



Cryptography

Cryptography

Crypto + graphy = secret + writing

- to make information secret, use a **cipher**, an algorithm that converts plain text to **ciphertext**, which is gibberish unless you have a **key** that undo the cipher.
- **encryption**: the process of making text secret
- **decryption**: the reverse process, unscrambling gibberish text to plaintext

Julius Caesar uses a **Caesar cipher**; he shifts all of the letters forward by three.

- A becomes D and "brutus" becomes "euxwxv".
- to decipher, you need both the algorithm and the shift number, which acts as the key. In this case, the key is 3.
- The Caesar cipher uses 26 keys and is easy to break.

Substitution Cipher

Caesar cipher is an example of a larger class of ciphers called **substitution cipher**.

- every letter is replaced by a different letter.
- one drawback of basic substitution cipher is letter frequency is preserved.
- the letter e is the most common letter in English. If cipher translates e to x, then x would be the most common letter in ciphertext.
- it was the breaking of a substitution cipher that led to the execution of Mary, Queen of Scots, in 1587, for plotting to kill Queen Elizabeth.

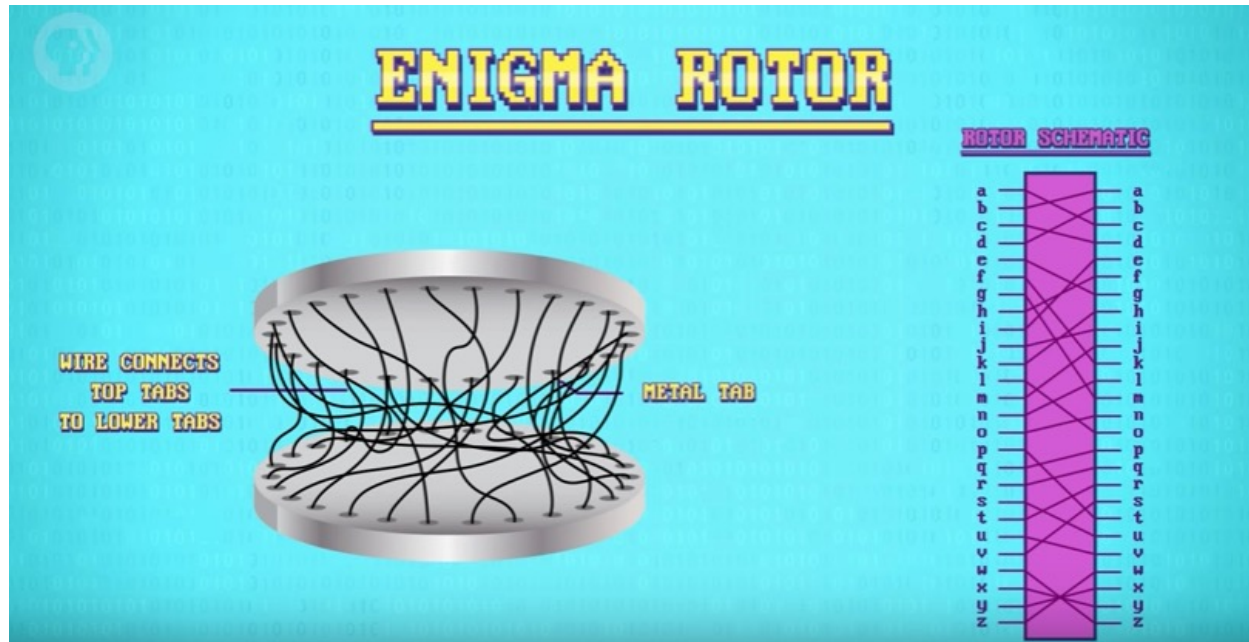


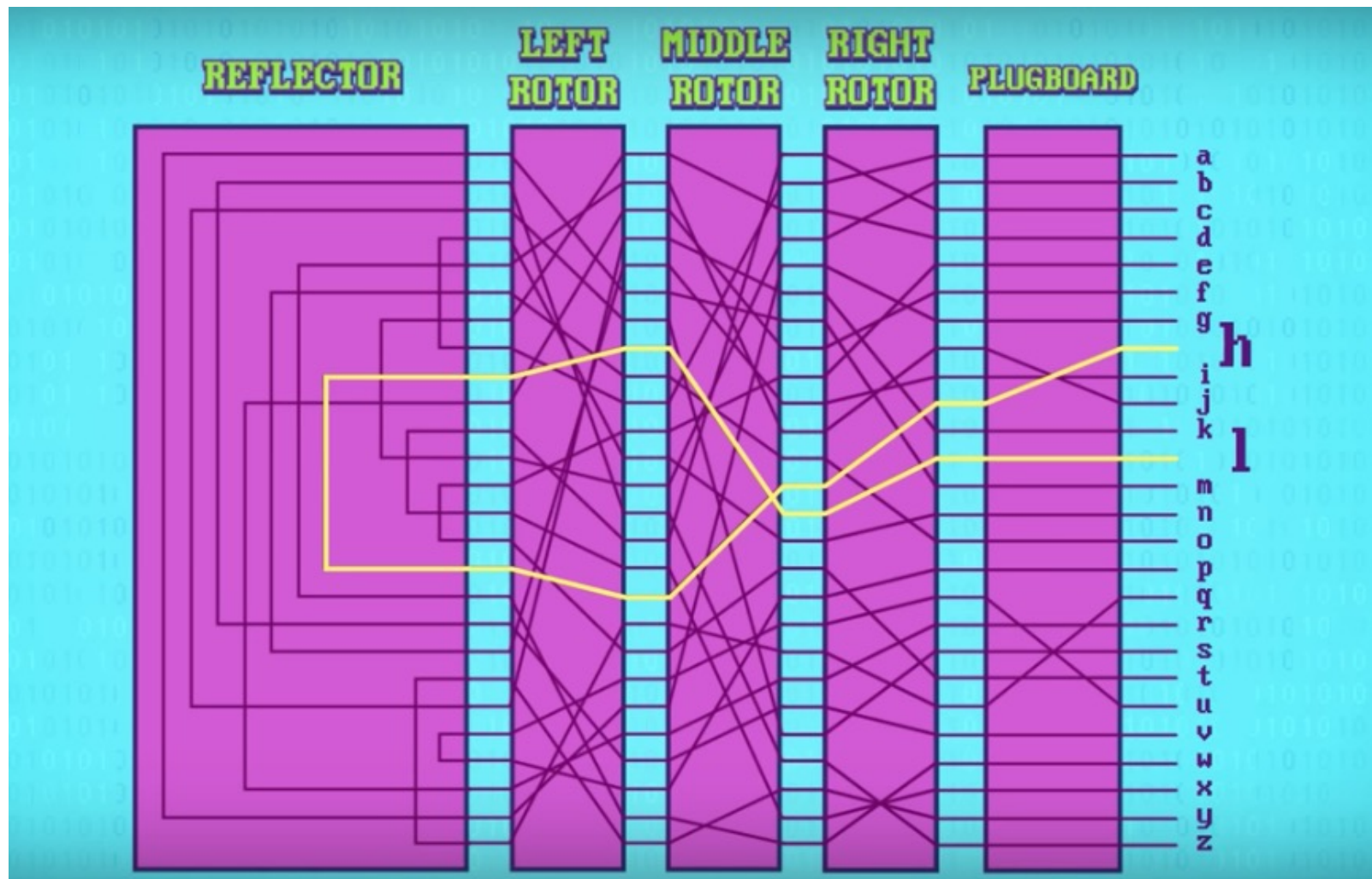
Enigma(optional)

In the 1900s, cryptography was performed using machines.

The German Enigma was an example of one such machine. It encrypt wartime communication.

The Enigma was a typewriter-like machine with a keyboard with the full alphabet and configurable rotors that were the key to the encryption.





The Enigma(optional)

To prevent it from being simple substitution cipher, after every letter, the rotor rotate by one, changing both the cipher and key.

- For example “AAA” might be encoded as “BDK”

The enigma was hard to crack but Alan Turing and his team at Bletchley Park were able to solve it and even automate the entire process!

One of Enigma’s flaw was every letter can’t be mapped into itself.

(for a movie version of this story, watch “The Imitation Game”, Benedict Cumberbatch.)



DES(optional)

One of the earliest widespread software ciphers was the **Data Encryption Standard(DES)** developed by NASA and IBM in 1977.

It was a 56-bit encryption and thus has 2^{56} keys which is 72 quadrillion(thousand trillions).

Back in 1977, no one has the computing power to brute force that many keys.

In 1999, powerful computers can begin to crack DES.

AES(optional)

In 2001, **Advanced Encryption Standard(AES)** was published

- uses up to 256 bits, brute force is virtually impossible.
- For example, even with 128 bits, if you use every computer on the planet, it'll take trillion of years to try every combination of keys. With 256 bits, it would take up about the age of the universe to crack.

AES chops data into 16-byte chunks, perform a series of permutations and substitutions and other operations to obscure the message, then repeat 10 or more times.

- No fancy math, unlike the other popular alternative, RSA.

AES is used everywhere.

- encrypting files on the Iphone.
- encrypting data over WiFi with WPAS and HTTPS protocol.

In October 2017, "Krack Attack" was leveraged against WPA2 by researchers, particularly its handshake protocol.

- The vulnerability was not with the AES algorithm but its implementation.

Symmetric Encryption

Cryptographic techniques so far rely on keys known by both sender and receiver. The sender encrypt message with a key and the receiver decrypt it with the same key.

Symmetric encryption: encryption technique that uses the same key for both encryption and decryption.

- For example, in the Caesar cipher, if encryption is shifting right by 3 letters, decryption is shifting left by 3 letters. Both uses the key = 3. Thus the Caesar cipher is a symmetric encryption technique.

In the old days, keys are shared physically or by voice.

- Germans use codebooks with daily settings for Enigma machines.

Today, server needs to send a key over the internet.

- making a purchase on Amazon, login on to Gmail, etc.
- keys can be intercepted.

Key Exchange(optional)

Solution is to use **key exchange**.

key exchange: an algorithm that allows two computer to agree on a key without sending one!

This is done using one-way functions.

A **one-way function** is a function that is easy to compute but hard to invert.

- it's easy to mix colors but hard to unmix, i.e., figure out the component colors used to get a mixed color.

We'll use painting to illustrate how this is done conceptually.

Key Exchange(optional)

Alice and Bob want to establish a shared secret key for encryption/decryption.
Each starts with a secret color.

**Eve is listening.
(Eve for "eavesdrops")**

**1. Alice's
secret
color**



**1. Bob's
secret
color**

Key Exchange(optional)

Eve is listening.

**1. Alice's
secret
color**



**2. Mix in
public
color**



**1. Bob's
secret
color**



**2. Mix in
public
color**



Both mix the public color with their secret color.

Key Exchange(optional)

Eve is listening.

**1. Alice's
secret
color**



**2. Mix in
public
color**



**3. Send
to Bob**

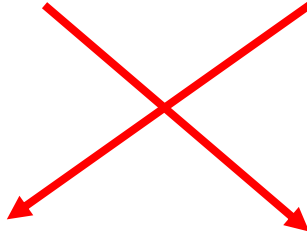


**1. Bob's
secret
color**

**2. Mix in
public
color**



**3. Send
to Alice**



Then send it to each other publicly. Then what?

Key Exchange(optional)

Eve is listening.

**1. Alice's
secret
color**



**2. Mix in
public
color**



**3. Send
to Bob**



**4. Mix in
secret
color**



**1. Bob's
secret
color**

**2. Mix in
public
color**



**3. Send
to Alice**



**4. Mix in
secret
color**



Each then adds in his/her secret color to obtain a shared secret color(key).
The key is then used for symmetric encryption.

Diffie-Helman(optional)

Diffie-Helman key exchange uses modular exponentiation as its one-way function.

Two positive integers x and y are **congruent modulo n** , denoted $x \equiv y \pmod{n}$, if $x \% n == y \% n$.

Note: $a \% b$ is the remainder of a divided by b .

For example, $7 \equiv 2 \pmod{5}$ since both 2 and 7 when divided by 5 give a remainder of 2.

- $13 \equiv 25 \pmod{2} \equiv 1 \pmod{2}$ (Thus 13, 25 and 1 are all congruent modulo 2)
- $3 \equiv 15 \pmod{12} \equiv 27 \pmod{12}$ (Thus 3, 15 and 27 are all congruent modulo 12)

When performing math operations, we reduce modulo n so that the result is a remainder from 0 to $n-1$.

- $3^5 \pmod{31} = 243 \pmod{31} = 26 \pmod{31}$

Diffie-Helman (optional)

Modular exponentiation is a one-way function because it is easy to raise a power but hard to invert.

For example, it is easy to compute $3^5 = 26 \bmod 31$. (One line of Python code.) But it's hard to find x such that $3^x = 26 \bmod 31$ if the base and modulo are very large integers (hundreds of digits long).

- This is called the **discrete logarithm problem**.

The base that is used for the modulo is a large prime (hundreds of digits long).

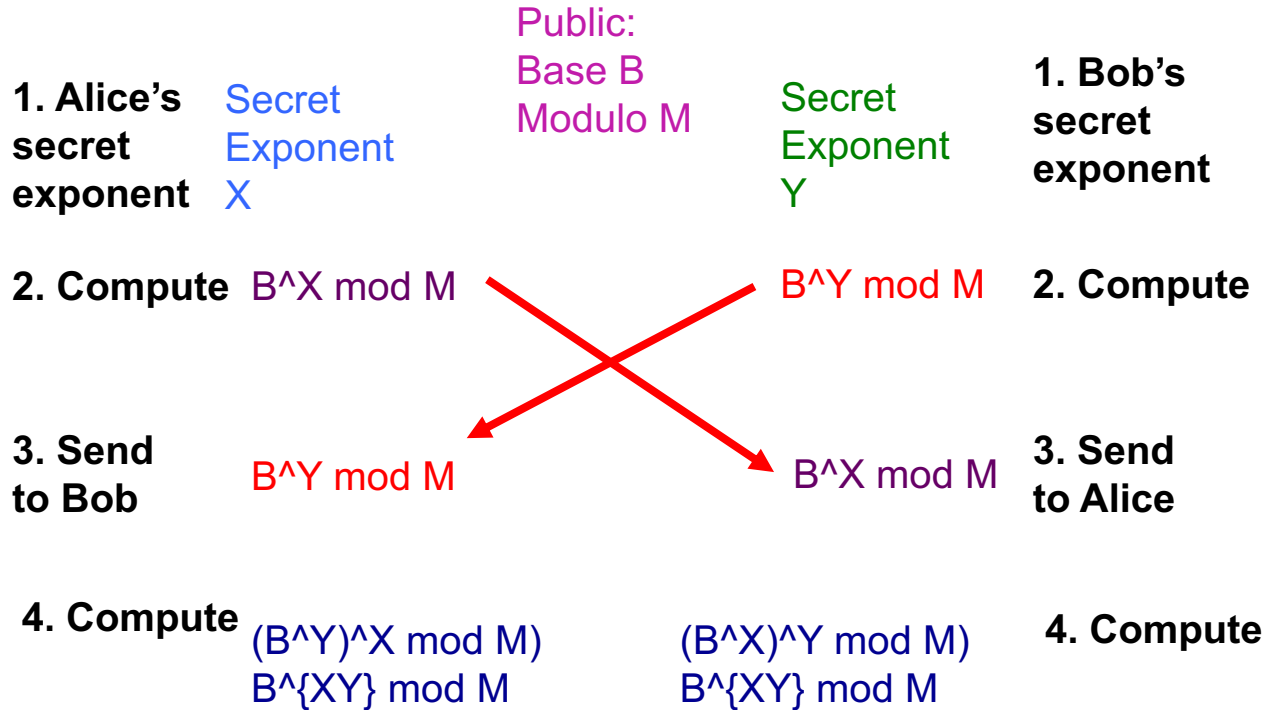
Solve this:

$$298742039473927847494324^x = 7646787876034389839493034 \bmod 85620542843242379862561534532898765235$$

Diffie-Helman is an algorithm that uses this one-way function of modular exponentiation to do key exchange. The process is similar to the painting example earlier.

Diffie-Helman Key Exchange (optional)

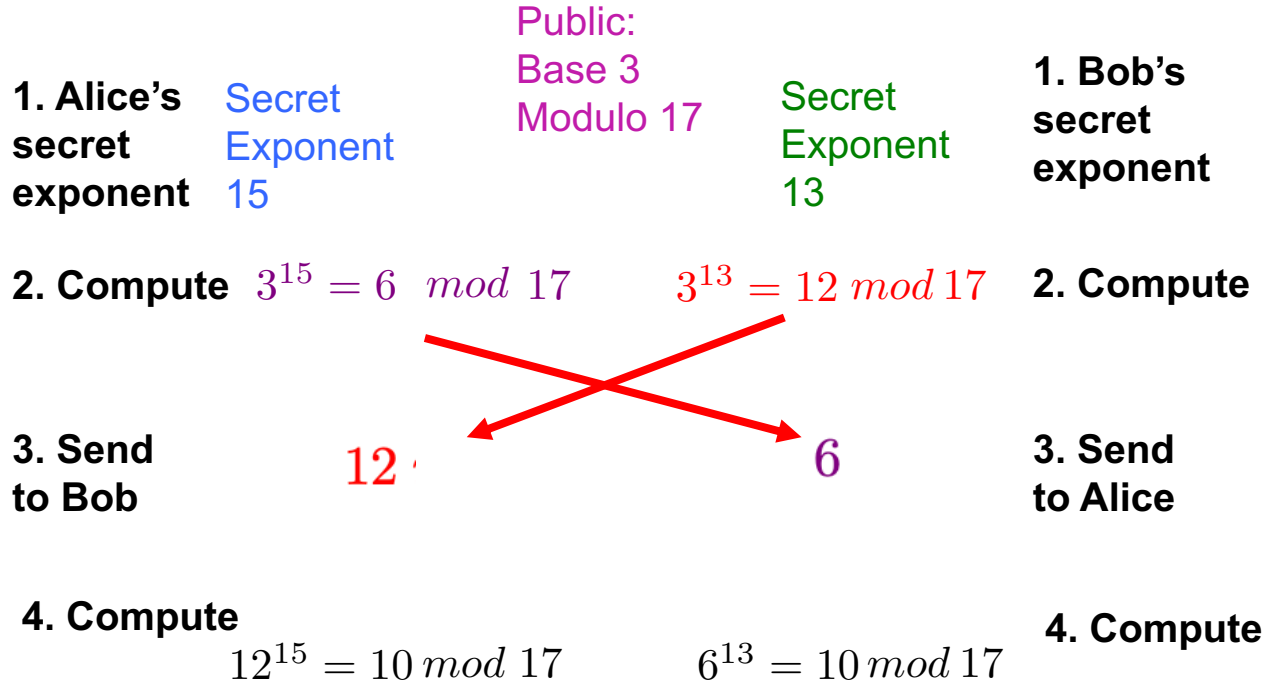
Eve is listening.



$B^{XY} \bmod M$ is the shared key.

An Example (optional)

Eve is listening.



Thus, the secret shared key is 10. For example, key = 10 can be used in the Caesar cipher by shifting/unshifting by 10

Diffie-Helman (optional)

One shortcoming of the Diffie-Helman key exchange is that it uses communication overhead to establish a shared key.

- Alice and Bob send back and forth different results of their calculations.

Another shortcoming is if Alice needs to send messages to different people, she needs to exchange different keys.

- if she is a bank for example, she needs thousands of distinct keys to communicate with each person.
- she would then need to send thousands of messages just to establish these secret keys with each person.

Asymmetric Encryption

James Ellis, a British Engineer, was working on a non-secret encryption in 1977. His idea is clever yet simple: lock and unlock are inverses. Use this as a one-way function!

- Easy to lock, but hard to unlock.

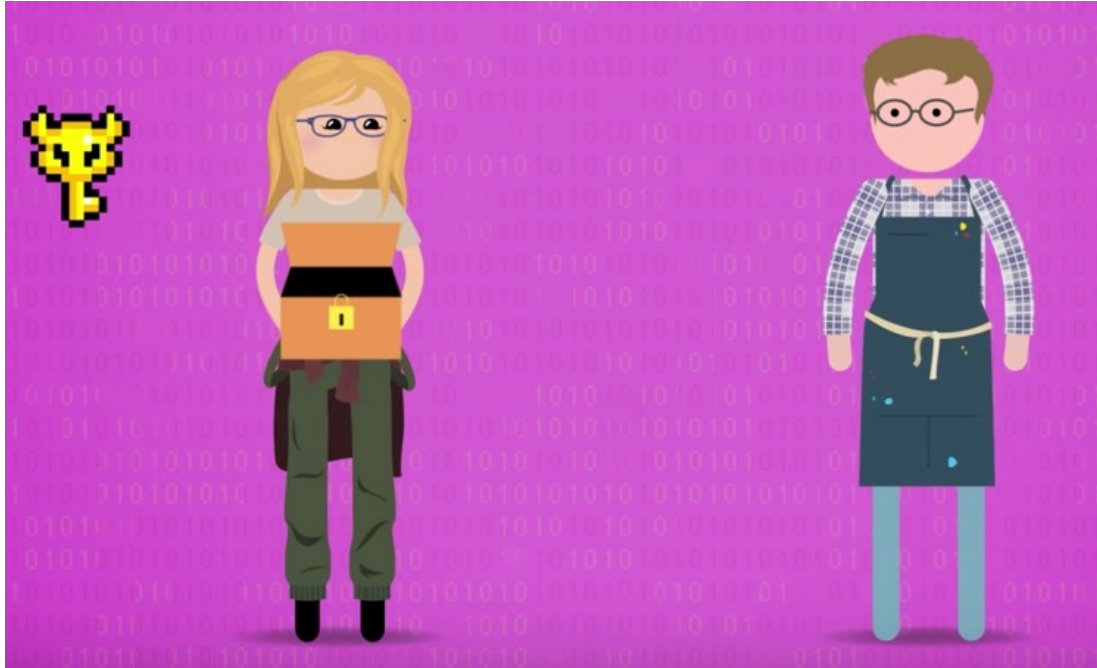
Ellis' idea is an example of **asymmetric encryption(public-key cryptography)**: encryption which uses a public and a private key. One for encryption and one for decryption.

Someone can encrypt a message with the public key but only the recipient with their private key can decrypt.

- the public key is use for encryption, the private key for decryption.
- the reverse is also useful, private key to encrypt and and public key to decrypt(see digital signatures later in these slides)

The most famous example of **asymmetric encryption is RSA** named after their inventors Rivest, Shamir and Adleman. The math behind RSA is beyond scope of the class. There's an optional video that you can watch at the end of the slides.

Alice has a private key and a lockbox(public key).
She wishes to receive an encrypted message from Bob.
Alice sends Bob the lockbox(public key).



Bob put his message inside the lockbox.
Lock it(one-way function, easy to lock)
Sends it back.



Alice can open it with her private key and receives the message. (even if the lockbox is intercepted, it cannot be opened without the private key)

If Alice is a bank, she can duplicate the lockbox(public key) and sends them to everyone she wishes to receive encrypted messages.

She only has to manage one private key.



Digital Signatures

A public digital key can encrypt something that can only be decrypted by a private digital key.

The reverse is also possible: encrypting with the private key something that can be decrypted with a public key.

- Use for **digital signatures**
- Server encrypts data with private key.
- Anyone can decrypt it using the server's public key.
- This acts as an unforgeable signature that only the owner, using the private key, can encrypt.
- This proves that you're getting data from the right server/website, not an imposter.
- This is how your browser knows that you are at the correct bankamerica.com and not a fake bank of America website. The browser knows Bank of America's public key and can verify. If it's successful, it gives you the green padlock confirmation.



Certificate authorities issue **digital certificates** that validate the ownership of encryption keys used in secure communications and are based on a trust model. For example, Google is a certificate authority. A website can apply for a digital certificate from Google to validate their ownership of their website (green padlock you see when you go to an https (secure http) website).

We can understand how digital signatures work by looking at **hash functions**.


Hash Functions(optional)

In the abstract, a **cryptographic hash function** is a mathematical function that takes input data of **any** size, performs an operation on it, and returns output data of a **fixed** size.

Since data(file, images, audio, video) can be encoded into a string of 0's and 1's, a hash function is simply a function $f(x)$ which accepts a string input of any length and returns a fixed-length string output. Here's an implementation of such a hash function using the popular SHA256 algorithm(outputs 256-bit strings):

```
import hashlib  
  
def hash(mystring):  
    hash_object= hashlib.sha256(mystring.encode())  
    return hash_object.hexdigest()
```

```
print(hash("hello"))  
print(hash("Hello"))
```

 a small change to a message should change
the hash value extensively

Output:

```
'2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824'  
'185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969'
```

Properties of Hash Functions(optional)

The ideal cryptographic hash function has the following main properties:

- it is deterministic meaning that the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message that yields a given hash value i.e. to reverse the process that generated the given hash value. This problem is hard to solve:

Given

$y = \text{'e3824dba5fb0a30e26e8352ac5b9e29e1b161e5c1fa7425e73043362938b9829'}$

It is hard to find x so that $\text{hash}(x) = y$.

- it is infeasible to find two different messages with the same hash value
- a small change to a message should change the hash value so extensively that a new hash value appears uncorrelated with the old hash value

Digital Signature

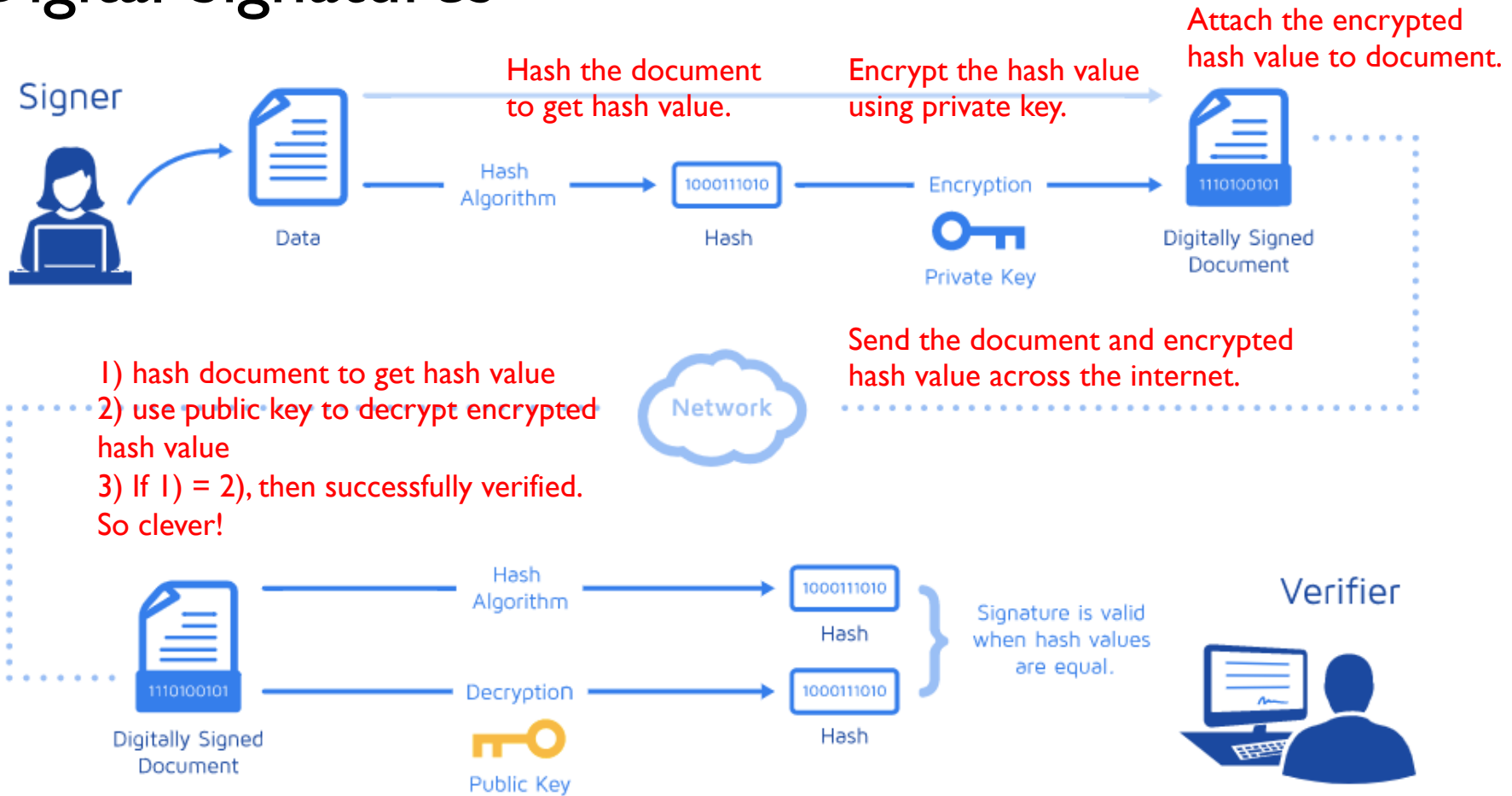
When a signer electronically signs a document, the signature is created using the signer's private key, which is always securely kept by the signer.

The mathematical algorithm acts like a cipher, creating data matching the signed document using the hash function and encrypting that data.

The resulting encrypted data is the **digital signature**. The signature is also marked with the time that the document was signed.

If the document changes after signing, the digital signature is invalidated. The image in the next slide explains this process.

Digital Signatures



How HTTPS Work

The “key” parts of modern cryptography:

- symmetric encryption
- key exchange
- public-key cryptography



When you connect to a secure website, such as Facebook, using HTTPS that has the green padlock, you know that cryptography was used to encrypt data.

When information is sent over regular HTTP, the information is broken into packets of data that can be easily “sniffed” using free software. This makes communication over the an unsecure medium, such as public Wi-Fi, highly vulnerable to interception.

In fact, all communications that occur over HTTP occur in plain text, making them highly accessible to anyone with the correct tools, and vulnerable to on-path attacks.

Technically speaking, HTTPS is not a separate protocol from HTTP. It is simply using TLS/SSL encryption over the HTTP protocol.

How HTTPS Work

Here's how HTTPS work:

- Youtube receives a **digital certificate** from a **certificate authority(CA)** which includes a public and private key. The private key lives on Youtube's server. The public key is available for anyone who wants to access Youtube.
- When a user connects to youtube.com, Youtube will send over its SSL certificate which contains the public key necessary to start the secure session. How do we know that this certificate is authentic?
- Youtube's certificate authority is Google. Google encrypts this certificate with its private key.
- Your browser which has access to secure information from certificate authorities uses Google's known secure public key and decrypt this certificate. If the public key successfully decrypt the certificate, then your browser knows that this is the real Youtube server. The greenpad lock is the indicator for verification process.
- If Youtube is down, a hacker can't create a clone of Youtube without a valid certificate.

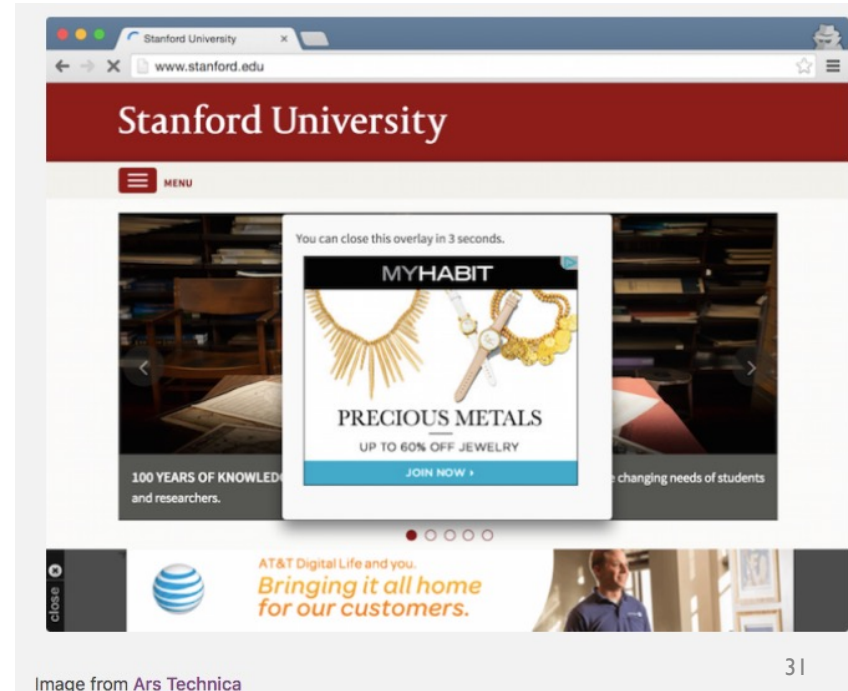
How HTTPS Work(optional)

In websites without HTTPS, it is possible for Internet service providers (ISPs) or other intermediaries to inject content into webpages without the approval of the website owner.

This commonly takes the form of advertising, where an ISP looking to increase revenue injects paid advertising into the webpages of their customers.

Unsurprisingly, when this occurs, the profits for the advertisements and the quality control of those advertisements are in no way shared with the website owner.

HTTPS eliminates the ability of unmoderated third parties to inject advertising into web content.



Cryptocurrency(optional)

A **cryptocurrency** (or crypto currency) is a digital asset designed to work as a medium of exchange that uses cryptography to secure its transactions, to control the creation of additional units, and to verify the transfer of assets.

cryptocurrency:

- decentralized currency
- Bitcoin was the first, established in 2009. Since then, other coins have been established, called altcoins.
- Bitcoin was invented by Satoshi Nakamoto.(Even today, no one knows who he is.)
- Uses public-key cryptography to make, secure transactions, verify ownerships, etc...
- Security is guaranteed by a concept known as "proof of work". Bitcoin miners are rewarded with bitcoins for this proof of work.
- Uses advanced math(elliptic curves) to do encryption.

Here's a reference on how bitcoins work:

<https://www.youtube.com/watch?v=Lx9zgZCMqXE>

Some Supplementary Videos(Optional)

1)Read and reread these lecture notes.

2)Watch(optional):

a)How bitcoin works(non technical)

<https://www.youtube.com/watch?v=l9jOJk30eQs>

3)Optional:

a)How RSA works. This is technical with some math but still accessible.

https://www.youtube.com/watch?v=wXB-V_Keiu8

b)How bitcoin works under the hood. Technical but still accessible. May need to rewatch, rewind several times.

<https://www.youtube.com/watch?v=Lx9zgZCMqXE>

c)Most of this lecture's material including some images come from PBS Digital Video on Cryptography.

<https://www.youtube.com/watch?v=jhXCTbFnK8o>

References

Part of this lecture is a recap of the following an episode from PBS Crash Course in Computer Science series.

PBS Crash Course in Computer Science. Cryptography. Retrieved from <https://www.youtube.com/watch?v=jhXCTbFnK8o>