

# CSCI 1515: Applied Cryptography

P. Miao

Spring 2026

These are lecture notes for CSCI 1515: Applied Cryptography taught at BROWN UNIVERSITY by Peihan Miao in the Spring of 2026.

These notes were originally taken by Jiahua Chen in Spring 2023, and were updated in Spring 2024 by Sudatta Hor. They are currently being maintained by Chloe Qiao and Ian Hajra. There has been gracious help and input from classmates and fellow TAs. Please direct any mistakes/errata to a thread on Ed, or feel free to pull request or submit an issue to the [notes repository](#).

Notes last updated January 22, 2026.

## Contents

<b>1</b>	<b>January 21, 2026</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.1.1	Staff . . . . .	2
1.1.2	Course Philosophy and Logistics . . . . .	2
1.2	What is cryptography? . . . . .	3
1.3	Secure Communication . . . . .	4
1.3.1	Message Secrecy . . . . .	5
1.3.2	Message Integrity . . . . .	7
1.3.3	Projects . . . . .	9
1.4	Zero-Knowledge Proofs . . . . .	9
1.5	Secure Multi-Party Computation . . . . .	11
1.6	Fully Homomorphic Encryption . . . . .	14
1.7	Further Topics . . . . .	15
1.8	Q & A . . . . .	16

## §1 January 21, 2026

### §1.1 Introduction

The course homepage is at <https://cs.brown.edu/courses/csci1515/spring-2026/>, where you can find information such as the [syllabus](#), projects, homeworks, calendar, lectures and more.

The course is offered in-person in *Friedman 108*, as well as synchronously over Zoom and recorded asynchronously (lectures posted online). Lecture attendance and participation is highly encouraged!

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

#### §1.1.1 Staff

Peihan has been at Brown for a couple of years and this is the third time she is teaching this course. Before Brown, she was at the University of Illinois Chicago. Before that, she finished her PhD at UC Berkeley in 2019 with a focus in cryptography. Afterwards, she worked in industry for a couple of years (Visa) before deciding to come back to academia. She still collaborates with industry to see what problems need to be solved in practice.

During her PhD, she started off doing more theoretical cryptography but also did internships and found applied cryptography fascinating as well. Now she works in both.

Our course staff all have strong backgrounds in cryptography and are excited to help you learn!

#### §1.1.2 Course Philosophy and Logistics

If you look up other *applied* cryptography courses online or at other universities, you will find courses that have “applied” in their title. However, if you look at their syllabus or content, it’s still mostly theoretical crypto. This may (1) deter students from learning about crypto and (2) leave a gap between theoretical crypto and crypto in practice. (2) is bad because if someone makes a mistake in the crypto domain, the consequences are often significant.

As such, it’s helpful for students to get hands-on experience with cryptography:

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

The closest similar course is found at Stanford, which covers theoretical crypto in the first half and more applied crypto in the second. But even that course only covers very basic crypto that are very

well established. In the past 10 years or so, there are new and exciting topics in crypto that are gradually becoming more and more common which we will also cover in this course.

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 and 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 like CSCI 1510 and MATH 1580.

Capstone requirements are cancelled in FY25–26 for CS and joint-CS concentrators.

The following is the grading policy:

<i>Type</i>	<i>Percentage</i>
Introductions	1%
Project 0	4%
Projects 1 & 2 & 5	24% (8% each)
Projects 3 & 4	24% (12% each)
Code Review	8% (4% each)
Homeworks	25% (5% each)
Final Project	14%

You have a total of four (4) late days that you can use to extend the due dates of *Projects 0–5* free of charge, but at most two (2) late days may be applied to any one project. Beyond that, you will be penalized 40% of the project's value for each day it is late. There are **no** late days for *homeworks*, as we want to release the solution guide as soon as possible.

All projects are independent, but collaboration is allowed and encouraged. However, you *must write up your own code*.

If you're sick, let Peihan know with a Dean's note. For a full description of the late policy, refer to the syllabus.

## §1.2 What is cryptography?

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

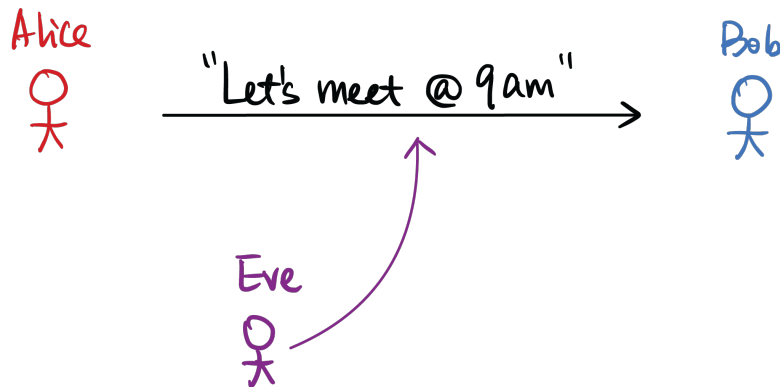
**Question.** Where is cryptography used in practice? What guarantees do we want in these scenarios?

- Online transactions

- When you make a purchase, you might not want people to see your bank balance, what else you have purchased, etc.
- You also want to ensure that it was really *you* who purchased the item and not somebody else i.e. authentication
- Secure messaging
  - End-to-end texting, iMessage
  - We don't want anyone else to see our messages
- Online voting
  - Privacy of votes, validity of votes
- Databases
  - Secure storage
- There is much more!

### §1.3 Secure Communication

We'll start with the most classic form 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.

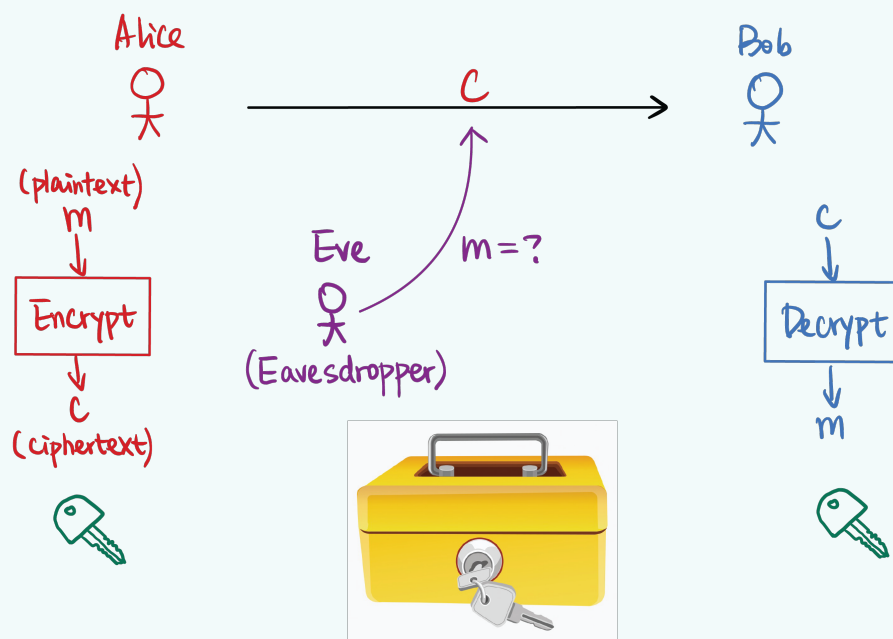
These two guarantees are the most important guarantees! The former is called message secrecy, the latter is called message integrity.

### §1.3.1 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*  $c$ . We call this an *encryption scheme*.

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. The easiest way is for Alice and Bob to agree on a shared secret key.

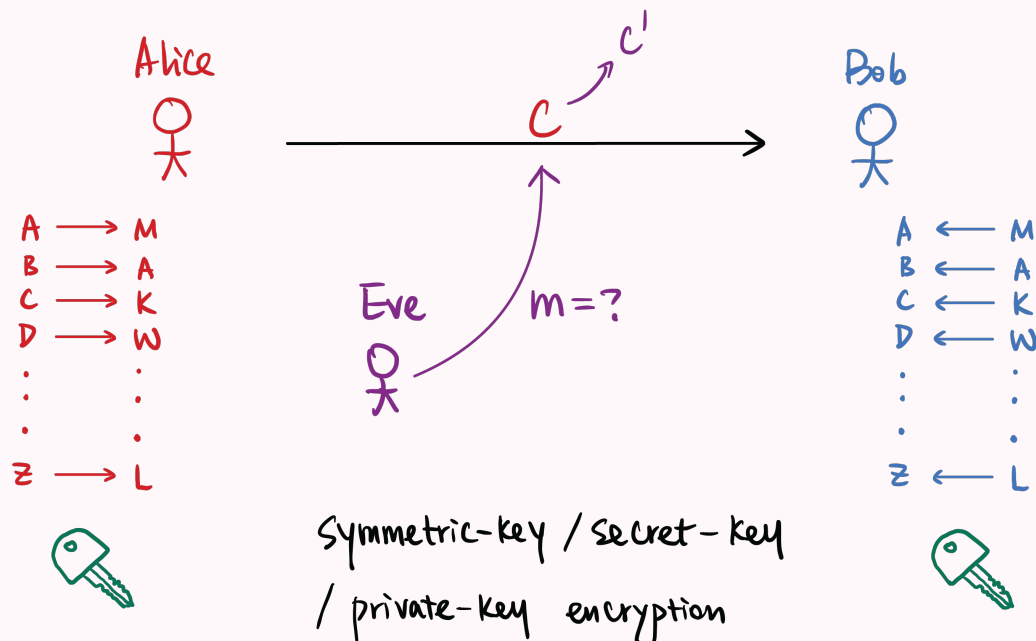


In this model, Eve is a weaker adversary, an *eavesdropper*. Eve can only see the message, not alter it.

#### 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.



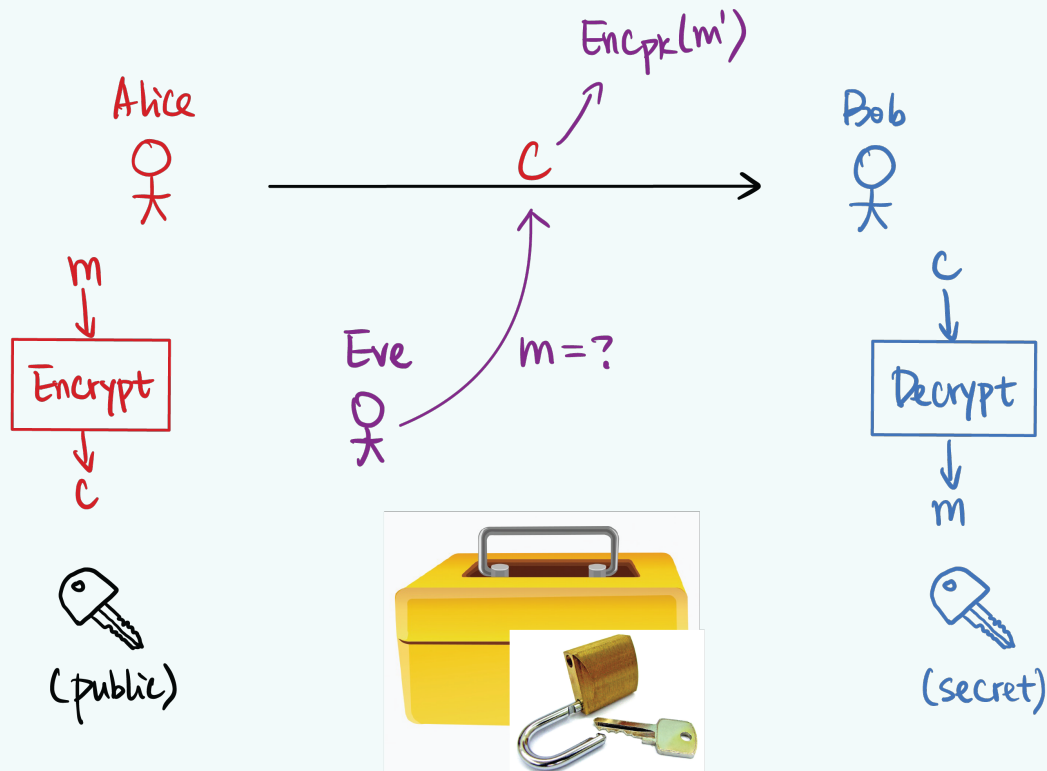
This scheme is not quite secure! *Why?*

Given a large enough text, you can apply frequency analysis and break the substitution cipher.

**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 somehow agree first on the same secret key.

**Definition 1.3 (Public-key Encryption)**

There is another primitive that is much stronger: public-key encryption. Bob generates both a *public* key and a *private* key, and then publishes his public key. You can consider a lock where you don't need a key to lock it<sup>1</sup>, and only Bob has the key to unlock it.

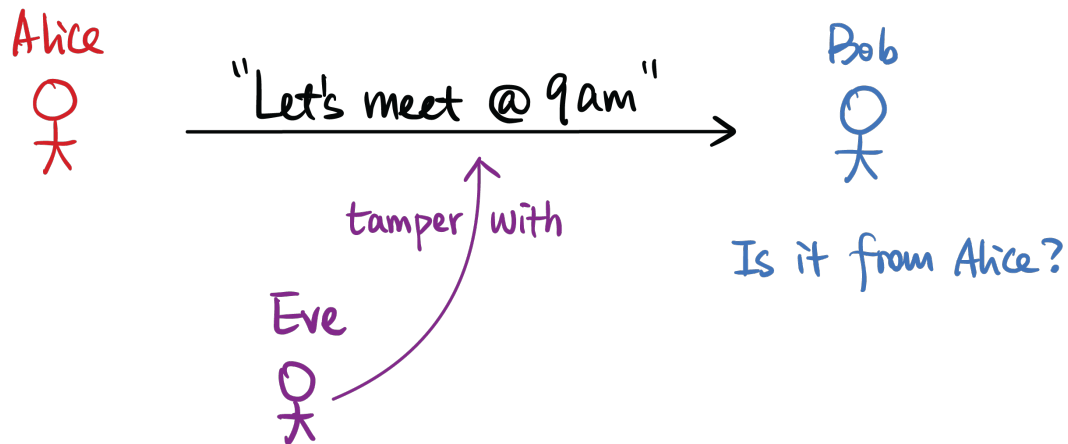


This is seemingly magic! Bob could publish a public key on his homepage, anyone can encrypt using a public key but only Bob can decrypt. *Stay tuned, we will see public-key encryption schemes next lecture!*

**§1.3.2 Message Integrity**

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

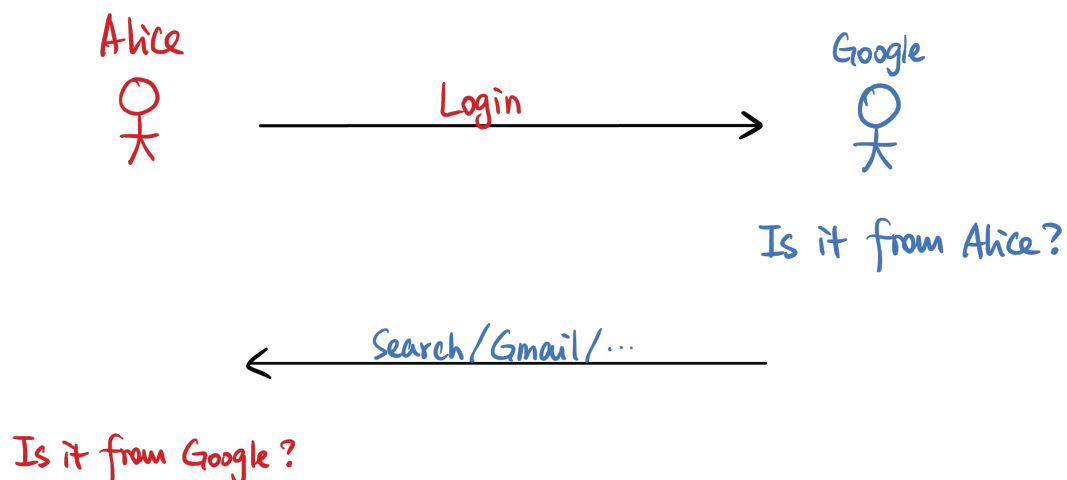
<sup>1</sup>You literally click it closed



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.



This property that we want is called message integrity.



### §1.3.3 Projects

#### *Projects Overview*

0. *Cipher* Warm-up, you will implement some basic cryptographic schemes.
1. *Signal* Secure Communication: how to communicate in secret.
2. *Auth* Secure Authentication: how to authenticate yourself.
3. *Yaos* Secure Multiparty Computation: we'll implement a way to run any function securely between two parties.
4. *Vote* Zero-Knowledge Proofs: we'll use ZKPs to implement a secure voting scheme.
5. *PIR* Private Information Retrieval (using homomorphic encryption, a form of post-quantum cryptography).

We'll now introduce the latter three projects!

### §1.4 Zero-Knowledge Proofs

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

For example, Alice may want to

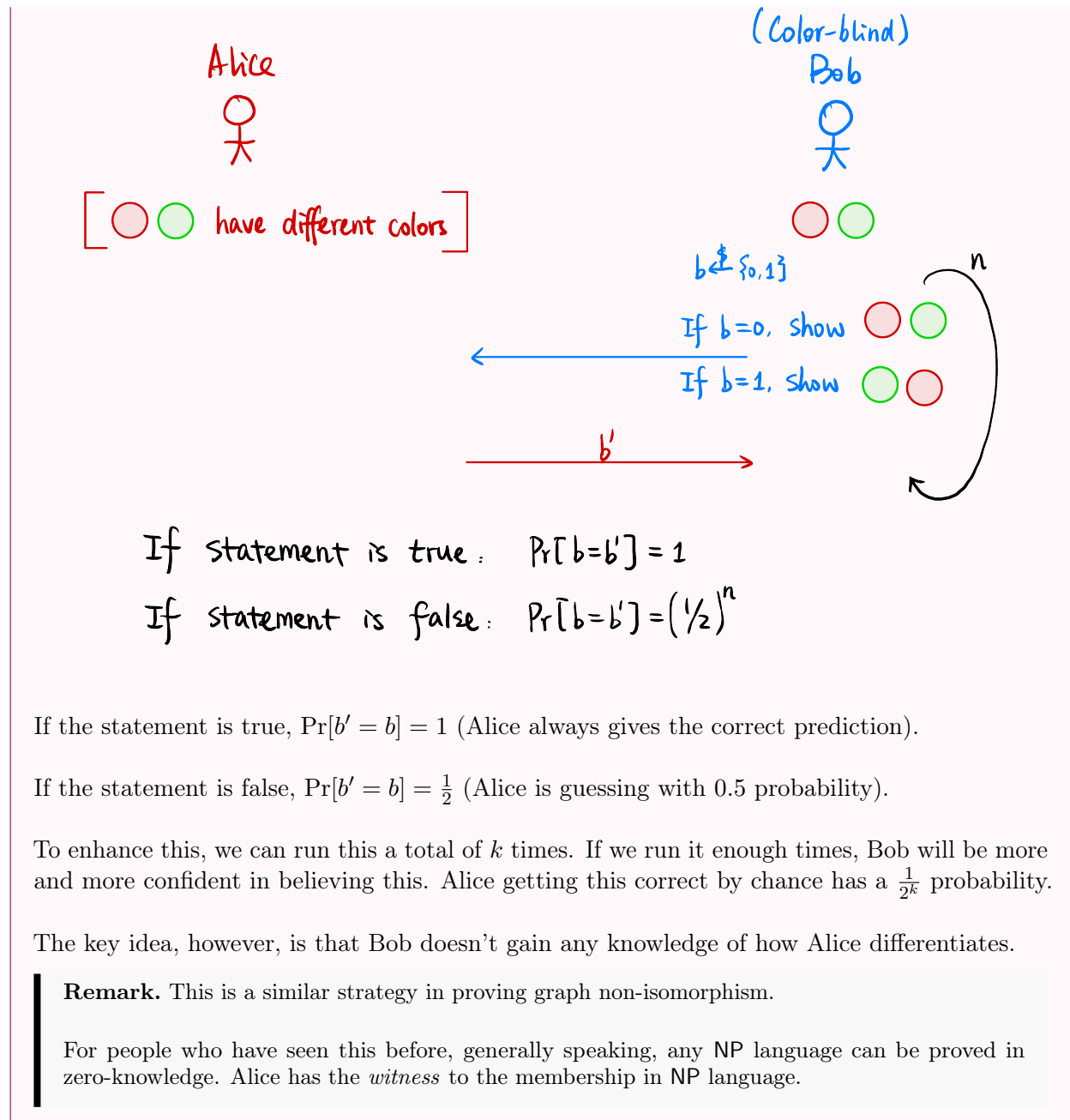
- Prove she knows the difference in taste between Coke and Pepsi without revealing how
- Prove that you have a bug in your code without revealing the bug
- She has the secret key for this ciphertext without revealing the plaintext
- Prove that she owns two different colored pens to her colorblind friend Bob

How is this possible? Let's examine this first scenario, which closely follows the red-green pens example from class *othered – greenballsproblem*.

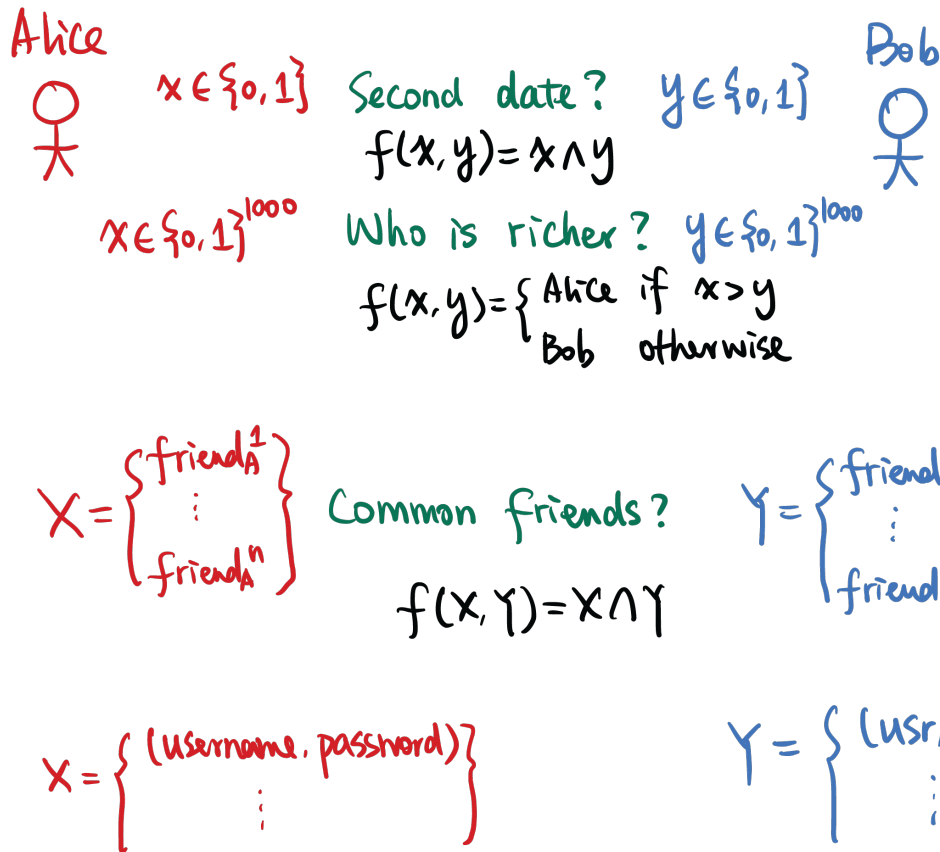
#### **Example**

Alice claims to have two different colored markers. She wants to prove this to her friend Bob, but there is an issue - Bob is colorblind!

Bob will randomly sample a bit  $b \xleftarrow{\$} \{0, 1\}$ , with  $b = 0$  being "Stay" and  $b = 1$  being "Swap". Bob will then either swap the order of the markers and show them to Alice, or keep them the same. Alice will give a guess  $b'$  of whether Bob swapped or not.



## §1.5 Secure Multi-Party Computation

**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  $f(x,y) = x \wedge y$ .

**Remark 1.4.** Couldn't a party still figure out how the other party feels? For example, if Bob's bit was 1 and the joint result was 0, Bob can *infer* that Alice's bit was 0.

This is, in effect, the best we can do. The ideal guarantee is that each party only learns any information they can infer from the *output* and their input. However, they should not learn anything more.

What are we trying to achieve here? We want to jointly compute some function, where each party has private input, such that each party only learns the output. They should not learn anything about other parties' inputs.

#### 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  $X = \{\text{friend}_A^1, \text{friend}_A^2, \dots, \text{friend}_A^n\}$  and Bob also has a set  $Y = \{\text{friend}_B^1, \text{friend}_B^2, \dots, \text{friend}_B^m\}$ . 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 of input records is revealed weaker than using a trusted third-party?

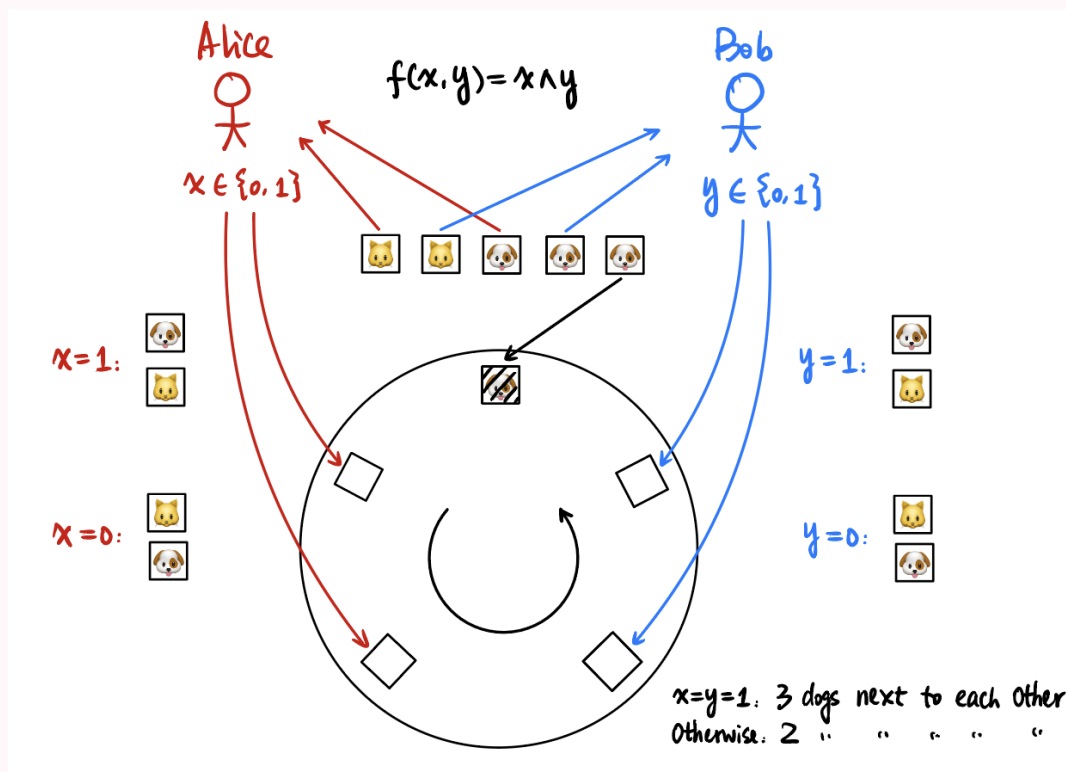
Yes, however in some cases (hospital health records), parties are legally obliged to keep data secure. We wish for security more than the secrecy of cardinality.

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  $x_1, \dots, x_n$  that jointly compute  $f(x_1, \dots, x_n)$  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 *dog* facing down. Alice and Bob each have identical *dog*, *cat* cards (each of the *dog* and *cat* cards are indistinguishable from cards of the same value). Note that our use of *dog*, *cat* is arbitrary, and could be any other set of two elements: 0, 1,  $x$ ,  $o$ , etc.



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

*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.*

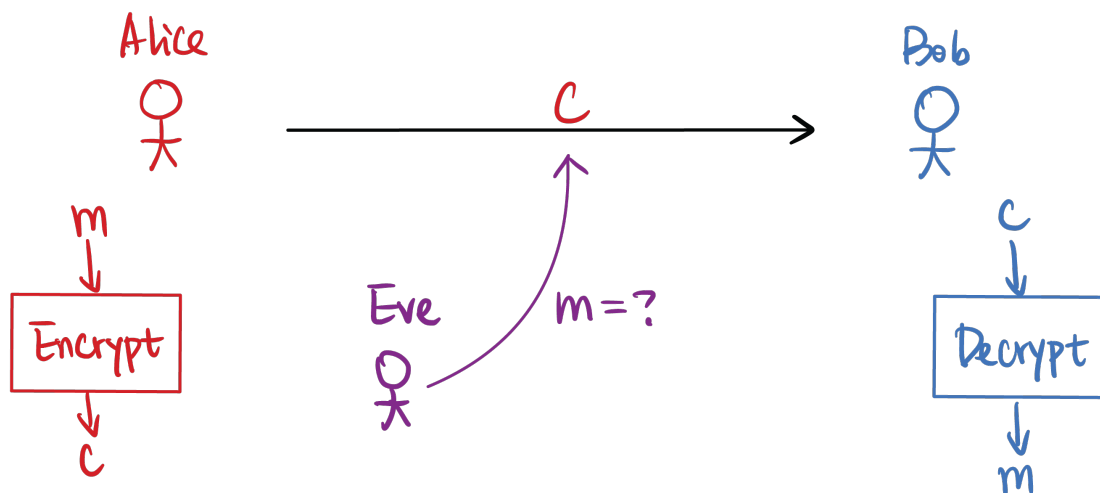
## §1.6 Fully Homomorphic Encryption

We'll come back to the secure messaging example.

Alice wants to send Bob a message. She encrypts it somehow and sends a ciphertext  $c_1 = \text{Enc}(m_1)$ . 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)

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

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

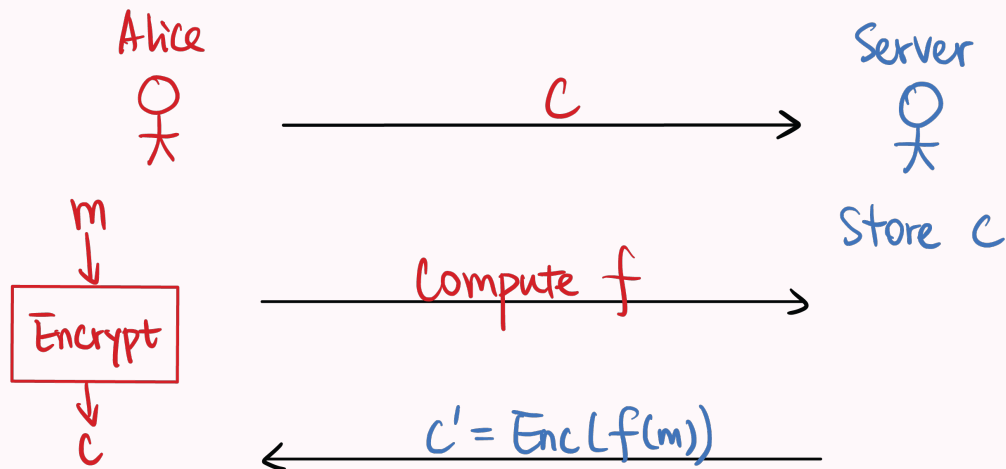


$$\begin{aligned} c_1 &= \text{Enc}(m_1) \\ c_2 &= \text{Enc}(m_2) \end{aligned} \Rightarrow \begin{aligned} c' &= \text{Enc}(m_1 + m_2) \\ c'' &= \text{Enc}(m_1 \cdot m_2) \end{aligned}$$

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. For example, she may want to search across that data without the server decrypting it (or sending her the entire encrypted DB). With homomorphic encryption, the server may be able to query and return an encrypted value to Alice without learning anything about the query itself.

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 which is a post-quantum secure!

## §1.7 Further Topics

We might cover some other topics:

- Differential Privacy
- Crypto applications in machine learning
- Crypto techniques used in the blockchain<sup>2</sup>

<sup>2</sup>One important techniques is Zero-Knowledge proofs, for example.

- LLM Watermarking

*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?*

Feel free to let us know on Ed!

## §1.8 Q & A

- *Do I need to have a crypto background?*

No!

- *Why C++*

Existing crypto libraries are mostly in C++ and most students have seen C/C++ in either cs33 or cs300. We did, however, consider implementing everything in Rust!

- *Class Participation*

Course attendance is expected for all students, since a large portion of the class structure relies on students answering questions in class. Even if you don't actively volunteer, try to think through the answer on your own! Students may join class via Zoom if they are remote-only, traveling, or sick.

- *What is the difference between CSCI 1515 and CSCI 1510, MATH 1580, or CSCI 1040?*

CSCI 1510 is essentially “theoretical cryptography.” It covers formal definitions and constructions and proofs. There is no coding, just proofs.

MATH 1580 considers crypto from the mathematical perspective. They try to understand some of the computational assumptions we assume from a mathematical standpoint. I.e. why is factoring hard to compute, and what is the best algorithm to compute it? In CS, we simply assume factoring is hard. MATH 1580 is more similar to number theory and group theory.

CSCI 1040 is a cryptography course for students without a computer science or math background. It is self described as "crypto for poets!"

CSCI 1515, on the other hand, takes a hands on approach to cryptography. This class is suitable for both students with limited cryptography exposure more interested in cryptography from a software engineering perspective, as well as students with a cryptography theory background looking to understand what makes an implementation efficient in practice.

This course **will not** be offered in Spring 2027.