CMSC 657: Introduction to Quantum Information Processing
Lecture 10

Instructor: Daniel Gottesman

Fall 2024

1 Classical Cryptography

1.1 One-Time Pad

The simplest and most secure encryption protocol is the *one-time pad*. The one-time pad is an example of a *private-key cryptosystem*. In a private key cryptosystem, Alice and Bob share a secret key k (the same for both of them) which is unknown to Eve. This is basically the only limitation on Eve for the one-time pad, although we also assume that Alice and Bob have private computers to perform encryption and decryption without Eve seeing. In the one-time pad, k is just a string of uniformly random bits as long as the message (which has also been converted to a string of bits).

- **Encryption:** Given message m, the ciphertext is $e = m \oplus k$, the bitwise XOR of k and m.
- **Decryption:** $e \mapsto e \oplus k = (m \oplus k) \oplus k = m$.

In the one-time pad, the ciphertext is completely random. Every bit string is equally likely from Eve's point of view regardless of the message. Thus, she has no information about the message. Note that the key really needs to be completely random from Eve's point of view; otherwise she can get a little bit of information about the message. This also means that you can use the key only once and then you must discard it and use a new set of random bits for the next message. There was a case during the Cold War in which Soviet spies were accidentally using one-time pad keys twice, and this enabled the NSA to break a number of their messages.

This means setting up the key for a one-time pad is rather difficult. It requires a trusted courier or prior physical meeting to set up the initial secret key, and the key will run out and need to be renewed if you send a lot of messages or long messages. Therefore, one-time pads are only normally used for the applications with the highest security needs. Later in the course we will see quantum key distribution (QKD), which helps with establishing the secret key.

For most purposes, we want cryptosystems with shorter and reusable keys. To make this secure, we need to make an additional assumption about Eve's limitations. Usually we assume that Eve has limited computational power, and in particular, we might assume that she can only perform polynomial-time computations. (You can base cryptography on other kinds of assumptions, however.) Note that this is a theoretical cryptography approach: Practical cryptography cares about the actual amount of time needed, not the scaling for large systems, and constant factors can matter a lot. Moreover, since we are not good at proving computational problems are actually hard, we have to resort to relying on educated guesses about which problems seem to be hard. This leads to private-key cryptosystems like AES. Like the one-time pad, Alice and Bob share a secret key which is not known to Eve, but unlike the one-time pad, it is much shorter than the message and can be reused.

Still, this is a bit inconvenient, since Alice and Bob have to establish that secret key somehow and they need to protect it from Eve. There is no way to talk to someone with whom you have had no prior direct contact. For that, we use a *public-key cryptosystem*. In a public-key cryptosystem, Bob generates a pair of

keys, a *private key* d, known only to him, and a *public key* e, which can be published or given out to people that might want to send messages to Bob. d and e are not the same but must be generated together as a pair. e is then used by the encryption algorithm to generate the ciphertext, whereas d is used by the decryption algorithm. Since potentially anyone can know e, anyone can encrypt messages to send to Bob, but since he is the only one that knows d, only he can read those messages. Again, the security of a public-key cryptosystem is usually based on some sort of computational assumption about Eve.

1.2 RSA

RSA (named for its inventors Rivest, Shamir, and Adleman) is an example of a public key cryptosystem. It works as follows:

- **Key generation:** Bob choose 2 large prime numbers p and q. Let $N = pq$. Bob also chooses $e < N$ and finds d such that $x^{ed} = x \bmod N$ for all x. (Discussion of how to do this is below.)
- **Private key:** Bob's private key will be d.
- **Public key:** The public key will be the pair (N, e).
- **Encryption:** Given message $x < N$, the ciphertext is $y = x^e \bmod N$. Larger messages can be broken up into blocks of $\log N$ bits; each block is encrypted separately.
- **Decryption:** Given ciphertext y, the decoded message is $x' = y^d \bmod N$. Note that by the relationship of e and d, the decoded message

$$x' = y^d \bmod N = (x^e)^d \bmod N = x \bmod N = x. \tag{1}$$

Key generation relies on Euler's theorem:

Theorem 1 (Euler's theorem)**.** *For all $x < N$, $x^{\varphi(N)} = 1 \bmod N$. Here $\varphi(N)$ is Euler's* totient *function, the number of natural numbers less than N and relatively prime to N. There is a straightforward formula for $\varphi(N)$ in terms of its factors; for instance $\varphi(pq) = (p-1)(q-1)$ for prime p, q.*

Once Bob selects p and q, he can easily calculate $\varphi(N)$. He chooses e to be relatively prime to $\varphi(N)$ and then can use Euclid's algorithm to find a d such that $de = 1 \bmod \varphi(N)$. Once we have that, we know that $de = 1 + c\varphi(N)$. Therefore,

$$x^{ed} = x \cdot x^{c\varphi N} = x \cdot 1^c \bmod N = x \bmod N, \tag{2}$$

using Euler's theorem.

Algorithm 1 (Euclid's Algorithm)**.**
Input: *Two positive integers m and n*
Output: *Two integers a and b such that $am + bn = \gcd(m, n)$.*

(Bob will let $m = \varphi(N)$ and $n = e$, and then let d be the output b. We don't care about a.)
Euclid's algorithm works as follows:

1. Let $i = 1$, $x_0 = \max(m, n)$, $x_1 = \min(m, n)$.
2. If $m > n$, let $a_0 = b_1 = 1$ and $b_0 = a_1 = 0$. Otherwise, let $a_0 = b_1 = 0$ and $b_0 = a_1 = 1$.
3. While $x_i > 0$, do:
   (a) Let $r_i = \lfloor x_{i-1}/x_i \rfloor$ and let $x_{i+1} = x_{i-1} - r_i x_i$
   (b) Let $a_{i+1} = a_{i-1} - r_i a_i$ and $b_{i+1} = b_{i-1} - r_i b_i$
   (c) Increase i by 1

4. Output $a = a_{i-1}$ and $b = b_{i-1}$.

Euclid's algorithm tells us the gcd of m and n and also how to decompose the gcd into an integer linear combination of m and n. The run time is polynomial in $\log m$ and $\log n$.

Proof. The definitions of r_i and x_{i+1} ensure that $x_{i-1} = r_i x_i + x_{i+1}$, with r_i a positive integer and $0 \leq x_{i+1} < x_i$. Thus, x_i decreases with each iteration. Moreover, by induction $g = \gcd(m, n)$ divides x_i for all i: It does so for $i = 0$ and $i = 1$, and it divides both terms on the RHS of the formula for x_{i+1}. Thus $x_i \geq g$ unless $x_i = 0$, but if $x_i = g$, then x_{i-1}/x_i is an integer and $x_{i+1} = 0$. Therefore, x_i decreases until it reaches g and then the algorithm stops.

Also, note that $x_i = a_i m + b_i n$. This is true for $i = 0$ and $i = 1$ by the initialization conditions. For greater i, we apply induction:

$$\begin{aligned} x_{i+1} &= x_{i-1} - r_i x_i && (3)\\ &= a_{i-1}m + b_{i-1}n - r_i(a_i m + b_i n) && (4)\\ &= (a_{i-1} - r_i a_i)m + (b_{i-1} - r_i b_i)n && (5)\\ &= a_{i+1}m + b_{i+1}n. && (6) \end{aligned}$$

When we terminate, $x_i = 0$, which means that $x_{i-1} = g$. We therefore have

$$g = x_{i-1} = am + bn, \tag{7}$$

as desired.

Finally, let us analyze the run time briefly. I claim that $x_{i+2} \leq x_i/2$. If $x_{i+1} \leq x_i/2$, this is certainly true, and if $x_{i+1} > x_i/2$, that means that $r_{i+1} = 1$, so $x_{i+2} = x_i - x_{i+1} \leq x_i/2$. Therefore, every 2 steps, x_i increases by at least a factor of 2, which means the algorithm terminates after $O(\log \max(m, n))$ iterations. Each iteration involves some arithmetic, which can be done in time polynomial in the number of digits in the numbers, so $\text{poly}(\log \max(m, n))$. □

Bob can therefore generate his key pair using a time polynomial in $\log N$, but what about encryption and decryption? This involves modular exponentiation, taking x^e, for instance. You might try to do this by calculating x, then x^2, x^3, x^4, etc., multiplying successively by x. However, since e and d can be $O(N)$, this approach takes a very long time, $O(N)$ time. We can do much faster.

Modular Exponention: We can pre-calculate $x^2, x^4, \ldots, x^{2^{n-1}}$ by repeated squaring, where $n = \lceil \log N \rceil$. All of them are modulo N, which means they don't blow up in size. We can find each of these from the last by one additional multiplication. We thus need a total of $n - 1$ multiplications to get all these answers, and each can be done in time $\text{poly}(\log N)$.

To calculate x^e, we decompose $e = \sum_i e_i 2^{n-1-i}$, its binary representation. Then $x^e = \prod_{i|e_i=1} x^{2^{n-i-1}}$. That is, we take the product over just those exponents with powers of 2 that appear in the binary expansion of e. There are again at most $n - 1$ multiplications needed here. The total time for the algorithm is thus $O(n)$ times the time needed for one multiplication. Long multiplication takes $O(n^2)$ time, giving a total $O(n^3)$ time for the whole modular exponentiation. However, there are faster multiplication algorithms, the fastest known one having asymptotic time complexity $O(n \log n \log \log n)$.

For secure implementation of RSA, there is further art that must go into picking p, q, and e to make sure that this is a hard example of the problem. In addition, rather than directly encrypting the message x, Alice chooses a random number for x and then encrypts that using RSA. The random number x is then used as the key for a more secure private-key cryptosystem. One reason to do this is to avoid the possibility that Eve has previously narrowed the message down to a small number of possibilities. Without the random number, Eve can easily try encrypting all of the possible messages and comparing to the ciphertext Alice sent.

Factoring: Given N, find $p, q > 1$ such that $N = pq$ if they exist. Normally we consider N which is a product of two primes, so p and q should both be prime, but this is not necessary. Note that this problem can be reduced to a decision problem: Given N, M, does there exist a non-trivial factor of N less than M? (Then do a binary search to identify the exact factors.) This decision version of factoring is in NP, the witness being a factor less than M. It is also in co-NP, which means there is a witness if the answer is no. Again, the factors can be the witness, along with an algorithm to test that they are actually prime. Thus, we believe that factoring is not NP-complete.

Note that multiplication (given p, q, find $N = pq$) is easy (polynomial time in $\log p + \log q$), but no efficient (i.e., poly($\log N$)) classical factoring algorithm is known. The best known classical factoring algorithm uses time about $\exp(\log N)^{1/3}$. There has been considerable work devoted to finding good factoring algorithms because if you can factor, you can break RSA: Given N (which is part of the public key), you factor to find p and q and then you know $\varphi(pq)$ as well. Euclid's algorithm again gives you d from e.

Period Finding: Let $f : \mathbb{Z} \to \mathbb{Z}$ be a periodic function with period r that does not otherwise repeat values. That is, r is the smallest positive integer such that $f(a + r) = f(a)$ and $f(a) \neq f(b)$ unless $r|b - a$. Find r.

Claim 1. *Factoring $N = pq$ reduces to finding the period of the function $f_x(a) = x^a \mod N$ for arbitrary x.*

The goal is going to be to find a value of x which has even order r. A random x has even order with constant probability (I will not prove this), so we can pick x randomly and use period finding to find its order. After a constant number of tries, we are likely to find an x with even order.

Once we have such an x, compute $y = x^{r/2}$. Then $y^2 = 1 \bmod N$, so

$$y^2 - 1 = (y + 1)(y - 1) = kN. \tag{8}$$

Either $N|y + 1$, $N|y - 1$, or $p|y + 1$ and $q|y - 1$. We know $y \neq 1 \bmod N$, or the order of x would actually be $r/2$ or less. This eliminates the possibility $N|y - 1$.

We also wish to eliminate $N|y + 1$, which is equivalent to saying $y = -1 \bmod N$. This can happen sometimes, but only with constant (less than 1) probability. Thus, by choosing x randomly, we can, with constant probability per attempt, find y such that $p|y + 1$ and $q|y - 1$.

In that case, let $p = \gcd(y + 1, N)$, $q = \gcd(y - 1, N)$ (again found using Euclid's algorithm). We know that these gcd's are non-trivial (because there is a common factor with $y + 1$ or $y - 1$ and N), and these are in fact factors of N, so they solve our problem.

There is no known efficient classical method of period finding. However, Shor's algorithm provides an efficient *quantum* algorithm.