

Disclaimer: *This lecture is a very brief introduction to cryptography, the goal being to show some practical ramifications of the question of whether P equals NP , and to get a feeling for why such a tremendous importance is attached to that question. The presentation here is simplified, and the definitions are not the “correct” ones, but are meant to capture the essential ideas. None of this material will be on the exam.*

Public-key cryptosystems

- Consider a scenario where you wish to exchange messages with a correspondent, in such a way that no eavesdropper can understand the messages.
- A traditional way to achieve this is called a “private-key” cryptosystem: you and your correspondent agree on a secret key K . To encode a message M you compute $C = f(M, K)$. Your correspondent decodes C by computing $M = g(C, K)$.
- The security of the system is dependent on keeping K secret. But, the correspondents must initially agree on a key. To do this, they must rely on physical security, e.g. meeting face-to-face in private.
- This fact makes private-key cryptosystems less than useful for certain applications: for example, if one party is an e-business that wishes to allow customers to securely send their credit card information, it is not feasible to arrange a face-to-face meeting with every potential customer to decide on a secret key.
- A public-key cryptosystem eliminates this difficulty. The idea is that one party (in the above example, the e-business) chooses a public key K and a secret key L . The public key is made available to the world. To encode a message M , the customer computes

$$C = f(M, K)$$

To decode C , the company computes

$$M = g(C, L)$$

- Consider particular, fixed keys K and L . Let f_K be the function f when its second argument is K , i.e. $f_K(M) = f(M, K)$, and similarly let g_L be the function g restricted to the case when its second argument is L , so that $g_L(C) = g(C, L)$. We will assume that f_K is one-to-one, so that it has an inverse function; and also that it is roughly length-preserving: the length of $f_K(x)$ is polynomial in the length of x . For every message M ,

$$g_L(f_K(M)) = M$$

i.e. g_L is the inverse of f_K , which we denote f_K^{-1} .

- For the system to be both secure and applicable, we need the following to be true.
 1. f_K must be efficiently computable (so that the customer can efficiently encode a message).
 2. f_K^{-1} , the inverse of f_K , must **not** be efficiently computable (so that an eavesdropper cannot efficiently decode a message).
 3. If L is known, then f_K^{-1} must be efficiently computable (so that the company can efficiently decode messages).
- As usual, we take our definition of “efficient” to be “polynomial time”.
- A function like f_K is called a one-way trapdoor function. The label “one-way” refers to the property that it is efficiently computable but not efficiently invertible, while “trapdoor” refers to the property that there is a key L that allows efficient inverting of the function.
- Clearly, in order for secure and useful public-key cryptosystems to exist, we need a one-way trapdoor function.
- A function that just satisfies the first two conditions (i.e. efficiently computable but not efficiently invertible) is called a one-way function¹
- If one-way functions do not exist, then one-way trapdoor functions do not exist, and so (secure/useful) public-key cryptosystems do not exist.
- **Fact:** If $\mathbf{P} = \mathbf{NP}$, then one-way functions do not exist.
- **Proof:** Assume $\mathbf{P} = \mathbf{NP}$, and let f be a function that is computable in polynomial time. We will describe how to compute f^{-1} in polynomial time. Let L be the following language:

$$L = \{\langle x, y \rangle \mid f^{-1}(y) \leq x\}$$

where \leq refers to the lexicographic (i.e. dictionary) ordering. Then $L \in \mathbf{NP}$: a non-deterministic algorithm for L , on input $\langle x, y \rangle$, guesses z and checks that $f(z) = y$ and $z \leq x$. Since $\mathbf{P} = \mathbf{NP}$ by assumption, we have that $L \in \mathbf{P}$. Let M be a polynomial time Turing machine that decides L . We can use M to compute $f^{-1}(y)$, by using M to do a binary search on values of x until we find x such that $x = f^{-1}(y)$. The number of steps required is linear in the length of x , and we have assumed that the length of x is polynomial in the length of $f(x) = y$, so this takes polynomial time. Since f can be inverted efficiently, it is not a one-way function.

¹Formally, the definition of “one-way” is slightly more complicated. The main difference is that the function should not only be hard to invert in the worst case, it should be hard to invert “on average”: a randomized, polynomial-time algorithm should not be able to compute the inverse with any significant probability.

- What does this mean? If $\mathbf{P} = \mathbf{NP}$, then secure public-key cryptosystems do not exist, and as a consequence every public-key cryptosystem (in particular those allowing “secure” connections to online services like banks, and electronic payment) is in fact not secure. So there is a lot resting on the assumption that $\mathbf{P} \neq \mathbf{NP}$.
- In fact, even if $\mathbf{P} \neq \mathbf{NP}$, it is possible that one-way functions (in the sense described in the footnote) do not exist. The assumption that one-way functions exist (which is believed to be true), is a stronger assumption than $\mathbf{P} \neq \mathbf{NP}$.

The RSA System

- Let’s consider a particular, and widely used, public-key system called RSA. The system is as follows:
 - Choose two large primes (typically 128 to 512 bits) p and q
 - Set $N = pq$
 - Set $m = (p - 1)(q - 1)$
 - Choose e such that $\gcd(e, m) = 1$
 - Choose d such that $ed \equiv 1 \pmod{m}$
- The public key is the pair (N, e)
- The private key is the pair (N, d)
- To encode a message M : set $C = M^e \pmod{N}$
- To decode C : set $M = C^d \pmod{N}$ (Note: we are assuming M is an integer between 0 and $N - 1$)
- First of all, let’s see that this works, i.e. that decoding is the inverse of encoding. For this we need the following facts.
 - Modular arithmetic:

$$(A + B) \pmod{n} = (A \pmod{n}) + (B \pmod{n})$$

$$(A \cdot B) \pmod{n} = (A \pmod{n}) \cdot (B \pmod{n})$$
 - If $\gcd(p, q) = 1$ and $A \pmod{p} = A \pmod{q} = r$, then $A \pmod{pq} = r$.
 - **Fermat’s Little Theorem:** Let p be a prime, and let $a \pmod{p} \neq 0$. Then $a^{p-1} \pmod{p} = 1$.

- Now we need to show that if $C = M^e \bmod N$ and $M' = C^d \bmod N$ then $M' = M$.

$$\begin{aligned}
 M' &= C^d \bmod N \\
 &= M^{ed} \bmod N \\
 &= M^{km+1} \bmod N \quad \text{for some integer } k, \text{ since } ed \bmod m = 1 \\
 &= M \cdot M^{km} \bmod N
 \end{aligned}$$

- Since $M^{km} = (M^{k(p-1)})^{(q-1)}$, by Fermat's little theorem we have $M^{km} \bmod q = 1$.
- Similarly, since $M^{km} = (M^{k(q-1)})^{p-1}$, by Fermat's little theorem we have $M^{km} \bmod p = 1$.
- As $\gcd(p, q) = 1$ and $N = pq$, we have $M^{km} \bmod N = 1$.
- So continuing our derivation, we have

$$\begin{aligned}
 M' &= M \cdot M^{km} \bmod N \\
 &= M \cdot 1 \bmod N \\
 &= M
 \end{aligned}$$

- Now consider an eavesdropper who wishes to decode C , but only knows the public key (N, e) .
- If the eavesdropper could factor N into its components p and q , then he/she could follow the same procedure described above to compute d , and use (N, d) to decode C . So the (presumed) security of RSA rests on the assumption that factoring N cannot be done in polynomial time.
- Consider the following language.

$$\text{FACTOR} = \{\langle n, k \rangle \mid n \text{ has a factor } f \text{ satisfying } 2 \leq f \leq k\}$$

It is easy to see that FACTOR is in **NP**. It is also not hard to see that if FACTOR is in **P**, then finding a factor of a number can be done in polynomial time; and if a factor can be found in polynomial time, then *all* prime factors can be found in polynomial time (recall that every number has a unique representation as a product of primes)

- So if FACTOR is in **P**, then RSA is not secure.
- Clearly, since RSA is used in practice, most people believe that FACTOR is not in **P**. Why would people believe this? One good reason would be if FACTOR was **NP**-complete. However, *this is not known to be the case*, and it is generally believed that FACTOR is actually not **NP**-complete.

- What is believed is that FACTOR is neither **NP**-complete, nor in **P**. The reason for this belief (some might call it “hope”) is that, as of yet (and despite significant effort), no polynomial time algorithm for factorization has been found.
- However, the closely related problem PRIMES – the problem of determining whether a number has a proper factor – was recently shown to be in **P**, so this may not be the end of the story.