CSCI 1515: Applied Cryptography

P. Miao

Spring 2023

These are lecture notes for CSCI 1515: Applied Cryptography taught at Brown University by Peihan Miao in the Spring of 2023.

These notes are taken by Jiahua Chen with gracious help and input from classmates. Please direct any mistakes/errata to me via email, post a thread on Ed, or feel free to pull request or submit an issue to the notes repository (`https://github.com/BrownAppliedCryptography/notes`).

FYI, todo's are marked like this.

Notes last updated February 4, 2023.

Contents

§1 January 26, 2023

§1.1 Introduction

If you find a lot of courses on cryptography online or at universities, you'll find a lot of theoretical content. However, it's helpful for students to get hands-on experience with cryptography:

- How cryptography has been used in practice,
- how cryptography will be used and implemented in the future.

The goal of this course is to

Introduction: Peihan Miao, please refer by Peihan (you will be corrected if you address as professor). Joined Brown last semester, taught a seminar on research topic (Secure Computation). Before that, was in Chicago, and Visa Research even before that. Really loved math and theoretical computer science, started PhD with theoretical computer science. Shifted to applied side of cryptography; it's exciting to see cryptography implemented in practice. However, realized that most crypto courses are very theoretical—there are not a lot of courses that build up advanced applications using the tools that have been set up.

For this course, it will be *much less* about math and proofs, and much more about how you can use these tools to do something more fun. It will be coding heavy, all projects will be implemented in C++ using crypto libraries. If, however, you are interested in the theoretical or mathematical side, you might consider other courses at Brown[1]

§1.2 Course Logistics

The course homepage is at https://brownappliedcryptography.github.io/.

Huge kudos to Nick and Jack for developing a new course from the ground up!

Please view the syllabus here. *If there are differences between this document and the syllabus, the syllabus trumps this document.*

The course is offered in-person in CIT 368, as well as synchronously over Zoom and recorded asynchronously (lectures posted online). You can do either[2] but try to attend online because there will be interaction! Lecture attendance is encouraged!

EdStem will be used for course questions, and **Gradescope** is used for assignments.

[1]CSCI 1510 and Math 1580 are good candidates. 1510 covers cryptographic proofs, and Math 1580 covers cryptography from a number theoretic perspective.

[2]It is a 9AM class...

For assignments: *Project 0* is a warmup for C++. Projects 1 & 2 is to develop secure communication or authentication systems using the underlying cryptographic libraries. The later project, 3, 4 & 5 will be on more advanced topics. The first 2 are more basic that are developed in practice, and the latter ones are more experimental in practice (so this is recent research in applied cryptography). The final project will be a combination of the existing projects or a project entirely new (separate from the earlier projects, but using the same cryptographic primitives).

Projects 1 through 5 will be accompanied with homework assignments (appropriately numbered 1 through 5) that develop a conceptual understanding of the materials.

Projects 1 & 2 are to be done individually, the later projects are done in pairs (you can choose to go solo if you so wish as well). You are encouraged to find partners earlier on to discuss and work with them from the beginning. You are *encouraged* to communicate with your partners on projects 1 & 2 so you gain a conceptual understanding. You should complete your own write-up and code for the first two projects, however.

There is an option to capstone this course, contact Peihan about this. It would also be best to find a partner who is also capstoning this course.

The following is the grading policy:

Type	*Percentage*
Project 0	5%
Projects 1 & 2	20% (10% each)
Projects 3, 4 & 5	45% (15% each)
Homeworks	20% (2% each)
Final Project	20%

You have 6 late days for *projects*, of which at most two can be used on a single project. Additionally, you have 3 late days for *homeworks*, of which at most one can be used on a single homework.

If you're sick, let Peihan know with a Dean's note.

§1.3 What is cryptography?

...or more importantly, what is cryptography used for?

At a high level, *cryptography is a set of techniques that protect information.*

Question. What are some cryptographic techniques used in practice and what does it achieve?

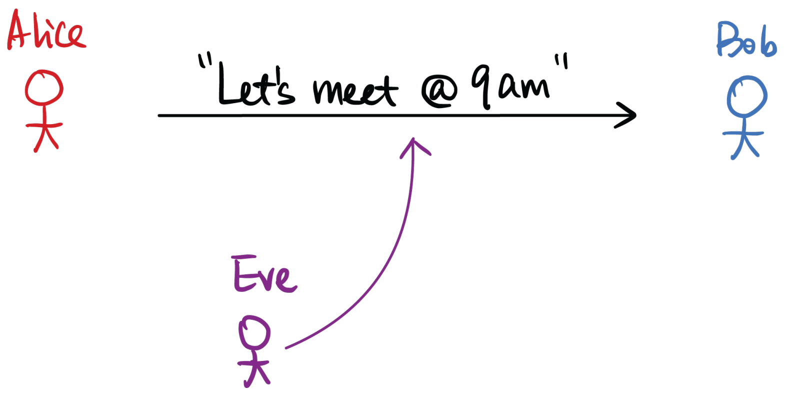
- Used in financial institutions.

  - When you make a purchase, you might not want people to see your bank balance, what else you have purchased, etc.

- Might be used in voting schemes. Making sure that ballots are kept private.
- Messaging systems, recently.
  - End-to-end texting.
- Storing medical records.
  - More generally, any sensitive information.
  - You don't want anyone else getting access to this sensitive information.
- Used in military or war.
  - Historically, it was used a lot in the military for secure communication.
- Authentication systems. Login systems.
  - There is authentication going on that ensures that *only you* can log in.

§1.4 Secure Communication

We'll start with the most primitive of cryptography: *secure communication.*



Assume Alice wants to communicate to Bob "Let's meet at 9am", what are some security guarantees we want?

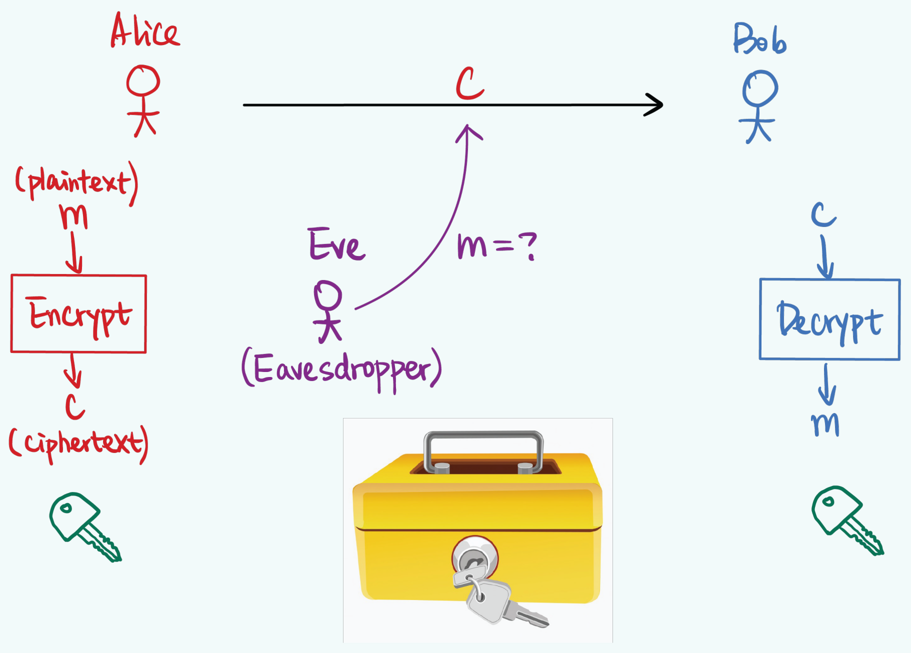
- Eve cannot *see* the message from Alice to Bob.
- Eve cannot *alter* the message from Alice to Bob.

These two guarantees are the most important guarantees! The former is called message secrecy, the latter is called message integrity.

§1.4.1 Message Secrecy

Definition 1.1 (Message Secrecy)
We want cryptography to allow Alice to *encrypt* the message m (which we call *plaintext*) by running an algorithm that produces a *ciphertext* c.

Bob will be able to receive the ciphertext c and run a *decrypt* algorithm to produce the message m again. This is akin to a secure box that Alice locks up, and Bob unlocks, while Eve does not know the message.

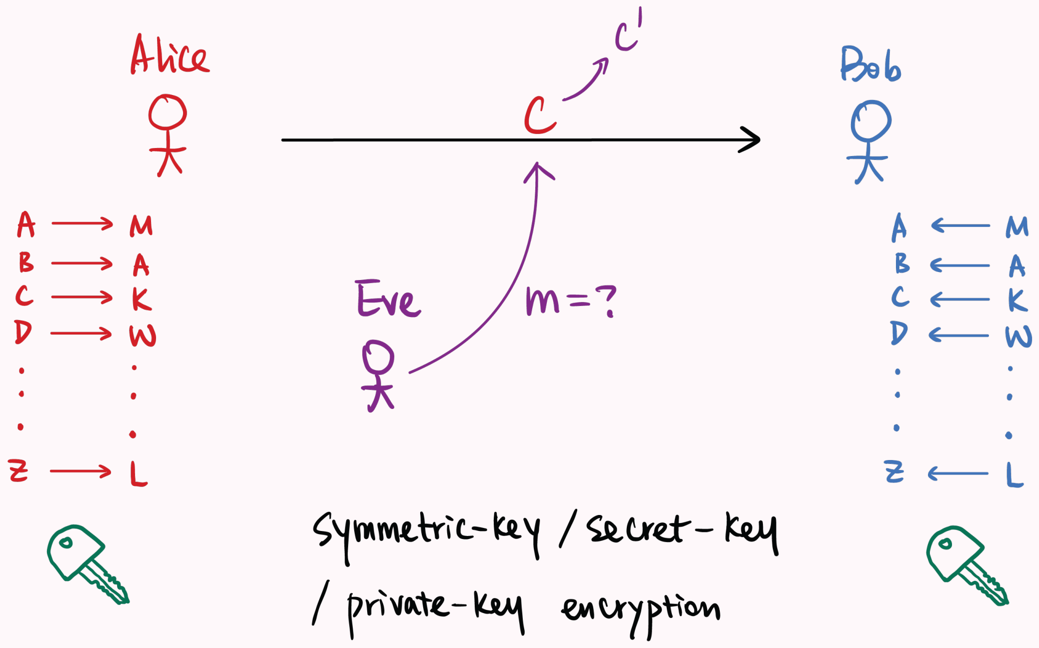


In this model, Eve is a weaker adversary, an *eavesdropper*. Eve can only see the message, not alter it.

Example 1.2 (Substitution Cipher)
The key that Alice and Bob jointly uses is a permutation mapping from $\{A \dots Z\} \rightarrow \{A \dots Z\}$.

This mapping is the *secret key.*

Bob also has the mapping, and takes the inverse of the permutation to retrieve the message.



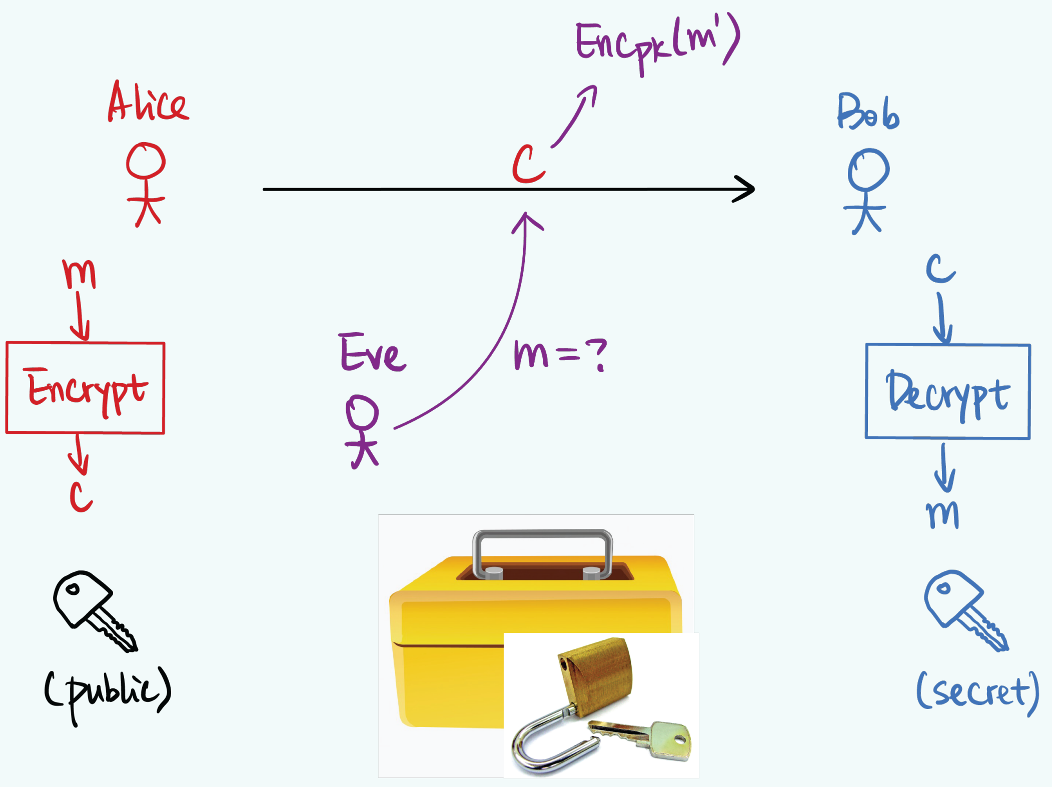
This scheme is not quite secure! *Why?*

You could guess a bunch of vowels and see what words could make up. If you have a long enough message, you can see which letters appear more often. We know that in English, the vowels appear more often; and you can make a lot of guesses.

Remark. This encryption scheme also requires that Alice and Bob meet up in person to exchange this shared private key. Schemes like this are called *symmetric-key, secret-key, or private-key encryption.* They need to meet up first to exchange secret keys.

Definition 1.3 (Public-key Encryption)

There is another primitive that is much more ideal/stronger, public-key encryption. Bob publishes both a *public* key and a *private* key. You can consider a lock where you don't need a key to lock it[3], and only Bob has the key to unlock it.

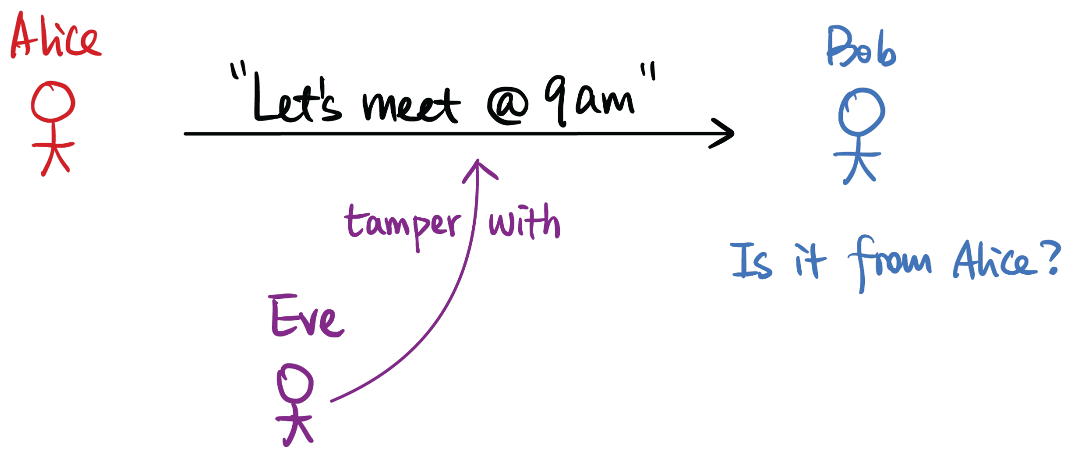


This is seemingly magic! Bob could publish a public key on his homepage, anyone can encrypt using a public key but only Bob can decrypt. *Stay tuned, we will see public-key encryption schemes next lecture!*

§1.4.2 Message Integrity

Alice wants to send a message to Bob again, but Eve is stronger! Eve can now tamper with the message.

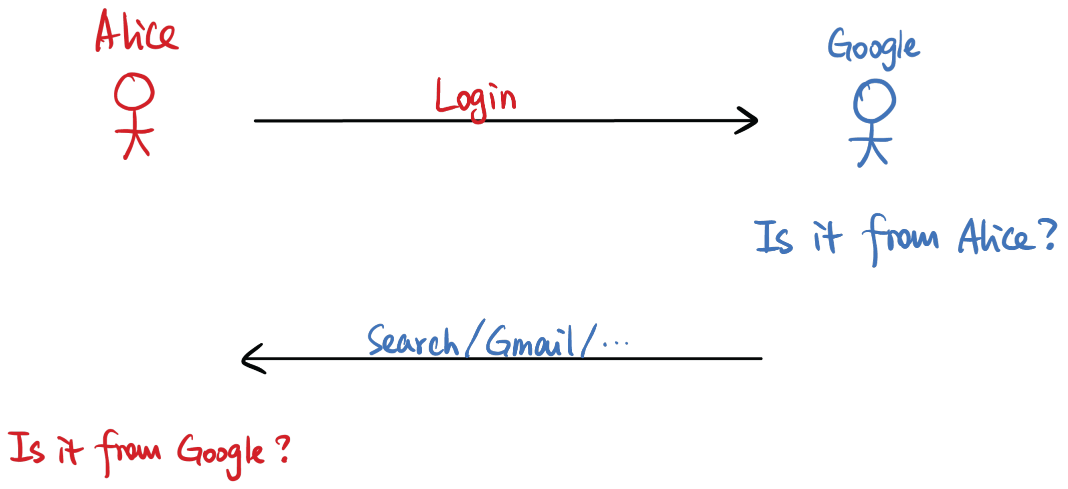
[3]You literally click it closed



Bob wants to ensure that the message *actually* comes from Alice. Does our previous scheme (of encrypting messages) solve this problem? Nope!

Eve can change the ciphertext to something else, they could pretend to be Alice. In secret-key schemes, if Eve figures out the secret-key, they can forge messages from Alice. Even if Eve doesn't know the underlying message, they could still change it to some other ciphertext which might be correlated to the original ciphertext, *without knowing the underlying message*. We'll see how Eve can meaningfully do this in some schemes. Alice could send a message "Let's meet at x AM" and Eve could tamper this to say "Let's meet at $x + 1$ AM."

This is sort of an orthogonal problem to message secrecy. For example, when Alice logs in to Google, Google needs to verify that Alice actually is who she claims to be.



§1.5 Project Overview

0. Warm-up, you will implement some basic cryptographic schemes.

1. Secure Communication: what was just introduced.

2. Secure Authentication: how to authenticate yourself to a server, also mentioned just now.

3. Zero-Knowledge Proofs: we'll use ZKPs to implement a secure voting scheme.

4. Secure Multiparty Computation: we'll implement a way to run any function securely between two parties.

5. Fully Homomorphic Encryption: a form of post-quantum cryptography.

We'll quickly introduce the concepts used in the projects.

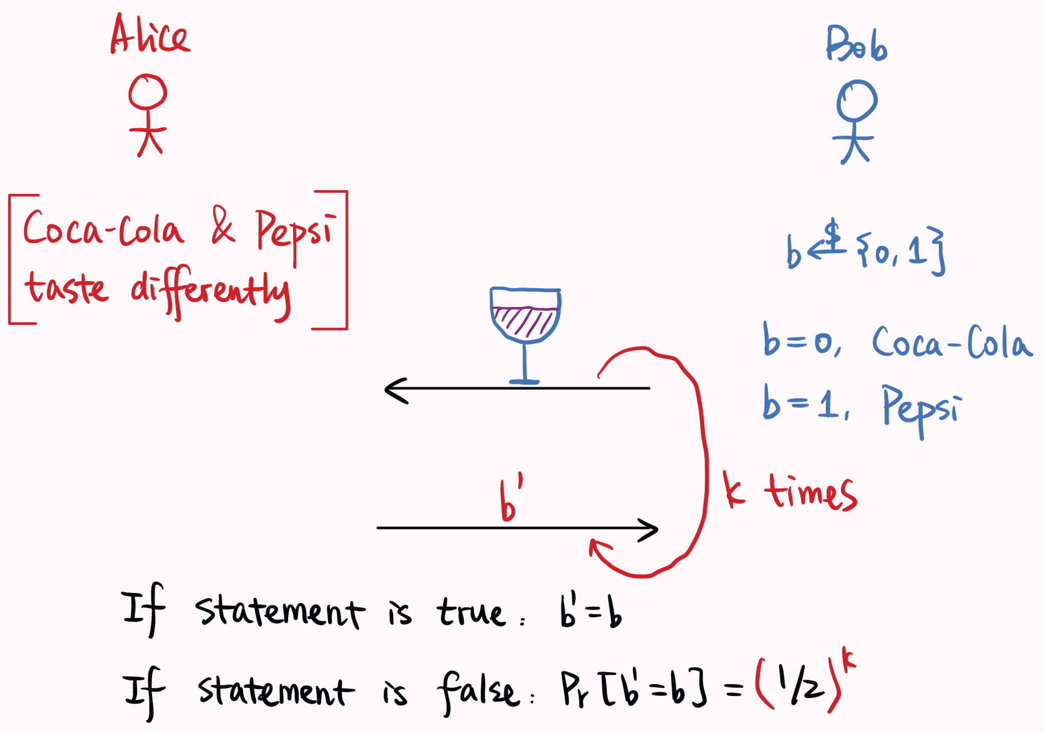
§1.5.1 Zero-Knowledge Proofs

This is to prove something without *revealing* any additional knowledge.

Example

Alice claims to be able to differentiate between Coca-Cola and Pepsi! She wants to prove this to Bob without revealing her secrets.

Bob will randomly sample a bit $b \overset{\$}{\leftarrow} \{0,1\}$, with $b = 0$ being Coca-Cola and $b = 1$ being Pepsi. Bob will let Alice taste this drink. Alice will give a guess b' of what drink it is.


Alice
Bob
Coca-Cola & Pepsi taste differently
$b \xleftarrow{\$} \{0,1\}$
$b=0$, Coca-Cola
$b=1$, Pepsi
b'
k times
If statement is true: $b'=b$
If statement is false: $\Pr[b'=b] = (1/2)^k$

If the statement is true, $b' = b$ (Alice gives the correct prediction).

If the statement is false, $\Pr[b' = b] = \frac{1}{2}$ (Alice is guessing at 0.5 probability).

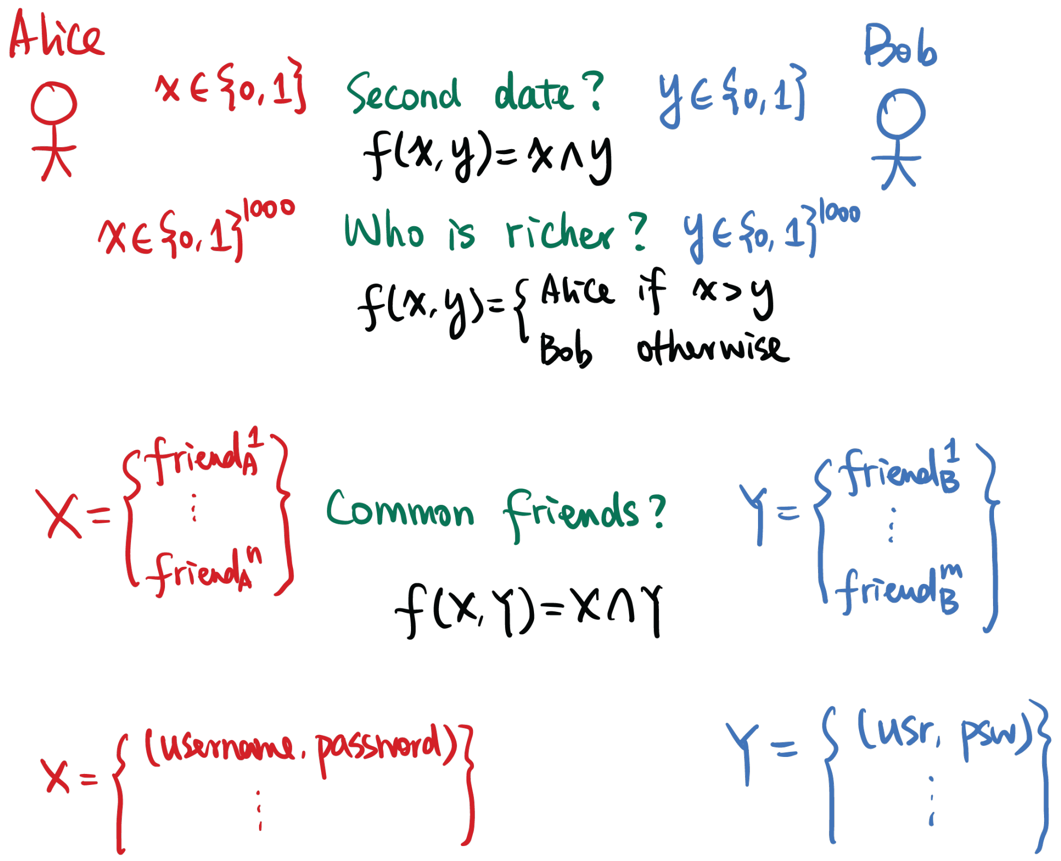
To enhance this, we can run this a total of k times. If we run it enough times, Bob will be more and more confident in believing this. Alice getting this correct by chance has a $\frac{1}{2^k}$ probability.

The key idea, however, is that Bob doesn't gain any knowledge of how Alice differentiates.

Remark. This is a similar strategy in proving graph non-isomorphism.

For people who have seen this before, generally speaking, any NP language can be proved in zero-knowledge. Alice has the *witness* to the membership in NP language.

§1.5.2 Secure Multi-Party Computation



Alice
$x \in \{0,1\}$
Second date?
$y \in \{0,1\}$
Bob
$f(x,y) = x \wedge y$
$x \in \{0,1\}^{1000}$
Who is richer?
$y \in \{0,1\}^{1000}$
$f(x,y) = \begin{cases} \text{Alice} & \text{if } x > y \\ \text{Bob} & \text{otherwise} \end{cases}$
$X = \{\text{friend}_A^1, \ldots, \text{friend}_A^n\}$
Common friends?
$Y = \{\text{friend}_B^1, \ldots, \text{friend}_B^m\}$
$f(X,Y) = X \cap Y$
$X = \{(\text{username}, \text{password}), \ldots\}$
$Y = \{(\text{usr}, \text{psw}), \ldots\}$

Example (Secure AND)

Alice and Bob go on a first date, and they want to figure out whether they want to go on a second date. They will only go on a second date if and only if both agree to a second date.

How will they agree on this? They could tell each other, but this could be embarrassing. One way is for them to share with a third-party (this is what dating apps do!). However, there might not always be an appropriate third party (in healthcare examples, not everyone can be trusted with the data).

In this case, Alice has a choice bit $x \in \{0,1\}$ and Bob has a choice bit $y \in \{0,1\}$. They are trying to jointly compute $f(x,y) = x \wedge y$.

Example (Yao's Millionaires' Problem)

Perhaps, Alice and Bob wants to figure out who is richer. The inputs are $x \in \{0,1\}^{1000}$ and $y \in \{0,1\}^{1000}$ (for simplification, let's say they can express their wealth in 1000 bits). The output is the person who has the max.

$$f(x,y) = \begin{cases} \text{Alice} & \text{if } x > y \\ \text{Bob} & \text{otherwise} \end{cases}$$

Example (Private Set Intersection)

Alice and Bob meet for the first time and want to determine which of their friends they share. However, they do not want to reveal who specifically are their friends.

X is a set of A's friends $X = \{\mathsf{friend}_A^1, \mathsf{friend}_A^2, \cdots, \mathsf{friend}_A^n\}$ and Bob also has a set $Y = \{\mathsf{friend}_B^1, \mathsf{friend}_B^2, \cdots, \mathsf{friend}_B^m\}$. They want to jointly compute

$$f(X,Y) = X \cap Y.$$

You might need to reveal the cardinality of these sets, but you could also pad them up to a maximum number of friends.

This has a lot of applications in practice! In Google Chrome, your browser will notify you that your password has been leaked on the internet, without having access to your passwords in the clear. X will be a set of *your* passwords, and Google will have a set Y of *leaked* passwords. The *intersection* of these sets are which passwords have been leaked over the internet, without revealing all passwords in the clear.

Question. Isn't the assumption that the size is revealed weaker than using a trusted third-party?

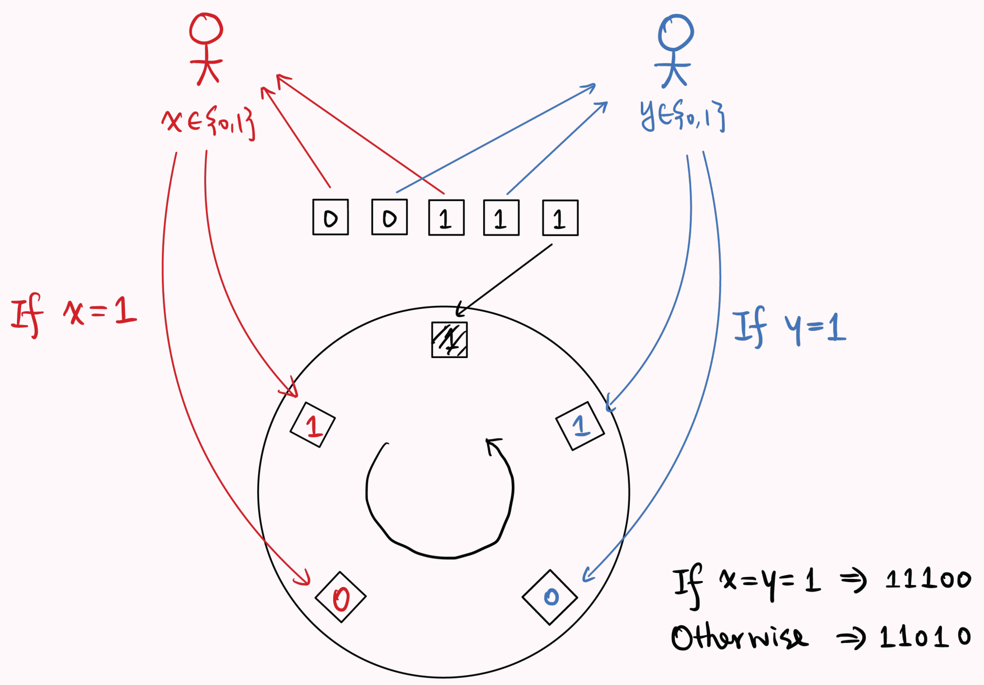
Yes, however in some cases (hospital health records), parties are legally obliged to keep data secure. We wish for security more than the secrecy of cardinality.

In the general case, Alice and Bob have some inputs x and y with bounded length, and they want to jointly compute some function f on these inputs. This is Secure Two-Party Computation. Furthermore, there could be multiple parties $x_1, \ldots, x_n$ that jointly compute $f(x_1, \ldots, x_n)$ that hides each input. This is Secure Multiparty Computation.

We'll explore a toy example with the bit-AND from the dating example.

Example (Private Dating)

Alice and Bob have choice bits $x \in \{0,1\}$ and $y \in \{0,1\}$ respectively. There is a *physical* round table with 5 identical slots, one already filled in with a 1 facing down.



Alice and Bob each have identical $0, 1$ cards (each of the 0 and 1 cards are indistinguishable from cards of the same value). Alice places her cards on the 2 slots in some order, and Bob does the same.

They then spin the table around and reveal all the cards, learning $x \wedge y$.

If $x = 1$, Alice places it as 1 on top of 0, and if $y = 1$, Bob places it as 1 on top of 0 as well. Otherwise, they flip them. If $x = y = 1$, then the 0's will be adjacent. If $x \neq y$, the order will be $1, 1, 0, 1, 0$ (the 0's are not adjacent), regardless of which of Alice or Bob produced $x = 0$ (or both!).

Question. If Alice puts 1, and the output is 0, she could infer the information from Bob. However, this is allowed. Whatever can be inferred from the desired output is inferred.

The more sensitive part is when if you put a 0, you don't learn whether the other party also put down a 0.

This is a toy example! It doesn't use cryptography at all! Two parties have to sit in front of a table. This is called card-based cryptography. We will be using more secure primitives.

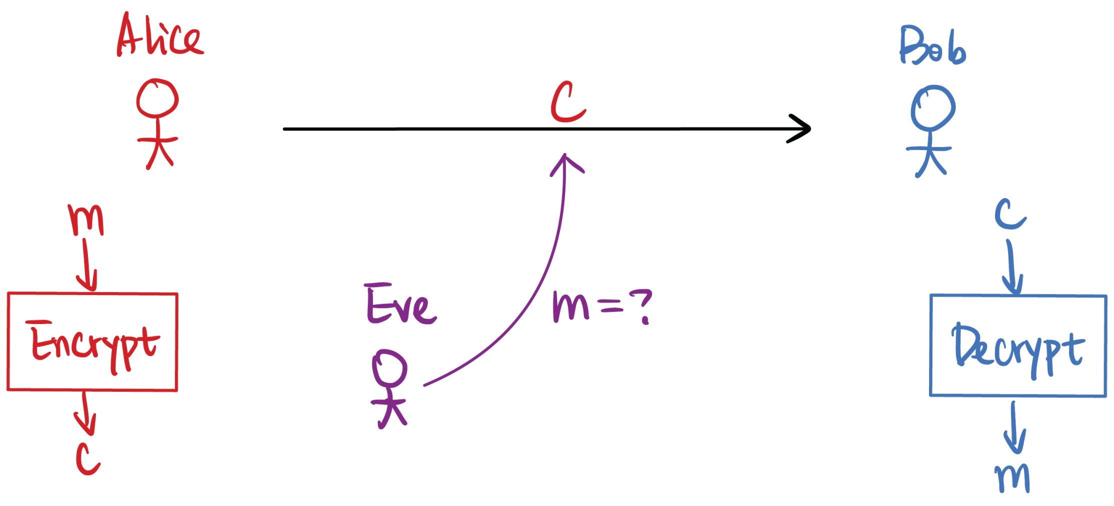
§1.5.3 Fully Homomorphic Encryption

We'll come back to the secure messaging example.

Alice wants to send Bob a message. She encrypts it somehow and sends a ciphertext $c_1 = \mathsf{Enc}(m_1)$. A nice feature for some encryption schemes is for Eve to do some computation homomorphically on the ciphertexts. Eve might possibly want to add ciphertexts (that leads to plaintext adding)

$$c_1 = \mathsf{Enc}(m_1), c_2 = \mathsf{Enc}(m_2) \Rightarrow c' = \mathsf{Enc}(m_1 + m_2)$$

or perhaps $c'' = \mathsf{Enc}(m_1 \cdot m_2)$, or compute arbitrary functions. *Sometimes*, this is simply adding $c_1 + c_2$, but usually not.

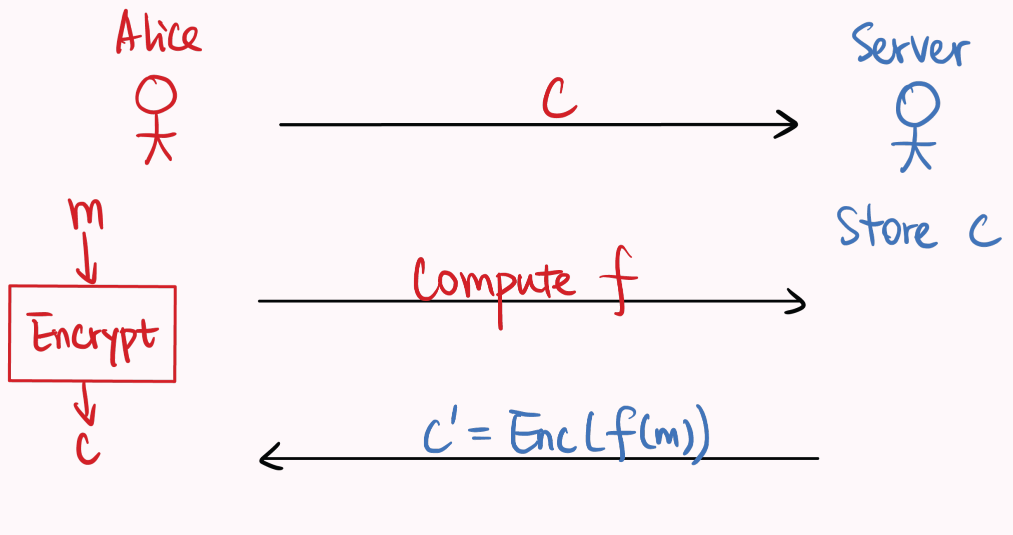


$$\begin{aligned} c_1 &= \mathrm{Enc}(m_1) \\ c_2 &= \mathrm{Enc}(m_2) \end{aligned} \quad \Rightarrow \quad \begin{aligned} c' &= \mathrm{Enc}(m_1 + m_2) \\ c'' &= \mathrm{Enc}(m_1 \cdot m_2) \end{aligned}$$

We want to hopefully compute any function in polynomial time!

Example (Outsourced Computation)

Alice has some messages but doesn't have enough compute. There is a server that has *a lot* of compute!



Alice encrypts her data and stores it in the server. At some point, Alice might want to compute a function on the encrypted data on the server, without the server revealing the original data.

This is an example of how fully homomorphic encryption can be useful.

Remark. This problem was not solved until 2009 (when Peihan started her undergrad). Theoretically, it doesn't even seem that possible! Being able to compute functions on ciphertexts that correspond to functions on plaintexts.

To construct fully-homomorphic encryption, we'll be using lattice-based cryptography. This is also the only cryptographic primitive that is quantum-secure[4].

§1.5.4 Further Topics

We might cover some other topics:

- Differential Privacy
- Crypto applications in machine learning
- Crypto techniques used in the blockchain[5]

What else would you like to learn? What else do you want to understand? Do go through the semester with these in mind! *How do I log into Google? How do I send messages to friends?*

Peihan will collect responses at the start and middle of the semester to shape course content.

[4]Everything before this can be broken if quantum computers become mainstream!
[5]One important techniques is Zero-Knowledge proofs, for example.

§1.6 A Quick Survey

Peihan conducted the following poll in-class to gauge content for future lectures. ***By all means, you don't need to know any/all of this going into this course! These will be self-contained in this course!***

Do you know what the following means?

- Polynomial-time algorithm.
- NP-hard problems.
- "a divides b" ($a \mid b$)
- GCDs
- (Extended) Euclidean Algorithms
- Groups
- One-Time pads
- RSA encryption/signature
- Diffie-Hellman Key Exchange
- SHA (hash functions)

§2 January 31, 2023

§2.1 Logistics

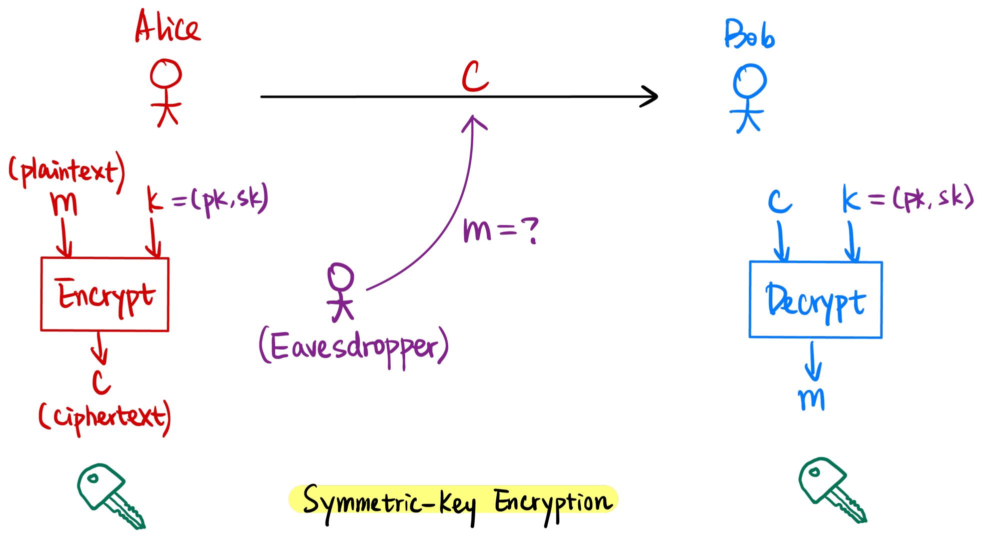
There's an EdStem post asking about topics you're interested, feel free to keep on posting!

We acknowledge the synchronization issues with Panopto. For now, if you want to watch the lecture recordings, you can use the Zoom link linked from the course home page. We can manually sync up EdStem but Panopto cannot be synced up, unfortunately.

§2.2 Encryption Schemes

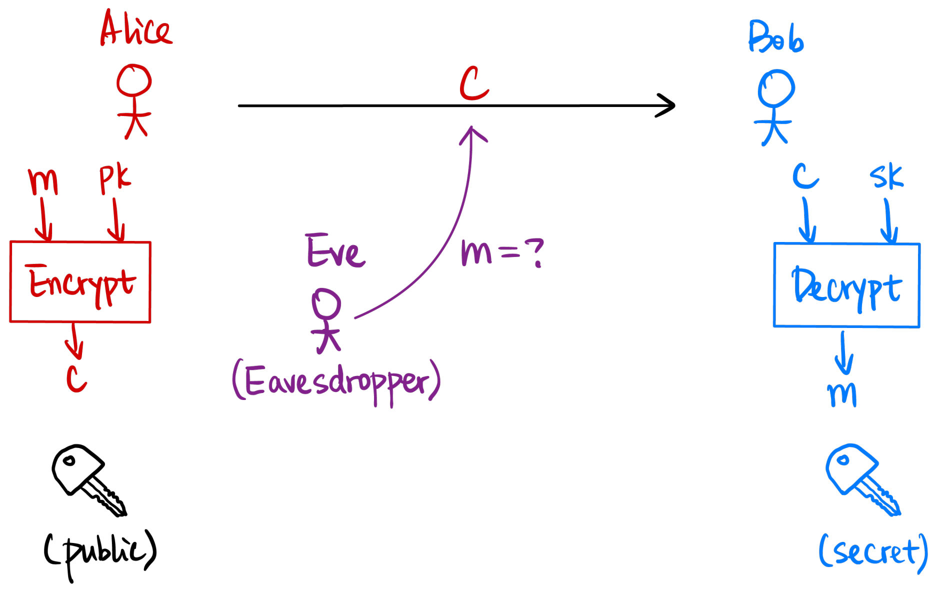
This lecture we'll cover encryption schemes. We briefly mentioned what encryption schemes were last class, we'll dive into the technical content: how we construct them, assumptions, RSA, ElGamal, etc.

Fundamentally, an encryption scheme protects message secrecy. If Alice wants to communicate to Bob, Alice will encrypt a message (plaintext) using some key which gives her a ciphertext. Sending the ciphertext through Bob using a public channel, Bob can use the key to decrypt the ciphertext and recover the message. An eavesdropper in the middle will have no idea what message has been transmitted.



In this case, they are using a shared key, which we called secret-key encryption or symmetric-key encryption.

A stronger version of private-key encryption is called public-key encryption. Alice and Bob do not need to agree on a shared secret key beforehand. There is a keypair (pk, sk), a *public* and *private* key.



§2.2.1 Syntax

Definition 2.1 (Symmetric-Key Encryption)
A symmetric-key encryption (SKE) scheme contains 3 algorithms, $\pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Generation. Generates key $k \leftarrow \mathsf{Gen}$.

Encryption. Encrypts message m with key k, $c \leftarrow \mathsf{Enc}(k, m)$, which we sometimes write as $\mathsf{Enc}_k(m)$.

Decryption. Decrypts using key k to retrieve message m, $m := \mathsf{Dec}(k, c)$, or written as $\mathsf{Dec}_k(c)$.

Note the notation $\leftarrow$ and $:=$ is different. In the case of generation and encryption, the produced key k or c follows a *distribution* (is randomly sampled), but we had better want decryption to be deterministic in producing the message.

Definition 2.2 (Public-Key Encryption)
A public-key encryption (PKE) scheme $\pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ contains the same 3 algorithms,

Generation. Generate keys $(pk, sk) \leftarrow \mathsf{Gen}$.

Encryption. Use the public key to encrypt, $c \leftarrow \mathsf{Enc}(pk, m)$ or $\mathsf{Enc}_{pk}(m)$.

Decryption. Use the secret key to decrypt, $m := \mathsf{Dec}(pk, c)$ or $\mathsf{Dec}_{pk}(c)$.

Question. If we can construct public-key encryption, why do we even bother with secret-key encryption? We could just use the (pk, sk) pair for our secret key, and this does the same thing.

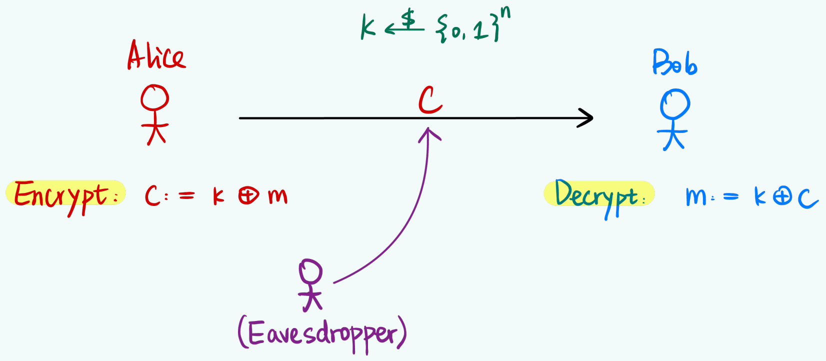
1. First of all, public-key encryption is almost always *more expensive*. Symmetric-key encryption schemes give us efficiency.

2. Public-key encryption relies on much stronger computational assumptions.

§2.2.2 Symmetric-Key Encryption Schemes

Definition 2.3 (One-Time Pad)

Secret key is a uniformly randomly sampled n bit string $k \xleftarrow{\$} \{0,1\}^n$.


$k \xleftarrow{\$} \{0,1\}^n$
Alice
Bob
c
Encrypt: $c := k \oplus m$
Decrypt: $m := k \oplus c$
(Eavesdropper)

Encryption. Alice uses the secret key and bitwise-XOR with the plaintext.

$$\begin{array}{rr} \text{secret key} & k = 0100101 \\ \oplus\ \text{plaintext} & m = 1001001 \\ \hline \text{ciphertext} & c = 1101100 \end{array}$$

Decryption. Bob uses the secret key and again bitwise-XOR with the ciphertext

$$\begin{array}{rr} \text{secret key} & k = 0100101 \\ \oplus\ \text{ciphertext} & c = 1101100 \\ \hline \text{plaintext} & c = 1001001 \end{array}$$

This is widely used in cryptography, called *masking* or *unmasking*.

Question. Why is this correct?

An XOR done twice with the same choice bit b is the identity. Or, an element is its own inverse with the XOR operator.

Question. Why is this secure?

We can think about this as the distribution of c. $\forall m \in \{0,1\}^n$, the encryption of m is uniform over $\{0,1\}^n$ (since k was uniform).

Another way to think about this is that for any two messages $m_0, m_1 \in \{0,1\}^n$, $\mathsf{Enc}_k(m_0) \equiv \mathsf{Enc}_k(m_1)$. That is, the encryptions follow the *exact same* distribution. In this case, they are both uniform, but this is not always the case.

Question. Can we reuse k? Should we use the same key again to encrypt another message? Or, it is possible for the eavesdropper to extract information.

For example, $\mathsf{Enc}_k(m)$ is $c := k \oplus m$, and $\mathsf{Enc}_k(m')$ is $c' := k \oplus m'$. If the two messages are the same, the ciphertexts are the same.

By XOR c and c', we get

$$\begin{aligned} c \oplus c' &= (\cancel{k} \oplus m) \oplus (\cancel{k} \oplus m') \\ &= m \oplus m' \end{aligned}$$

This is why this is an *one-time* pad. This is a bit of an issue, to send an n-bit message, we need to agree on an n-bit message.

In fact, this is *the best* that we can do.

Theorem 2.4

Informally, for perfect (information-theoretic[6]) security, the key space must be at least as large as the message space.

$$|\mathcal{K}| \geq |\mathcal{M}|$$

where $\mathcal{K}$ is the key space and $\mathcal{M}$ is the message space.

Question. How can we circumvent this issue?

The high level idea is that we weaken our security guarantees *a little*. Instead of saying that they have to be *exactly the same* distribution, we say that they are *hard to distinguish* for an adversary with limited computational power. This is how modern cryptography gets around these lower bounds in classical cryptography. We can make *computational assumptions* about cryptography.

We can think about computational security,

[6]That the distributions of ciphertexts are identical, that $\mathsf{Enc}_k(m_0) \equiv \mathsf{Enc}_k(m_1)$.

Definition 2.5 (Computational Security)

We have computational security when two ciphertexts have distribution that cannot be distinguished using a polynomial-time algorithm.

Definition 2.6 (Polynomial-Time Algorithm)

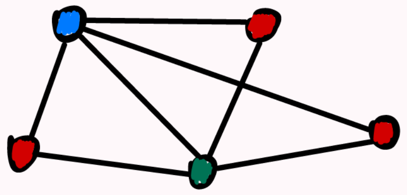
A polynomial time algorithm $A(x)$ is one that takes input x of length n, A's running time is $O(n^c)$ for a constant c.

Definition 2.7 (NP Problem)

A decision problem is in nondeterministic polynomial-time when its solution can be *verified* in polynomial time.

Example 2.8 (Graph 3-Coloring)

Given a graph, does it have a 3-coloring such that no two edges join the same color? For example,



This can be *verified* in polynomial time (we can check if such a coloring is a valid 3-coloring), but it is computed in NP time.

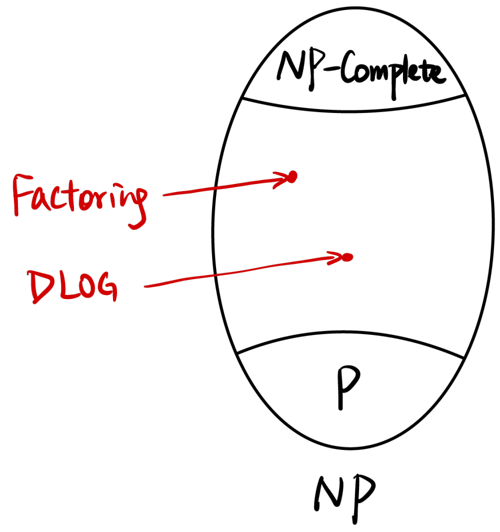
Definition 2.9

An NP-complete problem is a "hardest" problem in NP. Every problem in NP is at least as hard as an NP-complete problem.

Right now, we assume P $\neq$ NP. As of right now, there is no realistic algorithm that can solve any NP problem in polynomial-time.

Even further, we pick some problems not in NP-complete, not in P. We assume they are neither NP-complete nor in P (we don't yet have a reduction, but we don't know if one could exist)

The reasoning behind using these problems is as we have no good cryptoscheme relying solely on NP-complete problems (we need something weaker).



Going back to our definition of computational security definition 2.5,

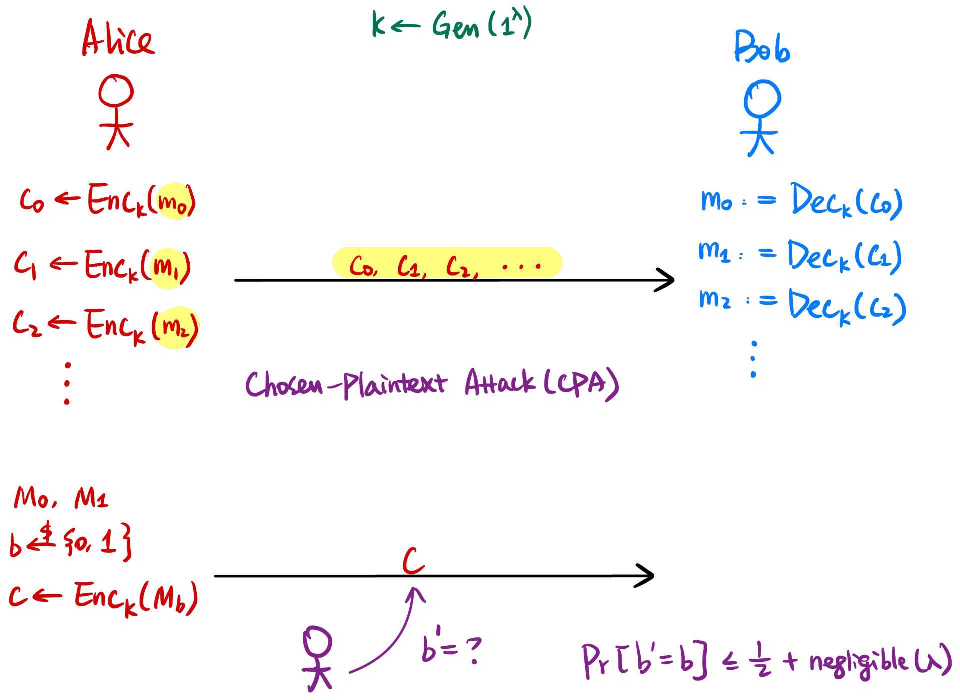
Definition (Computational Security)
We say that the adversary is computationally bounded (is only a *polynomial-time algorithm*), that ∀probabilic poly-time algorithm $\mathcal{A}$,

$$\mathsf{Enc}_k(m_0) \overset{c}{\simeq} \mathsf{Enc}_k(m_1)$$

Where $\overset{c}{\simeq}$ is "computationally indistinguishable".

What does it mean for distributions to be "computationally indistinguishable"? Let's say Alice encrypts multiple messages $m_0, m_1, \ldots$ to Bob and produces $c_0, c_1, \ldots$. Even if Eve can see all plaintexts m_i in the open and ciphertexts c_i in the open, between known m_0, m_1 and randomly encrypting one of them $c \leftarrow \mathsf{Enc}_k(m_b)$ where $b \overset{\$}{\leftarrow} \{0,1\}$, the adversary cannot determine what the random choice bit b is. That is, $\Pr[b = b'] \leq \frac{1}{2} + \mathsf{negligible}(\lambda)$[7]. This is Chosen-Plaintext Attack (CPA) Security.

[7] λ is the security parameter, roughly a measure of how secure the protocol is. If it were exactly equal $\frac{1}{2}$, we have information-theoretic security.


$k \leftarrow Gen(1^\lambda)$
Alice
Bob
$c_0 \leftarrow Enc_k(m_0)$
$c_1 \leftarrow Enc_k(m_1)$
$c_2 \leftarrow Enc_k(m_2)$
$c_0, c_1, c_2, \dots$
$m_0 := Dec_k(c_0)$
$m_1 := Dec_k(c_1)$
$m_2 := Dec_k(c_2)$
Chosen-Plaintext Attack (CPA)
M_0, M_1
$b \xleftarrow{\$} \{0,1\}$
$c \leftarrow Enc_k(M_b)$
c
$b' = ?$
$Pr[b'=b] \le \frac{1}{2} + negligible(\lambda)$

For a key generated $k \leftarrow \mathsf{Gen}(1^\lambda)$.

Theoretically, for λ a security parameter and an adversary running in time $\mathsf{poly}(\lambda)$, the adversary should have distinguishing advantage $\mathsf{negligible}(\lambda)$ where

$$\mathsf{negligible}(\lambda) \ll \frac{1}{\lambda^c} \qquad \forall \text{ constant } c.$$

In practice, we set $\lambda = 128$. This means that the best algorithm to break the scheme (e.g. find the secret key) takes time $\sim 2^\lambda$. Currently, this is longer than the age of the universe.

Example 2.10

If the best algorithm is a brute-force search for k, what should our key length be?

It can just be a λ bit string.

Example

What if the best algorithm is no longer a brute-force search, but instead for a key length l takes $\sim \sqrt{2^l}$?

Our key length should be 2λ. Doing the math, we want $\sqrt{2^l} \equiv 2^\lambda$, solving for l gives 2λ.

Going back to the original problem of secret-key encryption, how can we use our newfound cryptographic constructions to improve this?

From a pseudorandom function/permutation (PRF/PRP), we can reuse our secret key by passing it through the pseudorandom function.

The current practical construction for PRD/PRP is called the block cipher, and the standardized implementation is AES[8]

It is a computational assumption[9] that the AES construction is secure, and the best attack is currently a brute-force search (in both classical and quantum computing realms).

§2.2.3 Public-Key Encryption Schemes

Using computational assumptions, we explore some public-key encryption schemes.

RSA Encryption. This is based on factoring/RSA assumption, that factoring large numbers is hard.

ElGamal Encryption. This is based on the discrete logarithm/Diffie-Hellman Assumption, that finding discrete logs in $\mathbb{Z}_p$ is hard.

Lattice-Based Encryption. The previous two schemes are not quantum secure. Quantum computation will break these schemes. Lattice-based encryption schemes are post-quantum secure. They are associated with the difficulty of finding 'short' vectors in lattices[10].

Another thing worth mentioning is that

Theorem 2.11

(Very informally,) It is impossible to construct PKE from SKE in a black-box way. This is called "black-box separation".

We first need to define a bit of number-theory background.

Definition 2.12

We denote $a \mid b$ as a divides b, that is, there is integer c such that $b = a \cdot c$.

Definition 2.13

The $\gcd(a, b)$ is the greatest common divisor of a, b. If $\gcd(a, b) = 1$, then a, b are coprime.

Question. How do we compute gcd? What is its time complexity?

[8]Determined via a competition for such an algorithm in the early 2000s.
[9]Based on heuristic, not involving any number theory!
[10]Covered later in class, we focus on the first two now.

Example

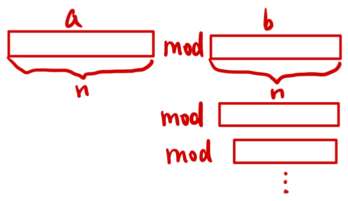
We use the Euclidean Algorithm. Take $\gcd(12, 17)$,

$$\begin{aligned} 17 &\mod 12 = 5 \\ 12 &\mod 5 = 2 \\ 5 &\mod 2 = 1 \\ 2 &\mod 1 = 0 \end{aligned}$$

or take $\gcd(12, 18)$

$$\begin{aligned} 18 &\mod 12 = 6 \\ 12 &\mod 6 = 0 \end{aligned}$$

If we have two bitstrings of length n bits, what is the running time of the Euclidan Algorithm?



Very informally, we see that every step, the length of a, b decrease by approximately 1 bit. Then, finding gcd is roughly order $O(n)$.

Definition 2.14 (Mod)

$a \mod N$ is the remainder of a when divided by N.

$a \equiv b \pmod N$ means when a and b are congruent modulo N. That is, $a \mod N = b \mod N$.

Question. How might we compute $a^b \mod N$? What is the time complexity? Let a, b, N be n-bit strings.

Naïvely, we can repeatedly multiply. But this takes b steps (2^n).

We can ‘repeatedly square’. For example, we can get to a^8 faster by getting a^2, squaring to get a^4, and again to get a^8. We can take the bitstring of b and determine how to compute this.

Example

If $b = 100101_2$, we take $a \cdot a^4 \cdot a^{32} \mod N$ which can be calculated recursively (an example is given in the first assignment).

The time complexity of this is order $O(n)$ for n-bit a, b, N[11].

Theorem 2.15 (Bezout's Theorem, *roughly*)

If $\gcd(a, N) = 1$, then $\exists b$ such that

$$a \cdot b \equiv 1 \pmod{N}.$$

This is to say, a is invertible modulo N. b is its inverse, denoted as a^{-1}.

Question. How do we compute b?

We can use the Extended Euclidean Algorithm!

Example

We write linear equations of a and N that sum to 1, using our previous Euclidean Algorithm. Take the previous example $\gcd(12, 17)$,

$$\begin{aligned} 17 &\mod 12 = 5 \\ 12 &\mod 5 = 2 \\ 5 &\mod 2 = 1 \\ 2 &\mod 1 = 0 \end{aligned}$$

We write this as

$$\begin{aligned} 5 &= 17 - 12 \cdot 1 \\ 2 &= 12 - 5 \cdot 2 = 12 \cdot x + 17 \cdot y \\ 1 &= 5 - 2 \cdot 2 = 12 \cdot x' + 17 \cdot y' \end{aligned}$$

where we substitute the linear combination of 5 into 5 on line 2, substitute linear combination of 2 into 2 on line 1, each producing another linear combination of $12, 17$.

If $\gcd(a, N) = 1$, we use the Extended Euclidean Algorithm to write $1 = a \cdot x + N \cdot y$, then $1 \equiv a \cdot x \pmod{N}$.

[11] Not exactly order n, we should add the complexity of multiplication. However, this should be bounded by N since we can log at every step.

Definition 2.16 (Group of Units mod N)

We have set

$$\mathbb{Z}_N^\times := \{a \mid a \in [1, N-1], \gcd(a, N) = 1\}$$

which is the group of units modulo N (they are units since they all have an inverse by above).

Definition 2.17 (Euler's Phi Function)

Euler's phi (or totient) function, $\phi(N)$, counts the number of elements in this set. That is, $\phi(N) = |\mathbb{Z}_N^\times|$.

Theorem 2.18 (Euler's Theorem)

For all a, N where $\gcd(a, N) = 1$, we have that

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

With this, we can start talking about RSA.

§2.2.4 RSA

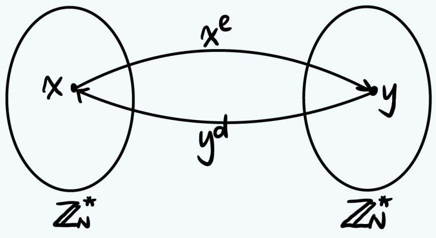
We first define the RSA assumption.

Definition 2.19 (Factoring Assumption)

Given two n-bit primes p, q, we compute $N = p \cdot q$. Given N, it's computationally hard to find p and q (classically).

Definition 2.20 (RSA Assumption)
Given two n-bit primes, we again compute $N = p \cdot q$, where $\phi(N) = (p-1)(q-1)$. We choose an e such that $\gcd(e, \phi(N)) = 1$ and compute $d \equiv e^{-1} \pmod{\phi(N)}$.

Given N and a random $y \overset{\$}{\leftarrow} \mathbb{Z}_N^\times$, it's computationally hard to find x such that $x^e \equiv y \pmod N$.



However, given p, q, it's easy to find d. We know $\phi(N) = (p-1)(q-1)$, so we can compute d from e by running the Extended Euclidean Algorithm. Then, taking $(x^e)^d \equiv x^{ed} \equiv x$ which allows us to extract x again.

Encrypting is exactly raising by power d, and decrypting is raising again by power e.

Remaining questions:

- How can we generate primes p, q?
- How can we pick e such that $\gcd(e, \phi(N)) = 1$?
- What security issues can you see?

We'll continue next class.

§3 February 2, 2023

§3.1 RSA Encryption, *continued*

Recall: that the RSA encryption algorithm contains 3 components:

$\mathsf{Gen}(1^\lambda)$: Generate two n-bit primes p, q. We compute $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$. Choose e such that $gcd(e, \phi(N)) = 1$. We compute $d = e^{-1} \mod \phi(N)$. Our public key $pk = (N, e)$, our secret key is $sk = d$.

$\mathsf{Enc}_{pk}(m)$: $c = m^e \mod N$.

$\mathsf{Dec}_{sk}(c)$: $m = c^d \mod N$.

We have a few remaining questions:

1. How do we generate 2 primes p, q?
2. How do we choose such an e?
3. How do we compute $d = e^{-1} \mod \phi(N)$?
4. How do we efficiently compute $m^e \mod N$ and $c^d \mod N$.

How do we resolve these issues to ensure the Gen step is efficient (polynomial time).

1. We pick an arbitrary number p and check for primality efficiently (using Miller Rabin, a probabilistic primality test). We pick random numbers until they are prime. Since primes are 'pretty dense' in the integers, this can be done efficiently.
2. We can also guess! Since we're unsure whether coprime numbers are dense, we can pick small prime e.
3. We can compute d using the Extended Euclidean Algorithm.
4. We can repeatedly square (using fast power algorithm).

A note that we have correctness with this scheme: $(m^e)^d = m^{ed} = m \mod N$.

Question. Still, are there any security issues?

- It relies on factoring being difficult (this is the computational assumption). Post-quantum, Shor's Algorithm will break RSA.
- Recall last lecture that CPA (Chosen-Plaintext Attack) security was defined as an adversary not being able to discern between an encryption of m_0 and m_1, *knowing* m_0 and m_1 in the clear.

  Eve could just encrypt m_0 and m_1 themselves using public e, and discern which of the plaintexts the ciphertext corresponds to. Using RSA, you *really* have to be careful. For RSA, this is a very concrete attack.

  The concrete reason is that the encryption algorithm Enc is *deterministic*. If you encrypt the same message twice, it will be the same ciphertext. We really want to be sure that $m \xleftarrow{\$} \mathbb{Z}_N^\times$ (that it has enough entropy).

Returning on the RSA assumption. It's crucial that the $y \xleftarrow{\$} \mathbb{Z}_N^\times$ is randomly sampled.

Question. In practice, how can RSA be useful with these limitations?

As long as we pick the plaintext which is randomly sampled, security for RSA holds.

> **Remark.** In practice, we usually set length of p and q to be 1024 bits, and the key length is 2048 bits. This is because of better algorithms for finding big primes.

Question. Asked on Ed: What happens if $p \mid m$ (or $q \mid m$)?

Correctness still holds[12].

However, security will be broken. If $p \mid (m^2 \mod N)$, then $gcd(c, N) = p$ will factor N.

However, this is *very* unlikely! Sampling $m \xleftarrow{\$} [1, N-1]$, then

$$\Pr[p \mid m] \equiv \frac{q}{N} = \frac{1}{p} = \frac{1}{2^{\theta(\lambda)}} = \mathsf{negligible}(\lambda)$$

When sending k messages,

$$\Pr[p \mid m \text{ for any } m] \leq \frac{k}{p} \equiv \frac{\mathsf{poly}(\lambda)}{2^{\theta(\lambda)}}$$

which is still negligible. Put simply, Alice has just as much chance to break RSA by randomly factoring as she is to send a message that is a multiple of p or q.

§3.2 Intro to Group Theory

[12] Can do out, or by using Chinese Remainder Theorem to see that it preserves the qualities we need mod p and mod q.

Definition 3.1 (Group)

A group is a set $\mathbb{G}$ along with a binary operation $\circ$ with properties:

Closure. $\forall g, h \in \mathbb{G}$, $g \circ h \in \mathbb{G}$.

Existence of an identity. $\exists e \in \mathbb{G}$ such that $\forall g \in \mathbb{G}$, $e \circ g = g \circ e = g$.

Existence of inverse. $\forall g \in \mathbb{G}$, $\exists h \in \mathbb{G}$ such that $g \circ h = h \circ g = e$. We denote the inverse of g as g^{-1}.

Associativity. $\forall g_1, g_2, g_3 \in \mathbb{G}$, $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.

We say a group is additionally *Abelian* if it satisfies

Commutativity. $\forall g, h \in \mathbb{G}$, $g \circ h = h \circ g$.

For a finite group, we use $|\mathbb{G}|$ to denote its *order*.

Example 3.2

$(\mathbb{Z}, +)$ is an Abelian group.

We can check so: two integers sum to an integer, identity is 0, the inverse of a is $-a$, addition is associative and commutative.

$(\mathbb{Z}, \cdot)$ is not a group.

$(\mathbb{Z}_N^\times, \cdot)$ is an Abelian group ($\cdot$ is multiplication mod N).

Definition 3.3 (Cyclic Group)

Let $\mathbb{G}$ be a group of order m. We denote

$$\langle g \rangle := \{e = g^0, g^1, g^2, \ldots, g^{m-1}\}.$$

$\mathbb{G}$ is a cyclic group if $\exists g \in \mathbb{G}$ such that $\langle g \rangle = \mathbb{G}$. g is called a generator of $\mathbb{G}$.

Example

$\mathbb{Z}_p^\times$ (for prime p) is a cyclic group of order $p - 1$[13].

$$\mathbb{Z}_7^\times = \{3^0 = 1, 3^1, 3^2 = 2, 3^3 = 6, 3^4 = 5, 3^5 = 5\}.$$

[13]A proof of this extends beyond the scope of this course, but you are recommended to check out Math 1560 (Number Theory) or Math 1580 (Cryptography). You can take this on good faith.

Question. How do we find a generator?

For every element, we can continue taking powers until $g^\alpha = 1$ for some α. We hope that $\alpha = p-1$ (the order of g is the order of the group), but we know at least $\alpha \mid p-1$.

§3.3 Computational Assumptions

We have a few assumptions we make called the Diffie-Hellman Assumptions, in order of **weakest to strongest**[14] assumptions.

Let $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$ be a cyclic group $\mathbb{G}$ or order q (a $\theta(\lambda)$-bit integer) with generator g. For integer groups, keys are usually 2048-bits. For elliptic curve groups, keys are usually 256-bits.

Definition 3.4 (Discrete Lgoarithm (DLOG) Assumption)
Let $x \overset{\$}{\leftarrow} \mathbb{Z}_q$. We compute $h = g^x$.

Given $(\mathbb{G}, q, g, h)$, it's computationally hard to find the exponent x (classically).

Definition 3.5 (Computational Diffie-Hellman (CDH) Assumption)
$x, y \overset{\$}{\leftarrow} \mathbb{Z}_q$, compute $h_1 = g^x$, $h_2 = g^y$.

Given $(\mathbb{G}, q, g, h_1, h_2)$, it's computationally hard to find g^{xy}.

Definition 3.6 (Decisional Diffie-Hellman (DDH) Assumption)
$x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_q$. Compute $h_1 = g^x$, $h_2 = g^y$.

Given $(\mathbb{G}, q, g, h_1, h_2)$, it's computationally hard to distinguish between g^{xy} and g^z.

$$(g^x, g^y, g^{xy}) \overset{c}{\simeq} (g^x, g^y, g^z).$$

§3.4 ElGamal Encryption

The ElGamal encryption scheme involves the following:

[14] If one can solve DLOG, we can solve CDH. Given CDH, we can solve DDH. This is why CDH is *stronger* than DDH, and DDH is *stronger* than DLOG. It's not necessarily true the other way around (similar to factoring and DSA assumptions).

$\mathsf{Gen}(1^\lambda)$: We generate a group $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$. We sample $x \xleftarrow{\$} \mathbb{Z}_q$, compute $h = g^x$. Our public key is $pk = (\mathbb{G}, q, g, h)$, secret key $sk = x$.

$\mathsf{Enc}_{pk}(m)$: We have $m \in \mathbb{G}$. We sample $y \xleftarrow{\$} \mathbb{Z}_q$. Our ciphertext is $c = \langle g^y, h^y \cdot m \rangle$. Note that $h = g^x$, so $g^{xy} \overset{c}{\simeq} g^z$ is a one-time pad for our message m.

$\mathsf{Dec}_{sk}(c)$: To decrypt $c = \langle c_1, c_2 \rangle$, we raise

$$\begin{aligned} c_1^x &= (g^y)^x = g^{xy} \\ m &= \frac{g^{xy} \cdot m}{g^{xy}} = c_2 \cdot (c_1^x)^{-1}. \end{aligned}$$

Notes about ElGamal:

- Our group can be reused! We can use a public group that is fixed. In fact, there are *popular* groups out there used in practice. Some of these are Elliptic Curve groups which are much more efficient than integer groups. You don't need to use the details, yet you can use it! You can use any group, so long as the group satisfies the DDH assumption.
- Similar to RSA, this is breakable post-quantum. Given Shor's Algorithm, we can break discrete log.

§3.5 Secure Key Exchange

Using DDH, we can construct something very important, *secure key exchange.*

Definition 3.7 (Secure Key Exchange)
Alice and Bob sends messages back and forth, and at the end of the protocol, can agree on a shared key.

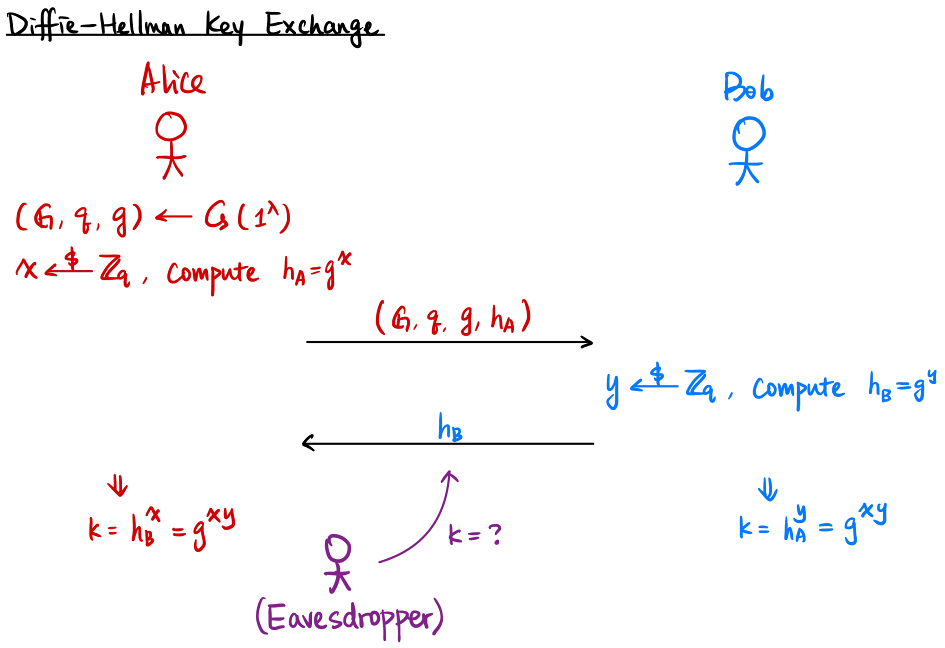
An eavesdropper looking at said communications cannot figure out what shared key they came up with.

Theorem 3.8
Informally, It's impossible to construct secure key exchange from secret-key encryption in a black-box way.

Question. How do we build a key exchange from public-key encryption?

Bob generates a keypair (pk, sk). Alice generates a shared key $k \xleftarrow{\$} \{0,1\}^\lambda$, and sends $\mathsf{Enc}_{pk}(k)$ to Bob.

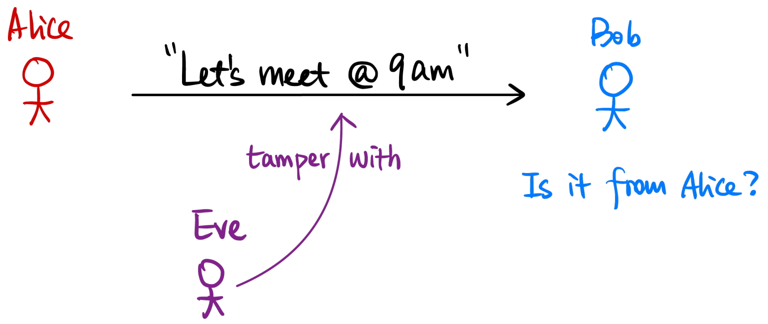
Using Diffie-Hellman, it's very easy. We have group $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$. Alice samples $x \xleftarrow{\$} \mathbb{Z}_q$ and sends g^x. Bob also samples $y \xleftarrow{\$} \mathbb{Z}_q$ and sends g^y. Both Alice and Bob compute $g^{xy} = (g^x)^y = (g^y)^x$.


Diffie-Hellman Key Exchange
Alice
Bob
$(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$
$x \xleftarrow{\$} \mathbb{Z}_q$, compute $h_A = g^x$
$(\mathbb{G}, q, g, h_A)$
$y \xleftarrow{\$} \mathbb{Z}_q$, compute $h_B = g^y$
h_B
$k = h_B^x = g^{xy}$
$k = ?$
$k = h_A^y = g^{xy}$
(Eavesdropper)

What happens in practice is that parties run Diffie-Hellman key exchange to agree on a shared key. Using that shared key, they run symmetric-key encryption. This gives us efficiency. Additionally, private-key encryptions don't rely on heavy assumptions on the security of protocols (such as the DDH, RSA assumptions).

§3.6 Message Integrity

Alice sends a message to Bob, how does Bob ensure that the message came from Alice?



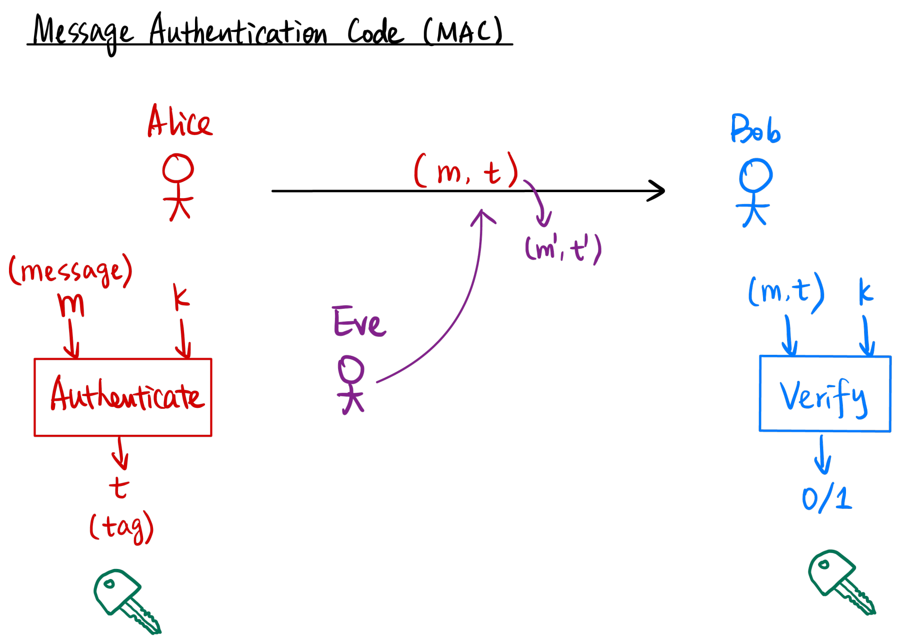
We can build up another line of protocols to ensure message integrity. It's similar to encryption, but the parties run 2 algorithms: *Authenticate* and *Verify*.

Using a message m, Alice can generate a *tag* or *signature*, and Bob can verify (m, t) is either valid or invalid.

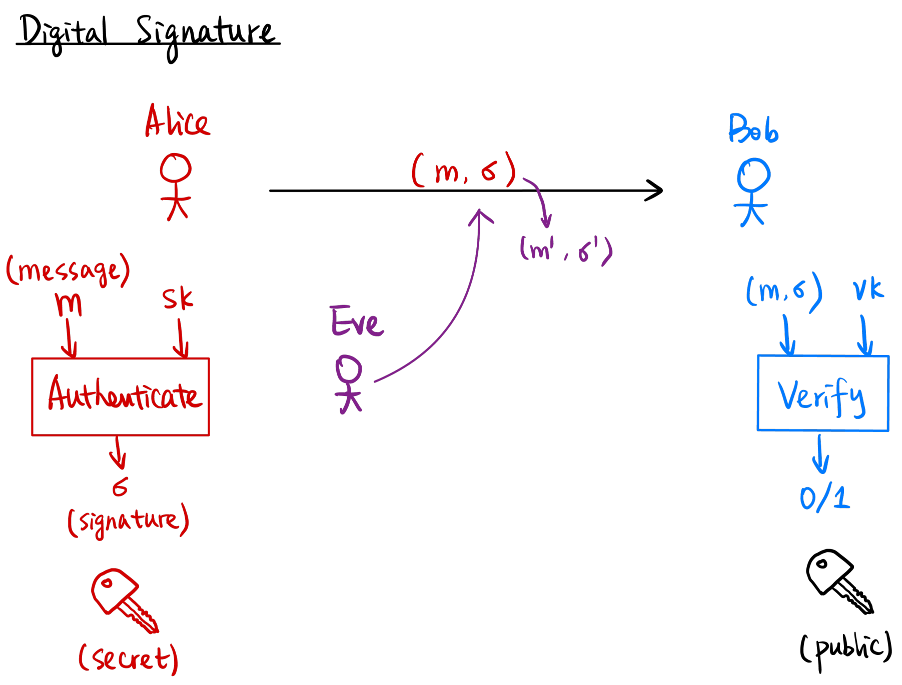
Our adversary has been upgraded to an Eve who can now tamper with messages.

Similarly to encryption, we have symmetric-key and public-key encryption.

Using a shared key k, Alice can authenticate m using k to get a tag k. Similarly, Bob can verify whether (m, t) is valid using k. This is called a Message Authentication Code.



Using a public key vk (verification key) and private key sk (secret/signing key), Alice can sign a message m using signing key sk to get a *signature* σ. Bob verifies (m, σ) is valid using vk. This is called a Digital Signature.



§3.6.1 Syntax

The following is the syntax we use for MACs and digital signatures.

A message authentication code (MAC) scheme consists of $\pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$.

Generation. $k \leftarrow \mathsf{Gen}(1^\lambda)$.

Authentication. $t \leftarrow \mathsf{Mac}_k(m)$.

Verification $0/1 := \mathsf{Verify}_k(m, t)$.

A digital signature scheme consists of $\pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

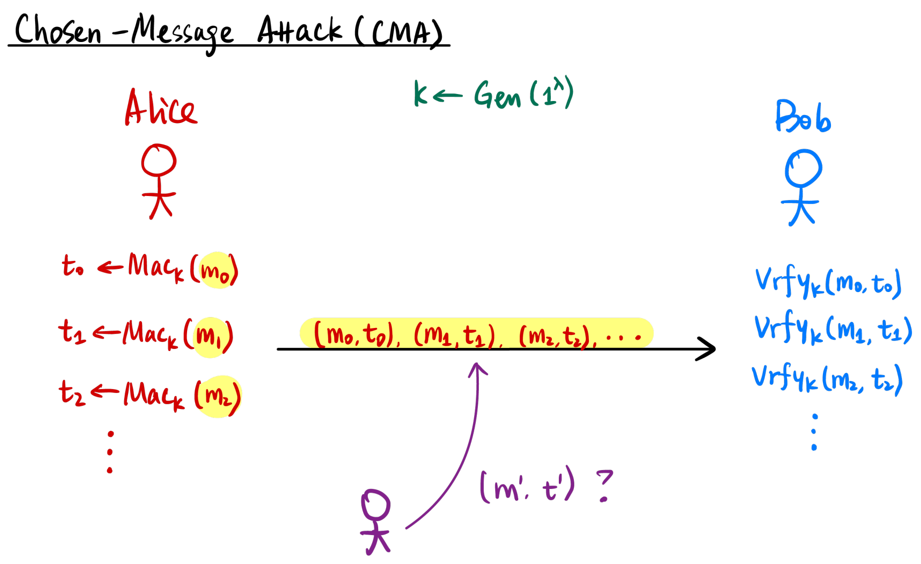
Generation. $(sk, vk) \leftarrow \mathsf{Gen}(1^\lambda)$.

Authentication. $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$.

Verification $0/1 := \mathsf{Verify}_{vk}(m, \sigma)$.

§3.6.2 Chosen-Message Attack

Similar to chosen-plaintext attack from encryption, we have chosen-message attack security. An adversary chooses a number of messages to generate signatures or tags for. After that, the adversary will try to generate another valid pair of message and tag. We want to make sure that generating a new pair of message and tag is hard.


Chosen-Message Attack (CMA)
$k \leftarrow \text{Gen}(1^\lambda)$
Alice
Bob
$t_0 \leftarrow \text{Mac}_k(m_0)$
$t_1 \leftarrow \text{Mac}_k(m_1)$
$t_2 \leftarrow \text{Mac}_k(m_2)$
$(m_0, t_0), (m_1, t_1), (m_2, t_2), \ldots$
$\text{Vrfy}_k(m_0, t_0)$
$\text{Vrfy}_k(m_1, t_1)$
$\text{Vrfy}_k(m_2, t_2)$
(m', t') ?

§3.6.3 Constructions

Very briefly, we discuss constructions for MAC and digital signatures.

Using block ciphers, we have CBC-MAC. Using a hash function, we have HMAC.

For digital signatures, we have RSA which relies on the RSA assumption, or DSA which relies on discrete-log algorithms. There are also lattice signature schemes for post-quantum digital signatures.