Stream ciphers; semantic security; agreeing secrets; asymmetric crypto

COSC312 + COSC412—Cryptography and Security

David Eyers (with thanks to Michael Albert!)

The problem with one time pads

- One time pads offer perfect security but:
  - 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 & stream ciphers

- Pseudo-random generator, G, is a function that takes a seed, produces much longer sequence efficiently:

$$G : \mathbf{2}^s \to \mathbf{2}^n \text{ where } s \ll n$$

- If agree about seed (key) then have access to "long" sequence of agreed bits: can use as if a one time pad:

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

- In what sense can corresponding cipher be secure?

Indistinguishable from random

- Set of outputs of pseudo-random generator (PRG) is tiny in output space—how could it possibly look random?
  - Adversary has to be testing the output and is somewhat limited
- PRG quality assurance: Is there an *effective* algorithm that *distinguishes* between output of our PRG and truly random sequences?
  - Need to define what *effective* and *distinguishes* mean here

Colouring sequences

- A statistical test is a map $A : \mathbf{2}^n \to \mathbf{2}$
- Informally, A takes sequences as inputs; assigns them as one of two colours (say red for 0 and blue for 1).
  - On truly random sequences, A colours some proportion of sequences red, and some proportion blue.
    - (Maybe 50/50 but it could be 90/10.)
  - On output from our PRG, A also colours proportion red & 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 encrypted conversation of some length ($\gg$ tens of kilobytes).
- They have access to fast and secure shared-key cryptosystem with key length $\approx$ tens or hundreds 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; decrypts it by 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

- Trapdoor fn. is 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 secure trapdoor, symmetric encryption scheme (E_s, D_s) and hash function $H : X \to K$
  - (Hash function 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 number *N*.
    - Where "reasonably large" is several hundred to a few thousand bits

- So how do we do that?

Products mod N using at most one extra bit

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

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

Exponents modulo N

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

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

Optimising the maths for computation

- Can be arranged so that all "modulo *N*" computations operate on numbers which are at most 2*N*.
  - Means "subtract *N* if greater than or equal to *N*".
  - May be some characteristics of *N* that make this a few machine-instructions faster
    - fewer 1 bits, a certain pattern of 1 bits, ...
- We won't explore this, though
  - (Can be thankful others have done so!)

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 later lectures!).

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,

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

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):

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

  - 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

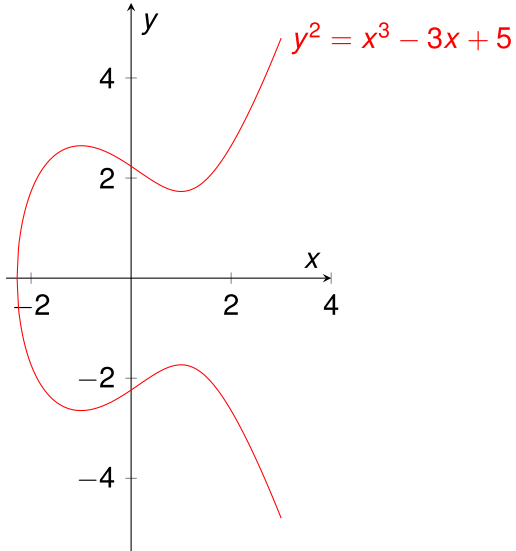
Elliptic curves

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

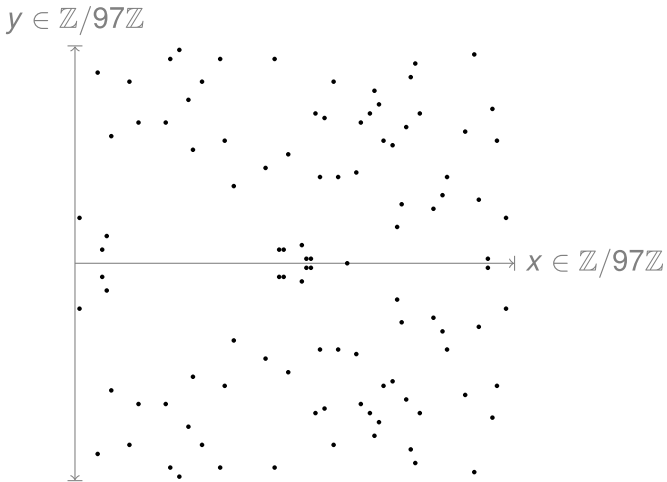
$$y^2 = x^3 + ax + b$$

  for some parameters a and b.
- Set of points *where*?
- Any conditions on a and b?
  - (Yes, e.g., $4a^3 + 27b^2 \neq 0$ but we won't worry about this here)

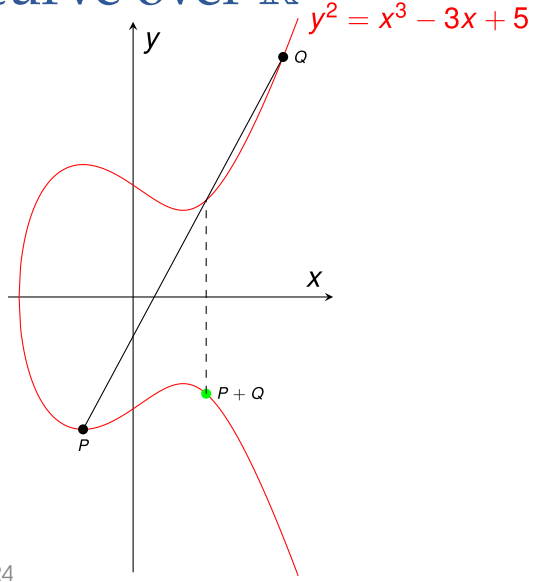
An elliptic curve over $\mathbb{R}$



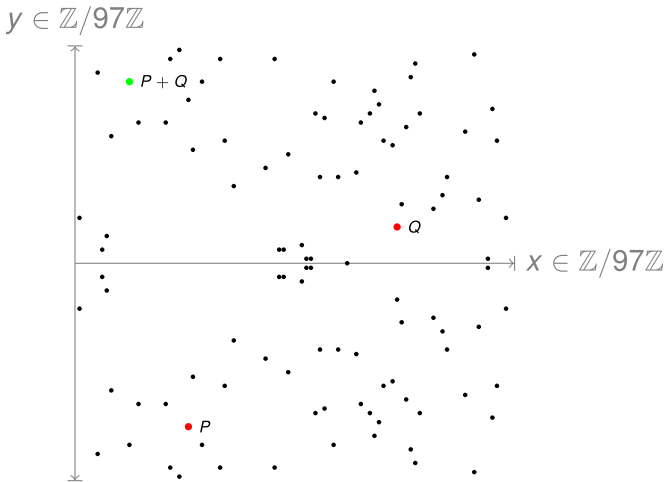
And over $\mathbb{Z}/97\mathbb{Z}$



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 both on that line and on the curve:

$$(mx + d)^2 = x^3 + ax + b$$
$$0 = x^3 - m^2x^2 + \cdots$$

But the formulas are still the same!

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

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

So

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

Compute R? Just one division (get m); a few $\times$ and $+$.

And then?

- Deal with edge cases: elliptic curve becomes a group!
  - What? Why? . . . Magic! (AKA lots of maths I haven't 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.
- Turns out sequence $G, 2G, 3G, \ldots$ looks pretty random!

Use for Diffie-Hellman key exchange

- All parties agree on large prime p, some elliptic curve over $\mathbb{Z}/p\mathbb{Z}$, & some generator G of order n on the curve.
  - Here "all parties" as much as "all Facebook users" or "Amazon customers" (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 & B 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 versus RSA

- ECC needs much smaller key sizes.
  - Preventing attacks using fewer than 2^{128} bit operations requires ECC key of 256 bits (384 usually used) but RSA key of 3072 bits.
- ECC arithmetic is really simple and good choices of p can even accelerate "modulo p" operation.
- ECC is generally superior to RSA. But . . .
  - ECC parameters p, a and b (lesser extent G & 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.
  - Both ECC and RSA are vulnerable to quantum attack.