Cosc {3,4}12: Cryptography and security
Lecture 3 (24/7/2023)
Stream ciphers,
Semantic security,
Agreeing a secret,
Asymmetric cryptosystems.

Michael Albert
michael.albert@otago.ac.nz

The problem with one time pads

- One time pads offer perfect security but,
- The key size and message size have to be the same,
- Key exchange is a limiting factor,
- Could we make do with a smaller key that somehow generates a full-sized key that *looks* random (enough?)

Pseudo-random generators and stream ciphers

- A pseudo-random generator, G, is a function that takes a *seed* and produces a much longer sequence efficiently:

$$G : \mathbf{2}^s \to \mathbf{2}^n \quad \text{where } s << n.$$

- If we agree about the seed (key) then we have access to a “long” sequence of agreed bits, which we can use as if it were a one time pad:

$$E(k, m) = G(k) \oplus m \quad D(k, c) = G(k) \oplus c.$$

- In what sense can the corresponding cipher be secure?

Indistinguishable from random

- The set of outputs of a PRG is tiny in the output space – how could it possibly look random?
- The adversary has to be testing the output and is somewhat limited
- So we ask:

PRG quality assurance

Is there an *effective* algorithm that *distinguishes* between the output of our PRG and truly random sequences?

- What does *effective* mean? What does *distinguishes* mean?

Colouring sequences

- A *statistical test* is a map $A : \mathbf{2}^n \to \mathbf{2}$
- Informally, A takes sequences as inputs and assigns them to one of two colours (let's say red for 0 and blue for 1).
- On truly random sequences, A will colour some proportion of the sequences red, and some proportion blue (maybe 50/50 but equally it could be 90/10).
- On output from our PRG, A will also colour some proportion red and some proporition blue.
- If the proportions differ significantly, then A *distinguishes* between truly random sequences and output from the PRG.
- The (absolute value of) the difference in proportion of red (or blue) is called the *advantage* of A over our PRG.

PRG Security

A PRG is secure if there is no efficient statistical test which has a non-negligible advantage over G.

- Why is efficiency important here?
- Can we build a provably secure PRG? (Probably not!)

The problem of key agreement

- Alice and Bob need to efficiently carry out an encrypted conversation of some length (upwards of tens of kilobytes).
- They have access to a fast and secure shared-key cryptosystem requiring a key of not more than a few tens or a few hundred bytes.
- Unfortunately they have no shared key.
- They need to “agree a secret” across an open channel.
- They’re happy to spend some (reasonable) amount of time on key agreement since, thereafter, encryption and decryption are very fast.

Prehistory: Merkle puzzles

- Alice sends Bob a large number of small encrypted texts using unknown (but fairly short) keys
- Bob chooses one randomly and decrypts it by a brute force attack
- The message contains a key to share, and an identifier - Bob sends the identifier to Alice
- For Eve to attack this she must decrypt (on average) half the messages
- Not really feasible in practice, but a proof of concept!

Diffie-Hellman key exchange

Uses a trick “exponentiation modulo a prime is easy but computing logarithms is hard”

- Alice and Bob publicly agree on a large prime p, and a primitive root g modulo p
- Alice randomly chooses $2 \leq a \leq p-2$ and transmits $g^a \pmod{p}$
- Bob randomly chooses $2 \leq b \leq p-2$ and transmits $g^b \pmod{p}$
- Alice computes $(g^b)^a = g^{ab}$ and Bob computes $(g^a)^b = g^{ab}$
- They hash this shared value to get the key

The big idea (public key encryption)

- There's no reason that the key used in encryption should be the same as the key used in decryption
- That is, we could have a pair of algorithms E and D and a pair of keys k_e and k_d such that:
$$D(k_d, E(k_e, m)) = m$$
- If we could publicly announce (k_e, E, D) without compromising k_d then we'd seem to be in good shape
- There are details to worry about!

Trapdoor functions

A *trapdoor* function is a triple (G, F, F^{-1}) of efficient algorithms:

- G is a randomised algorithm that outputs a key pair (p, s) (public, secret)
- For any p, $F(p, \cdot)$ defines a function $X \to Y$
- For any s such that (p, s) is a key pair, $F^{-1}(s, \cdot)$ is a function $Y \to X$ that inverts $F(p, \cdot)$, i.e, $F^{-1}(s, F(p, x)) = x$ for all x in X
- The trapdoor is secure if no efficient algorithm given p, and $y = F(p, x)$ (where x is chosen randomly from X) guesses x with non-negligible probability.

Public key systems from trapdoor functions

Starting from a secure trapdoor, a symmetric encryption scheme (E_s, D_s) and a hash function $H : X \to K$ (which makes "random X" look like "random K"):

- Generate a key pair (p, s) (and publish p)
- To encrypt m:
  - Choose x randomly from X
  - Let $y = F(p, x)$ and $k = H(x)$
  - Compute $c = E_s(k, m)$
  - Transmit (y, c)
- To decrypt:
  - Compute $x = F^{-1}(s, y)$
  - Compute $k = H(x)$
  - Compute $m = D_s(k, c)$.

Character of known public-key cryptosystems

- All common public-key cryptosystems rely on being able to compute efficiently "modulo N" i.e., when we take remainders after dividing by some reasonably large (several hundred to a few thousand bits) number N.
- So how do we do that?

Products modulo N using at most one extra bit

Problem: Compute $a \times b \pmod{N}$
Assumption: $0 \leqslant a, b < N$

while $b > 0$ **do**
  if $b \,\%\, 2 == 1$ **then**
    $c \leftarrow c + a \pmod{N}$
  end if
  $b \leftarrow b/2$
  $a \leftarrow 2 \times a \pmod{N}$
end while
return: c

Exponents modulo N

Problem: Compute $a^n \pmod N$
Assumption: $0 \leqslant a < N$, $0 \leqslant n$

$c \leftarrow 1$
while $n > 0$ **do**
  if $n \% 2 == 1$ **then**
    $c \leftarrow c \times a \pmod N$
  end if
  $n \leftarrow n/2$
  $a \leftarrow a \times a \pmod N$
end while
return: c

To note

- It can be arranged so that all the “modulo N” computations operate on numbers which are at most $2N$.
- In that context that means “subtract N if greater than or equal to N”.
- There may be some characteristics of N that make this a few machine-instructions faster (fewer 1 bits, a certain pattern of 1 bits, . . .)
- Not my area of expertise!

The holy grail of public key cryptosystems

Consists of (at least) three parts:

- Find an NP-complete problem for which almost all random instances are hard.
- Build a trap-door function around it that can only be opened by solving a random instance.
- Make sure it's resistant to quantum attacks (just in case).

It's not clear that this is completely achievable – though modern forms of *homomorphic encryption* come close (foreshadowing!).

The mathematics of the RSA trapdoor

- Let p and q be large primes, and $N = pq$
- Public key (N, e) and private key (N, d) where

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

- Given x which is coprime to N,

$$F(e, x) = x^e \pmod{N}$$

- And

$$F^{-1}(d, y) = y^d \pmod{N}$$

Signatures in RSA

- RSA is quasi-symmetric in that messages encoded with the private key could be decoded using the public key
- This allows a simple signature mechanism
- Bob transmits (with Alice's public key):

$$E(p_{\text{alice}}, \text{“From Bob: } E(s_{\text{bob}}, m)\text{”})$$

- Alice strips the header and decodes the message with Bob's public key
- So long as Bob's private key is private, no one else could have sent the message

Elliptic curves

For our purposes, an elliptic curve is the set of points (x, y) satisfying:

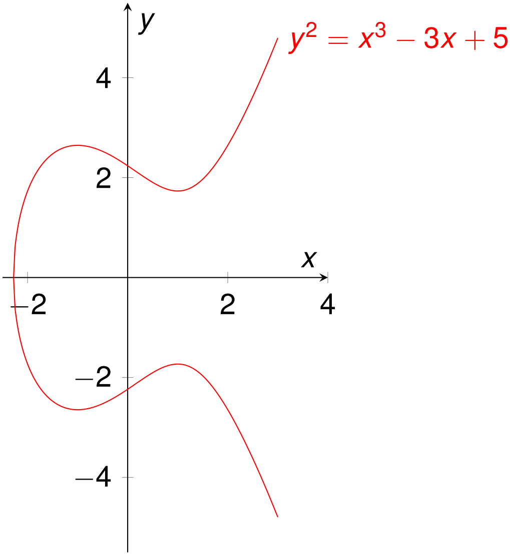
$$y^2 = x^3 + ax + b$$

for some parameters a and b.

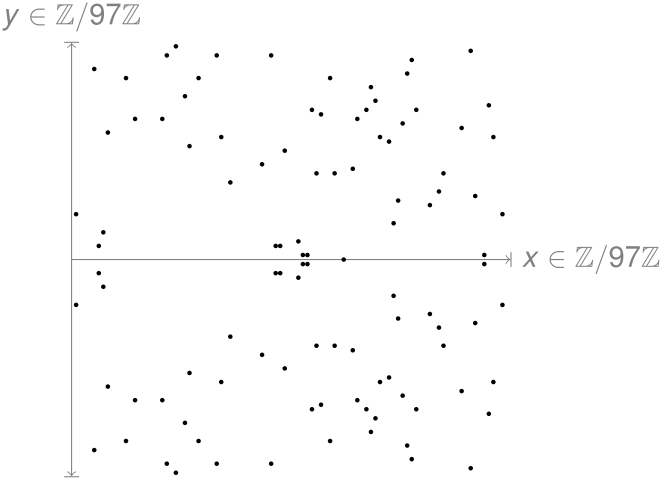
Set of points *where*?

Any conditions on a and b? $(4a^3 + 27b^2 \neq 0)$

An elliptic curve over $\mathbb{R}$

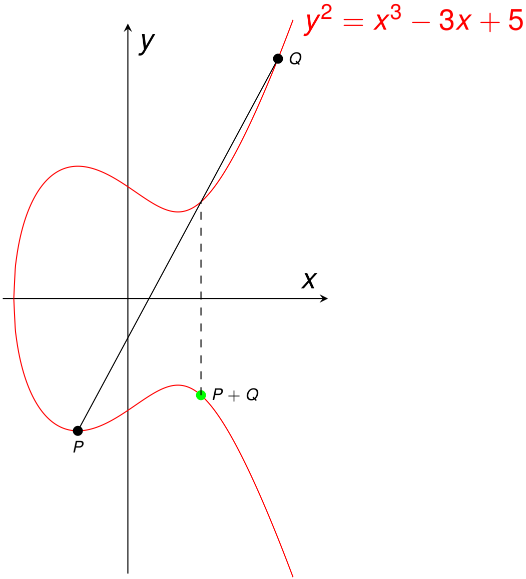


And over $\mathbb{Z}/97\mathbb{Z}$

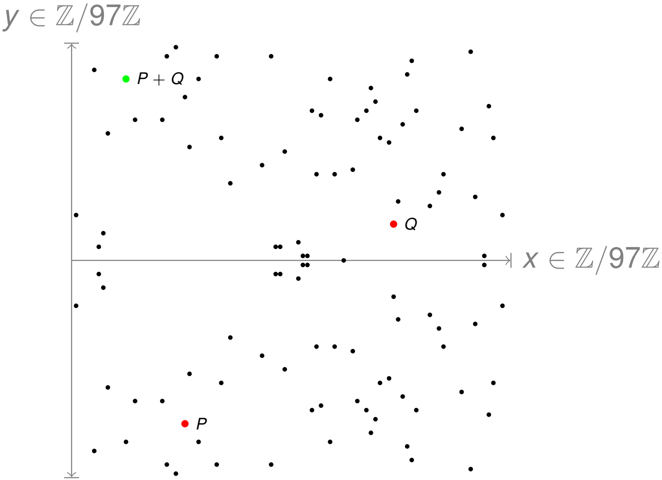


$y^2 = x^3 - 3x + 5$ over $\mathbb{Z}/97\mathbb{Z}$

A sum on the curve over $\mathbb{R}$



A sum on the curve in $\mathbb{Z}/97\mathbb{Z}$



But the formulas are the same!

Let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ and assume for the moment that $x_P \neq x_Q$.

Let
$$m = \frac{y_P - y_Q}{x_P - x_Q}.$$

Then the line joining P and Q has the equation:

$$y = y_P + m(x - x_P) = mx + d$$

Now consider points that are both on that line and on the curve:

$$\begin{aligned}(mx + d)^2 &= x^3 + ax + b \\ 0 &= x^3 - m^2x^2 + \cdots\end{aligned}$$

But the formulas are still the same!

If $R = (x_R, y_R)$ is the third point on both the curve and the line then:

$$(x - x_P)(x - x_Q)(x - x_R) = x^3 - m^2 x^2 + \cdots$$
$$x^3 - (x_P + x_Q + x_R)x^2 + \cdots = x^3 - m^2 x^2 + \cdots$$

So

$$\begin{aligned} x_P + x_Q + x_R &= m^2 \\ x_R &= m^2 - x_P - x_Q \\ y_R &= y_P + m(x_R - x_P). \end{aligned}$$

To compute R we only need to do one division (to get m), and a few multiplications and additions.

And then?

- After dealing with a bunch of edge cases the elliptic curve becomes a group. What? Why?
- Magic!
- Or, at least mathematics that I'm not prepared to explain (for all senses of the word "prepared").
- If we start with a point G on the curve then we can look at G, $2G$, $3G$ until we get to $nG = 0$ for some value of n.
- For any a and b and prime p the size of the curve over $\mathbb{Z}/p\mathbb{Z}$ is close to p. The number n must be a divisor of that size (that's group stuff) and we aim for a generator in which n is exactly equal to that size.
- Then it turns out that the sequence $G, 2G, 3G, \ldots$ looks pretty random!

Use for Diffie-Hellman key exchange

- All parties agree on a large prime p, some elliptic curve over $\mathbb{Z}/p\mathbb{Z}$, and some generator G of order n on the curve.
- Here "all parties" could be as much as "everyone on Facebook" or "every Amazon customer" (p is several hundred bits large).
- Each party chooses a public key, by taking a private k at random and announcing kG.
- Alice has $A = k_a G$, Bob has $B = k_b G$ (A and B are public).
- Their common key (no further communication required) is (an agreed hash of) $k_a k_b G$ which Alice can compute as $k_a B$ and Bob as $k_b A$.

ECC vs. RSA

- ECC needs much smaller key sizes. To prevent attacks using fewer than 2^{128} bit operations requires an ECC key of only 256 bits (in practice 384 is usually used) but an RSA key of 3072 bits.
- The ECC arithmetic is really simple and with good choices of p even the "modulo p" operation can be accelerated.
- ECC is generally superior to RSA. But . . .
- ECC parameters p, a and b (and to a lesser extent G and n) can't be chosen at time-of-use, as the RSA $N = pq$ can be. So, use of ECC is reliant on standard curves.
- Some people don't trust the standards.
- However, the main ECC functionality can't be backdoored (as far as anyone knows).
- Both ECC and RSA are vulnerable to quantum attack.
- See Koblitz and Menezes paper in resources.