1 + \sqrt{-3})$, then $\mathrm{End}\, E = \mathbf{Z}[\frac{1+\sqrt{-3}}{2}]$.

- Over $\mathbf{C}$, or more generally in characteristic zero, the possible endomorphism rings of complex multiplication have the form $\mathcal{O} \subset \mathbf{Q}(\sqrt{-d})$, where $\mathcal{O}$ is an order in the imaginary quadratic field $K = \mathbf{Q}(\sqrt{-d})$. So $\mathcal{O} = \mathbf{Z} + \mathbf{Z}\beta$ for an algebraic integer $\beta \in K$. The lattice L can be taken to be a suitable ideal $\mathfrak{a} \subset \mathcal{O}$, viewed as a subset of $\mathbf{C}$. So $E \cong \mathbf{C}/\mathfrak{a}$.

Part of Big Theorem: Assume that E has complex multiplication by an order $\mathcal{O} \subset K$, where K is an imaginary quadratic field. Then

- The j-invariant $j(E)$ is an algebraic integer.

- The field $K[j(E)]$ is a Galois extension of K, with **abelian** Galois group canonically isomorphic to the class group of $\mathcal{O}$, and very little ramification. (In terms of class field theory: $K[j(E)]$ is the ring class field of K with respect to $\mathcal{O}$.)

- Write an equation for E over $K[j(E)] \subset \overline{\mathbf{Q}}$. This equation can be reduced modulo almost all primes as follows: take a prime number $p \in \mathbf{Q}$, and a prime ideal $\mathcal{P}$ of the ring of integers of $K[j(E)]$, lying over p, with residue field $\mathbf{F}_{p^r}$. Then we can reduce the equation of E to obtain an elliptic curve $\overline{E}$ defined over $\mathbf{F}_{p^r}$.

- The theory predicts r easily from the ideal structure of $\mathcal{O}$, without needing to find an exact value of $j(E)$. In case $p = \alpha\overline{\alpha}$ with $\alpha \in \mathcal{O}$, then $r = 1$.

- The theory also gives a manageable formula for $\left|\overline{E}(\mathbf{F}_{p^r})\right|$. In case $p = \alpha\overline{\alpha}$ as above, we have

$$\left|\overline{E}(\mathbf{F}_p)\right| = p + 1 - \alpha' - \overline{\alpha'}$$

where α' is usually equal to $\pm\alpha$. (More generally, $\alpha' = \zeta\alpha$ with ζ a root of unity in K.)

Constructing elliptic curves to prove primality

Given N, which we want to prove prime.

- Look for a (small) d and $\alpha = x + y\sqrt{-d} \in K = \mathbf{Q}(\sqrt{-d})$, for which $x^2 + dy^2 = \alpha\overline{\alpha} = N$. ($x, y$ are integers or half-integers.)

- We know an elliptic curve $\overline{E}$ over the alleged finite field $\mathbf{F}_N$ with complex multiplication by the maximal order $\mathcal{O} \supset \mathbf{Z}[\alpha]$ usually has order $m = N + 1 \mp (\alpha + \overline{\alpha}) = N + 1 \mp 2x$. Try to factor m at least partially, to find a factor q of m with $q > (N^{1/4} + 1)^2$ and q probably prime, say by the Miller-Rabin test. Repeat with different d's until one such m is found with an appropriate factor q.

- **The hard part** Look for an elliptic curve E defined over $\overline{\mathbf{Q}}$ with complex multiplication by $\mathcal{O}$, and reduce modulo a prime above N. The idea is to look for $j(E)$ and all its Galois conjugates.
  - The conjugates are $j(\mathbf{C}/\mathfrak{a})$, for $\mathfrak{a}$ a set of representatives for the ideal class group of $\mathcal{O}$.

These are essentially classes of binary quadratic forms of discriminant $-d$ or $-4d$.
  - Since the j-values are algebraic integers, the polynomial $P(X) = \prod_{\mathfrak{a}}(X - j(\mathbf{C}/\mathfrak{a}))$ has integral coefficients. Now j can be calculated numerically as a complex number, so calculate the values of j to sufficient accuracy to get the coefficients of $P(X)$ with an error < 0.5.
  - We really want $j(\overline{E})$, which is the reduction of $j(E)$ modulo a prime $\mathcal{N}$ lying above N. We find this value by factoring $P(X)$ over $\mathbf{F}_N$.
  - Having found $j(\overline{E}) \in \mathbf{F}_N$, we construct the curve $\overline{E}$, search for a point $P \in \overline{E}(\mathbf{F}_N)$, and check that $[m]P = P_\infty$, $[m/q]P = [a : b : c]$ with $(c, N) = 1$ in projective coordinates.
  - Finally, recursively produce a primality certificate for q.

Error-correcting codes

Here the word "code" does not refer to cryptography. Instead, we are interested in how to send information across a **noisy** channel while minimizing errors in transmission.

Typical examples:

- Using a high-speed computer modem across a phone line with crackling and hissing,
- Having CDs and CD-ROMs that still work despite scratches and random faults,
- Communicating with a far-off satellite with a weak signal.

Example of a code: Every time we transmit three bits of data, we then send a fourth bit for "parity check," to make the number of 1 bits **even**. This allows us to detect if one bit was garbled in transmission, but we cannot correct it.

000	001	010	011	100	101	110	111
000**0**	001**1**	010**1**	011**0**	100**1**	101**0**	110**0**	111**1**

Formal setup for error-correcting codes: We fix a finite alphabet A, e.g., we used $\{0,1\}$ above, which we can identify with $\mathbf{F}_2$; more generally, we will stick to $A = \mathbf{F}_q$.

- A word of length n is an element $\mathbf{a} = (a_1, \ldots, a_n) \in A^n$.
- A **code** is a subset $\mathcal{C} \subset A^n$. An element of $\mathcal{C}$ is called a **codeword**.
- The **Hamming distance** between two words of length n is the number of places with differing letters: $d(\mathbf{a}, \mathbf{b}) = |\{i \mid a_i \neq b_i\}|$. Example: $d(1\underline{1}01\underline{1}1\underline{0}, 1\underline{0}01\underline{0}1\underline{1}) = 3$.
- The **minimum distance** $d(\mathcal{C})$ of the code $\mathcal{C}$ is the smallest value of $d(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \neq \mathbf{b} \in \mathcal{C}$.

Basic fact: Using $\mathcal{C}$ with $d(\mathcal{C}) = d$ allows us to detect $< d$ errors in each codeword. We can correct $< d/2$ errors per codeword. For a given n, d, our goal is to make $|\mathcal{C}|$ as large as possible.

Linear codes

This means that $A = \mathbf{F}_q$ is a finite field, and we take $\mathcal{C} \subset A^n$ to be an $\mathbf{F}_q$-subspace.

- If $k = \dim \mathcal{C}$, then $|\mathcal{C}| = q^k$; so a codeword transmits k letters of information using n symbols.
- $d(\mathcal{C})$ is easy for linear codes, since $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{a} - \mathbf{b}, \mathbf{0})$, and $\mathbf{a}, \mathbf{b} \in \mathcal{C} \implies \mathbf{a} - \mathbf{b} \in \mathcal{C}$. Thus $d(\mathcal{C})$ is the minimum number of nonzero entries in any nonzero codeword.
- Encoding of a linear code is easy, since we just need linear algebra to describe an injection $\varphi : \mathbf{F}_q^k \to \mathbf{F}_q^n$ with image $\mathcal{C}$. We can also decode by solving the linear system $\varphi(\mathbf{x}) = \mathbf{a}$; if the system has no solution, we know that a transmission error occurred. We will describe later how to **correct** fewer than $d/2$ errors for certain codes.
- Our parity-check example: $q = 2$ and $\mathcal{C} = \{(a_1, a_2, a_3, a_4) \in \mathbf{F}_2^4 \mid \sum a_i = 0\}$. The minimum distance is $d = 2$, so we can detect < 2 errors (i.e., ≤ 1 error), and correct $< 2/2$ (i.e., 0) errors.

Example: the $[7, 4]$ Hamming code

Here $\mathcal{C} \subset \mathbf{F}_2^7$ is the kernel of the matrix $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$, whose columns list the numbers 1–7 in binary.

Hence $\mathcal{C}$ is the subspace of vectors $(a_1, \ldots, a_7)$ in $\mathbf{F}_2^7$ satisfying $a_5 = a_2 + a_3 + a_4$, $a_6 = a_1 + a_3 + a_4$, and $a_7 = a_1 + a_2 + a_4$. Its elements are:

$$\begin{array}{llll} (0000000), & (0001111), & (0010110), & (0011001), \\ (0100101), & (0101010), & (0110011), & (0111100), \\ (1000011), & (1001100), & (1010101), & (1011010), \\ (1100110), & (1101001), & (1110000), & (1111111). \end{array}$$

Here (1101001) is shorthand for $(1, 1, 0, 1, 0, 0, 1)$. We see that the minimum distance of this code is $d = 3$, which is the fewest number of 1s in any nonzero codeword. Thus we can detect < 3 errors, i.e., up to two errors, and we can correct $< 3/2$ errors, i.e., up to one error.

Exercise: Figure out how to correct one error if it occurs. The matrix above is useful.

Efficiency of the Hamming code: We use 7 bits to send 4 bits of information, since $\dim \mathcal{C} = 4$. Our limit is 3 errors. Essentially, we can say that $4/7 \approx 57\%$ of the information that we transmit corresponds to the actual data we want to send, and the remaining $< 3/7 \approx 43\%$ allows for errors. This code is best possible in some sense, since (essentially) the sum cannot exceed 100% by some general theorems.

Reed-Solomon codes

Given n, q with $q \geq n$, we choose n distinct elements $t_1, \ldots, t_n \in \mathbf{F}_q$, and we define the code $\mathcal{C} \subset \mathbf{F}_q^n$ by

$$\mathcal{C} = \{\mathbf{a}_f := \big(f(t_1), \ldots, f(t_n)\big) \mid f(x) \in \mathbf{F}_q[x], \deg f \leq k\},$$

for some fixed $k < n$. Thus the map sending f to $\mathbf{a}_f$ is injective, since a nonzero f cannot have roots at all of $t_1, \ldots, t_n$. The dimension of this code is $\dim \mathcal{C} = k + 1$. (In case $k = n - 1$, $\mathcal{C}$ is all of $\mathbf{F}_q^n$, which is uninteresting. The purpose is to have k somewhat smaller than n, so we include more values of the polynomial f than are necessary to determine f.)

Immediate observation: The minimum distance of the above code is $d(\mathcal{C}) = n - k$. Reason: a nonzero f of degree $\leq k$ can vanish at up to k of the t_i. Thus we can detect $< n - k$ errors, and can correct $< (n - k)/2$ errors. Note that Reed-Solomon codes are hence also optimal from the point of view of information theory. Their **disadvantage** is that we need to take $q \geq n$, so the alphabet gets large.

Goppa (i.e., algebraic geometry) codes

Reinterpretation of Reed-Solomon codes using $\mathbf{P}^1$: We know $\mathbf{P}^1(\mathbf{F}_q) = \mathbf{F}_q \cup \{P_\infty\}$. Take the divisor $D = k \cdot P_\infty$; then $\mathcal{L}(D)$ is the space of polynomials $f(x)$ with $\deg f \leq k$. We then associate to $f \in \mathcal{L}(D)$ the codeword $\mathbf{a}_f = \big(f(t_1), \ldots, f(t_n)\big)$, where $t_1, \ldots, t_n$ are distinct points of $\mathbf{P}^1(\mathbf{F}_q)$.

Goppa codes: Take a curve X of genus g, defined over $\mathbf{F}_q$, and choose n distinct points $P_1, \ldots, P_n \in X(\mathbf{F}_q)$. Take a divisor D, with $\deg D = k < n$, and with D disjoint from $\{P_1, \ldots, P_n\}$. Then the corresponding Goppa code is

$$\mathcal{C} = \{\big(f(P_1), \ldots, f(P_n)\big) \mid f \in \mathcal{L}(D), \text{ i.e., } \operatorname{div}(f) \geq -D\}.$$

Note that the codeword $\mathbf{a}_f := \big(f(P_1), \ldots, f(P_n)\big)$ is well-defined since f can have poles only at D, which is disjoint from the P_i.

- The map $f \mapsto \mathbf{a}_f$ is injective, since if $f \neq 0$, then the number of zeros of f equals the number of poles, which is $\leq \deg D = k < n$.

- By Riemann-Roch, $\dim \mathcal{C} = \dim \mathcal{L}(D) = k + 1 - g + i(D)$. Usually we have $k \geq 2g - 1$ so $i(D) = 0$, and our code has g fewer dimensions than the corresponding Reed-Solomon code.

- The minimum distance is $d(\mathcal{C}) \geq n - k$, by the same reasoning as for Reed-Solomon codes. However, the inequality can now be strict. In principle, we can hope to get $d(\mathcal{C}) = n - k + g$ in some cases (roughly: ensure that the P_i are in linear general position with respect to $\mathcal{L}(D)$).

- Goppa codes give (asymptotically) the best performance known over a wide range of error rates. The idea is that the percentage of data transmitted is $(\dim \mathcal{C})/n \geq (k + 1 - g)/n$, in order to deal with an error rate $< (n - k)/n$. The sum is $(n + 1 - g)/n$, which is close to 1 if n can grow faster than g. For given q and g, Weil's theorem (the "Riemann Hypothesis for curves," generalizing Hasse's theorem), says that $|X(\mathbf{F}_q)| \leq q + 1 + 2g\sqrt{q}$. This is the maximum possible n, and families of curves are known with $g \to \infty$ which come close to this bound. They give asymptotically very good codes.

Decoding Reed-Solomon codes

How do we correct (not just detect) errors?

The basic problem: Given $\mathbf{b} = (b_1, \ldots, b_n)$, we want to find f of degree $\leq k$ such that $\mathbf{a}_f = \big(f(t_1), \ldots, f(t_n)\big)$ differs from $\mathbf{b}$ in $\ell < (n-k)/2$ coordinates. So we must allow up to ℓ wrong letters. To do this, we allow a corrector polynomial $c(x)$ with $\deg c \leq \ell$, that can vanish at the "bad" t_i. We now try to find polynomials c and f satisfying

$$\big(c(t_1)b_1, \ldots, c(t_n)b_n\big) = \big(c(t_1)f(t_1), \ldots, c(t_n)f(t_n)\big)$$

Now write $F(x) := c(x)f(x)$ with $\deg F \leq k+\ell$, and solve a **linear system** of equations for the coefficients of F and c. (We temporarily forget about f.)

- If $\mathbf{b}$ is within distance ℓ of a codeword $\mathbf{a}_f$, then a nontrivial solution (c_0, F_0) exists to this homogeneous system, namely the one with c_0 vanishing at the "bad" t_i, and $F_0 = fc_0$.
- **Claim:** If (c, F) is any nontrivial solution of the linear system, then $c(x)$ is a factor of $F(x)$, and $f = F/c$.

Proof of claim:

- Recall our system of equations is $c(t_i)b_i = F(t_i)$, for $i = 1, \ldots, n$.
- Note that $c(x)$ is not the zero polynomial. Otherwise $F(x)$ vanishes at $t_1, \ldots, t_n$, but $\deg F \leq k + \ell < n$, so $F(x)$ is also the zero polynomial. But we assumed that (c, F) was a nontrivial solution of our homogeneous linear system.
- Now use the existence of one "correct" solution (c_0, F_0) with $f = F_0/c_0$. It is enough to show that any other solution (c, F) satisfies $cF_0 = c_0F$.
- Both $c(x)F_0(x)$ and $c_0(x)F(x)$ are polynomials of degree $\leq \ell + k + \ell = k + 2\ell < n$, since $\ell < (n-k)/2$. They have the same value at all the t_i, namely $c(t_i)c_0(t_i)b_i$. Thus c_0F and cF_0 must be the same polynomial. Q.E.D.

Decoding Goppa codes (nonoptimal)

- We take the space of c's to be $\mathcal{L}(E)$, where E is a suitable divisor of degree $e < (n-k+g)/2$. This only ensures that c can vanish at $e-g < (n-k-g)/2$ arbitrary points, less than the "expected" $\ell = d(\mathcal{C})/2 \geq (n-k)/2$.
- E must be disjoint from the P_i. We also choose E so that $\mathcal{L}(D + 2E - P_1 - \cdots - P_n) = 0$. This is reasonable since $\deg D + 2E - P_1 - \cdots - P_n = k + 2e - n < g$, so a "generic" choice of E will make $\mathcal{L}(D + 2E - P_1 - \cdots - P_n) = 0$.
- The corresponding product F is the product of $f \in \mathcal{L}(D)$ by $c \in \mathcal{L}(E)$, so $F \in \mathcal{L}(D+E)$. We thus solve the system $c(P_i)b_i = F(P_i)$, for $i = 1, \ldots, n$, and obtain a nontrivial pair (c, F).
- If we only aim to correct $< (n-k-g)/2$ errors, and find f, we know that a nontrivial solution (c_0, F_0) exists with $F_0/c_0 = f$. But then $cF_0 - c_0F$ is an element of $\mathcal{L}(D+2E)$ that vanishes at all P_i, so is zero by construction. Thus $F/c = F_0/c_0$. (Note $c \neq 0$, since otherwise we get $F = 0$.)

Very brief bibliography

- Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic Curves in Cryptography*, Cambridge Univ. Press
- Henri Cohen, *A Course in Computational Algebraic Number Theory*, Springer
- Neal Koblitz, *A Course in Number Theory and Cryptography*, Springer
- René Schoof, Algebraic curves and coding theory, Abuja 1990, available from `http://swc.math.arizona.edu/oldaws/00Notes.html`
- Joseph H. Silverman and John Tate, *Rational Points on Elliptic Curves*, Springer
- Madhu Sudan, Course notes on coding theory, `http://theory.lcs.mit.edu/~madhu/coding/course.html`
- Jacobus Hendricus van Lint, *Introduction to Coding Theory*, Springer

. . . and a free software package: GP/PARI, available from `http://pari.math.u-bordeaux.fr/`