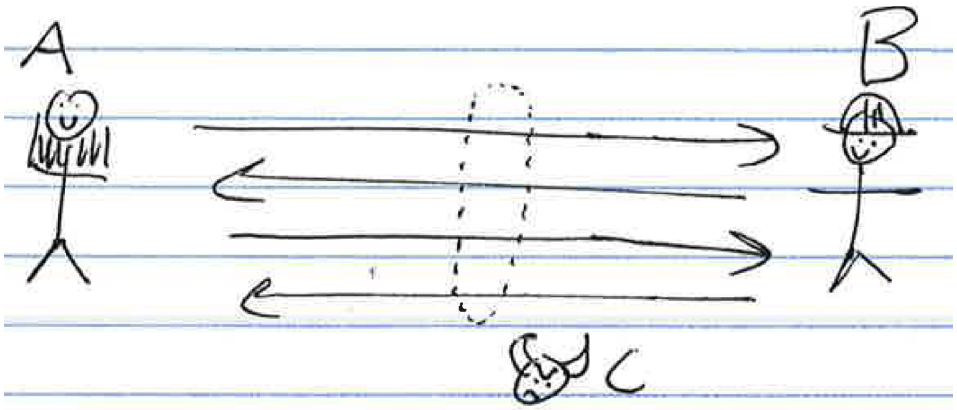
CS 3510 Algorithms 8/31/2023

Lecture 4: Cryptography

Lecturer: Abrahim Ladha *Scribe(s): Himanshu Goyal*

1 Motivation

Cryptography is the science of secrecy. We learn about it in this class (algorithms) because it is all about algorithms for secure communication. Consider the following scenario. You have two parties say Alice, Bob. They want to communicate and pass messages to each other with a goal that eavesdropper has a slight idea about what they are communicating about. There is a third party, the adversary, perhaps denoted as Charlie. Charlie can listen to what Alice and Bob send to each other. Is there an algorithm, or a protocol mathematically, in which Alice and Bob could agree to use to stop Charlie form listening their secret communication. This is not such a contrived scenario. However, For instance, Imagine You and Amazon could be parties Alice and Bob respectively. This protocol should stop someone from stealing your credit card info. There are numerous such scenario in our daily life. Much of the financial sector relies on cryptography. In this lecture, we will try to define methodologies to solve this problem.

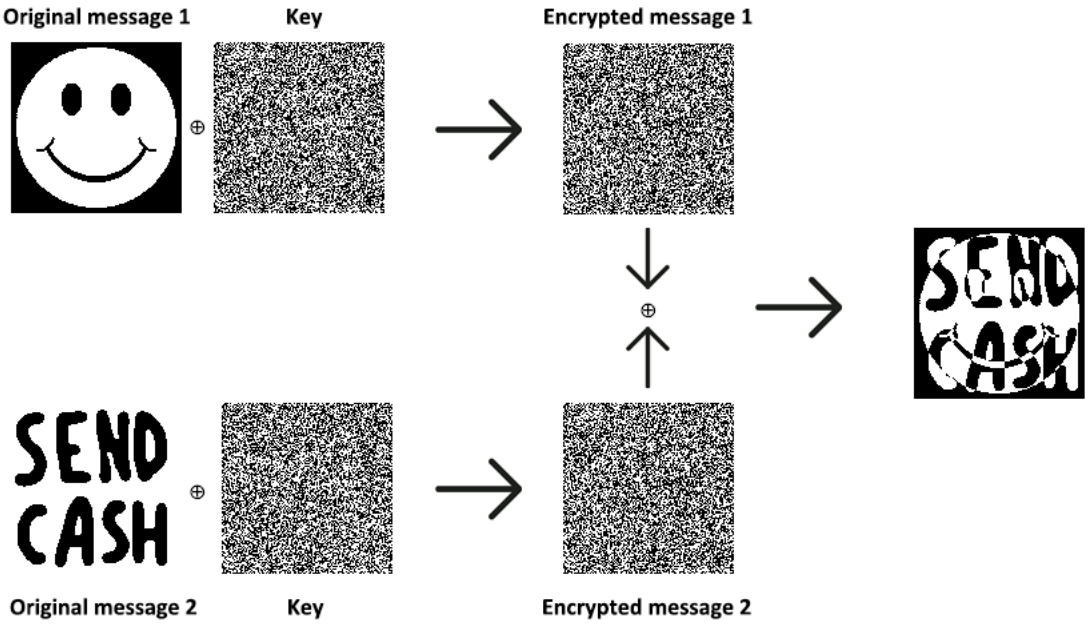


1.1 One Time Pad (OTP)

Let us see one of the one such protocol among many others, called One Time Pad (OTP).

- Communication
  - Suppose that Alice wants to send Bob a message m of length $|m| = n$. Alice and Bob already have a shared Secret Key $|s| = n$.
  - Alice computes $m \oplus s$, and sends it to Bob.
  - Bob decodes the message by doing $(m \oplus s) \oplus s = m$ and learns Alice's message m.

- **Security:** From this exchange, Charlie learns $m \oplus s$ from eavesdropping which provides little to no information. If s is completely random, then $m \oplus s$ looks as random as s, so Charlie can learn nothing about m from looking at $m \oplus s$.
- This scheme works fast but has three immediate issues:-
  - How do Alice, and Bob already know the secret s? They can't just directly share s, because Charlie would find it out.
  - The second problem is that $|s|$ has to be as long as $|m|$. But we want small keys and long messages. We don't want to store gigabytes of keys to send lots of info.
  - The biggest problem is reuse. It's called one time pad for a reason. Suppose, Bob after learning m, wants to reply with message m', so he sends $m' \oplus s$. Now, Charlie learns $(m \oplus s)$ and $(m' \oplus s)$. He may compute $(m \oplus s) \oplus (m' \oplus s) = m \oplus m'$, which provides extra information to the adversary (charlie) then anticipated information leakage. Generally, It is assumed that user messages does not have high entropy. For instance, most of our website passwords have high co-relation in between them.



1.2 Symmetric Key Encryption

- A symmetric encryption if one of the directions and it comprises of a *Generator, Encryptor, and Decrypter.*

  $SE = (G, E, D)$ (where G is the generator, E is the encryptor, and D is the decrypter)

  - $k \leftarrow G$ (the generator generates keys; k denotes a key)

- $c \leftarrow E_k(m)$ (the encryptor with a given key k takes a message m and creates a ciphertext c)
- $m \leftarrow D_k(c)$ (the decrypter with a given key k takes a ciphertext c and gives the message m)

- **Note:** Here k can be shorter than m (although the key will likely be much smaller than the OTP), and given only cipher text c to Charlie, it is asymptotically infeasible to compute m from c without the knowledge k.

- Communication:
  - Both Alice and Bob agree on a key, k, from G.
  - Alice receives c from $E_k(m)$ using the encryptor.
  - Alice transmits c (i.e, $E_k(m)$) over the wire to Bob. Charlie is also listening to this channel.
  - Bob decodes the message with $m = D_k(c)$.

- In this scheme, Charlie sees only the ciphertext c, i.e., $E_k(m)$, which will have high entropy, so he would learn little. There is an instantiation of this scheme commonly known as Advanced Encryption Standard (AES) today. The problem which it still *doesn't* solve is of having a common preshared secret key.

1.3 Asymmetric Cryptography

- A public key or asymmetric cryptosystem is comprised of three algorithms (G, E, D) and uses two keys pk, sk. In this, encryption is done with the public keys, however the decryption is done with secret keys. Every party participating in communication have their own set of (pk, sk). Parties tend to know only pk of every other party, keeping sk secret with themselves. First we define the protocol and later define the security.
  - $(pk, sk) \leftarrow G$ (generates a public key pk and secret key sk)
  - $c \leftarrow E_{pk}(m)$
  - $m \leftarrow D_{sk}(E_{pk}(m))$

- Communication
  - Bob generates a public key pk_B and a secret key sk_B.
  - He broadcasts the public key pk_B to Alice (or others).
  - Alice encrypts her message m using public key of Bob as $E_{pk_B}(m)$, and sends it to Bob.
  - Bob decodes the intended message m using its own secret key as $D_{sk_B}(E_{pk_B}(m))$.

- Problems
  - An integrity issue can arise if Charlie is able to repeat a message of Alice and wants to be assumed as Alice to Bob.

  - Another issue could arise if there are a small number of options which are sent often. Say a vote is taking place with two options and one message is sent 60% of the time and another message is sent 40% of the time. This is fixed by concatentation of the message with some randomness, that is discarded after decryption.

- Charlie will see pk_B and $E_{pk_B}(m)$. From just this information, we want to ensure that he learns nothing.

1.4 Semantic Security

- One time pad achieves perfect secrecy (or information theoretic security), which states that there is no positive correlation between the information of communicated ciphertext and the information learned about the message.

- In reality, however, we can assume that eavesdroppers like Charlie are bounded by realistic computational limits and therefore belong to the class of Probabilistic Polynomial Time (PPT) adversaries. So, now we will see security for PPT adversaries. It is also commonly referred as computation security.

- If a cryptographic scheme has a key k that is n-bits long, the probability of simply guessing the key is $Pr[\textit{guessing } k] = 2^{-n}$.

- We say a cryptographic scheme is secure if for all Probabilistic Polynomial Time (PPT) adversaries, $Pr[\textit{learn anything about } m] < 2^{-n}$.

- This essentially means the system is secure if the best option for someone without the key to decode the ciphertext is to *guess a key.*

- Although many people believe this holds true for cryptographic systems, it is *hard to prove*, and doing so would cause many major breakthroughs in complexity theory.

Most of cryptography algorithms heavily uses ideas from number theory. So, let us try to see some of related concepts before defining cryptographic algorithms.

1.5 Greatest common divisor (gcd)

Now we have described an ideal asymmetric cryptosystem. Let's actually give a protocol for it. First we need to develop tools form number theory to get there.gcd(a, b) computes the greatest common divisor of a, b. For example, $gcd(105, 30) = gcd(3 \cdot 5 \cdot 7, 2 \cdot 3 \cdot 5) = 3 \cdot 5$. The following is an easy divide and conquer algorithm discovered long ago by Euclid to calculate gcd of any two numbers.

```
function gcd(a, b)
if b = 0
    return a
else
    return gcd(b, a mod b)
```

- Proof of Correctness: We can prove correctness by showing $gcd(a, b) = gcd(a, a - b)$. Let $d = gcd(a, b)$. If $d|a$ (a divides d), and $d|b$ (b divides d), then $a = dk$, and similarly $b = dl$. So, $a - b = dk - dl = d(k - l)$. Therefore, d is a factor of $a - b$, hence $d|(a - b)$. So, $d|gcd(b, a - b)$. Let $gcd(b, a - b) = d'$. So, $d'|b$, $d'|(a - b)$. So $d'|(a - b) + b = a$. So $d'|a$ and $d'|b \implies d'|gcd(a, b) \implies d' = d$. Since these two numbers divide each other, they must be equal. Easy!!

- Runtime: Since $a \mod b < \frac{a}{2}$, every two recursive calls shrinks our answer by one bit. Therefore n calls, each performing an $\mod N$ bit remainder[1] in the $O(n^2)$, resulting in total time $O(n^3)$.

- We can use the stack trace execution[2] of the euclidean algorithm to compute modular inverse of an element. The inverse of a is a^{-1} such that $aa^{-1} \equiv 1 \pmod N$. For example, $3^{-1} \equiv 2 \pmod 5$, since $3 \cdot 2 = 6 \pmod 5 \equiv 1$. Similarly, $2^{-1} \equiv 3 \pmod 5$.

- The necessary property for the existence of inverse of an element a, is

$$a^{-1} exists \pmod N) \Leftrightarrow gcd(a, N) = 1$$

  In words, If a is relatively prime to N, then it has an inverse. *In general, if N is prime, then all elements less than it has an inverse.* For fun, try proving it!

1.6 Fermat's little theorem

We prove the following insanely useful fact. If p is prime and $1 \leq a < p$, then

$$a^{p-1} \equiv 1 \pmod p$$

This is called Fermat's little theorem.

Proof of Correctness:

- Let $S = \{1, 2, \ldots, p - 1\}$, and $aS = \{a, 2a, \ldots, (p - 1)a \mod p\}$. First we show $\pmod p$ that $S = aS$. It is sufficient for us to show the equality if we can show the following two:

  1. None of the elements in aS is 0, i.e. $0 \notin aS$.
  2. All the elements in aS are distinct.

- Suppose $ai = 0 \mod p$, then $a^{-1}ai \equiv a^{-1}0 \pmod p \implies i \equiv 0 \mod p$, but $i \geq 1$, contradiction

- For 2, let us try to prove by contradictions. Suppose aS contains two equal elements, i.e $a_i \equiv a_j \pmod p$. Since *gcd(a,p)=1*, therefore there exists an a^{-1} for a.

[1] We didn't cover this in class, but integer division with remainder, like multiplication can be done in $O(n^2)$ time. It can actually be done in not just linear time, O(n), but real time, n. No constants.

[2] We won't cover the extended euclidean algorithm, but you may supose that modular inverses may be computed easily in polytime given the modulus.

$$ai \equiv aj \pmod{p} \implies a^{-1}ai \equiv a^{-1}aj \implies i = j, \text{ a contradiction.}$$

- Since $aS = S$, we may product the elements of both sets. $\prod S = 1.2.3....(p-1) = (p-1)!$. Similarly, $\prod aS = a.(2a).(3a)....a(p-1) \equiv a^{p-1}(p-1)!$. So, $(p-1)! \equiv a^{p-1}(p-1)!$. Since *gcd((p-1)!, p) = 1*, then we see that $a^{p-1} \equiv 1 \pmod{p}$.
- **Corollary:** if p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p. Then,
$$a^p \equiv a \pmod{p}.$$

1.7 Euler Theorem

It is a generalisation which we won't prove. For any integer a co-prime to N, Euler's Totient theorem states that:

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

$\varphi(n)$ is the number of positive integers less than n that are relatively prime to n.

- For a prime number p, $\varphi(p) = p - 1$.
- For two distinct prime numbers p and q, $\varphi(pq) = (p-1)(q-1)$ (note: this does not work for p^2).
- It should be clear that Fermat's little theorem is a special case of Euler's theorem.

1.8 Security hardness assumptions

- We believe (a proof is still unknown) that the following problems are computationally infeasible:
  1. **Factoring:** Given $N = pq$, determine p, q
  2. **Discrete Log:** Given y, $x^y \pmod{N}$, and N, determine x.
  3. **Diffie-Hellman:** Given g^x, g^y, g, N, determining $g^{xy \bmod N}$.
- All asymmetric cryptosystems relies on these unproven assumptions that we stronly believe that hard to solve. Therefore, currently the security is, unfortunately conditional. You can design any other protocol using these assumptions, and you may assume these problems are hard while proving security of your protocol.
- It is believed that demonstrating the security of Public Key Cryptography poses a formidable challenge and would imply that $\mathbf{P} \neq \mathbf{NP}$.
- However, the uncertainty lies in whether these problems are indeed hard, as we are yet to determine if $\mathbf{P} = \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{NP}$.
- Factoring, in particular, is interesting because we can efficiently factor numbers with many small factors. To make factoring hard, cryptographic schemes often rely on the use of two large prime factors.

1.9 RSA

- Communication
  - Bob's Actions:
    - Bob generates large prime numbers p and q.
    - Bob computes $N = pq$.
    - Bob computes e, a number relatively prime to $(p-1)(q-1)$.
    - Bob computes $d \equiv e^{-1} \pmod{(p-1)(q-1)}$.
    - Public Key: $pk = (N, e)$, Private Key: $sk = d$.
    - Bob broadcasts the public key pk.
  - Alice's Actions:
    - A computes and sends $m^e \bmod N$.
  - Bob's Actions:
    - Bob computes $m \equiv (m^e)^d \bmod N$.
- Correctness

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$
$$ed = k(p-1)(q-1) + 1$$
$$(m^e)^d \equiv (m)^{ed} \pmod{N}$$
$$(me)^d \equiv m^{k(p-1)(q-1)+1} \pmod{N}$$
$$(me)^d \equiv m^{k\varphi(N)+1} \pmod{N}$$
$$(me)^d \equiv m^1(m^{\varphi(N)})^k \pmod{N}$$
$$(me)^d \equiv m1^k \pmod{N}$$
$$(me)^d \equiv m \pmod{N}$$

- Visibility to Charlie: Charlie will only see N, e, and $m^e \bmod N$. By our assumption, he cannot learn m.

1.10 Diffe-Hellman (Common Secret Exchange)

- Alice and Bob agree on prime numbers p and g such that the two sets $(1, 2, \ldots, p-1)$ and $(g^1, g^2, \ldots, g^{p-1}) \bmod p$ are bijective.

Communication

- Alice's Actions:
  - Alice computes x and g^x, then communicates them to Bob.
- Bob's Actions:
  - Bob computes y and g^y, then communicates them to Alice.
- Alice's Actions:
  - Alice computes $g^{xy} \equiv (g^y)^x \pmod{p}$.
- Bob's Actions:
  - Bob computes $g^{xy} \equiv (g^x)^y \pmod{p}$.

Visibility to Charlie

Charlie sees g, p, g^x, and g^y, but cannot compute $g^{xy} \bmod p$.

Common Use

This method is commonly used to establish a symmetric encryption algorithm, which is typically faster and more practical for actual encryption.