Asymmetric Encryption and Hashing

Cybersecurity

Cryptography Day 2

Class Objectives

By the end of today's class, you will be able to:

Calculate the required number of symmetric and asymmetric keys based on the number of people exchanging secure messages.

Use GPG to generate keys, and encrypt and decrypt private messages.

Use hashes to validate the integrity of data.

Use digital signatures to validate the authenticity of data.

Cryptography Review

Activity: Cryptography Refresher

In the activities today, you will continue your role of security analyst at Hill Valley Police Department.

In this review activity, you must encrypt and decrypt a message using OpenSSL.

Suggested Time:

20 Minutes

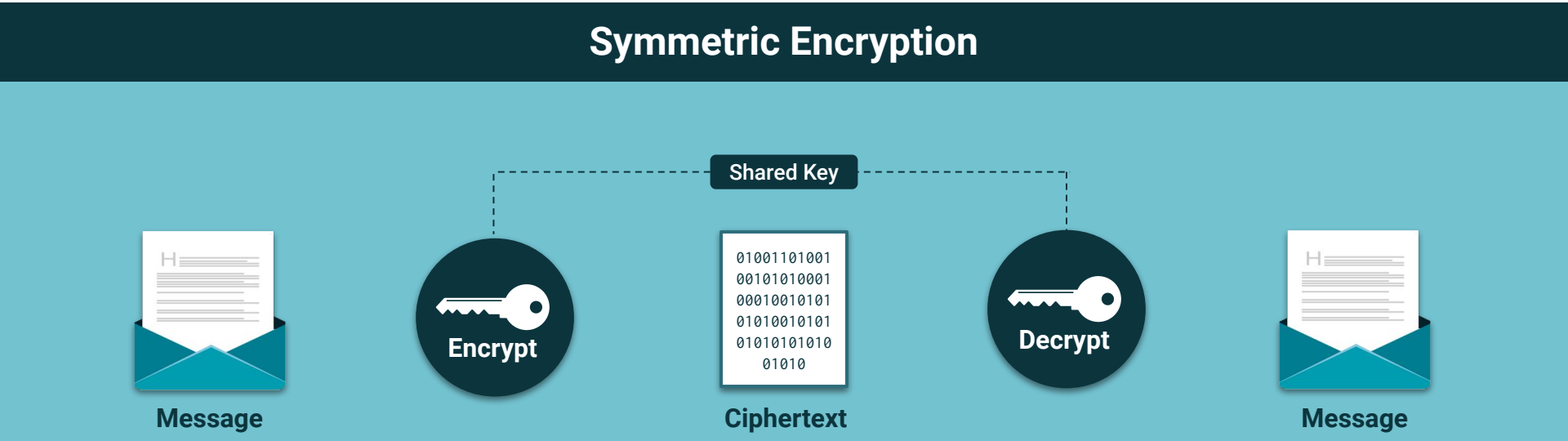
Time’s Up! Let’s Review.

Questions?

Key Management and Exchange

Key Management and Exchange

In the previous activity, we used **symmetric key encryption**.
While its benefits include speed, efficiency, and simplicity of use, symmetric encryption also has its disadvantages.



Disadvantage One: Secure Key Exchange

As more individuals are included in the key exchange, there needs to be a key for each combination of individuals.

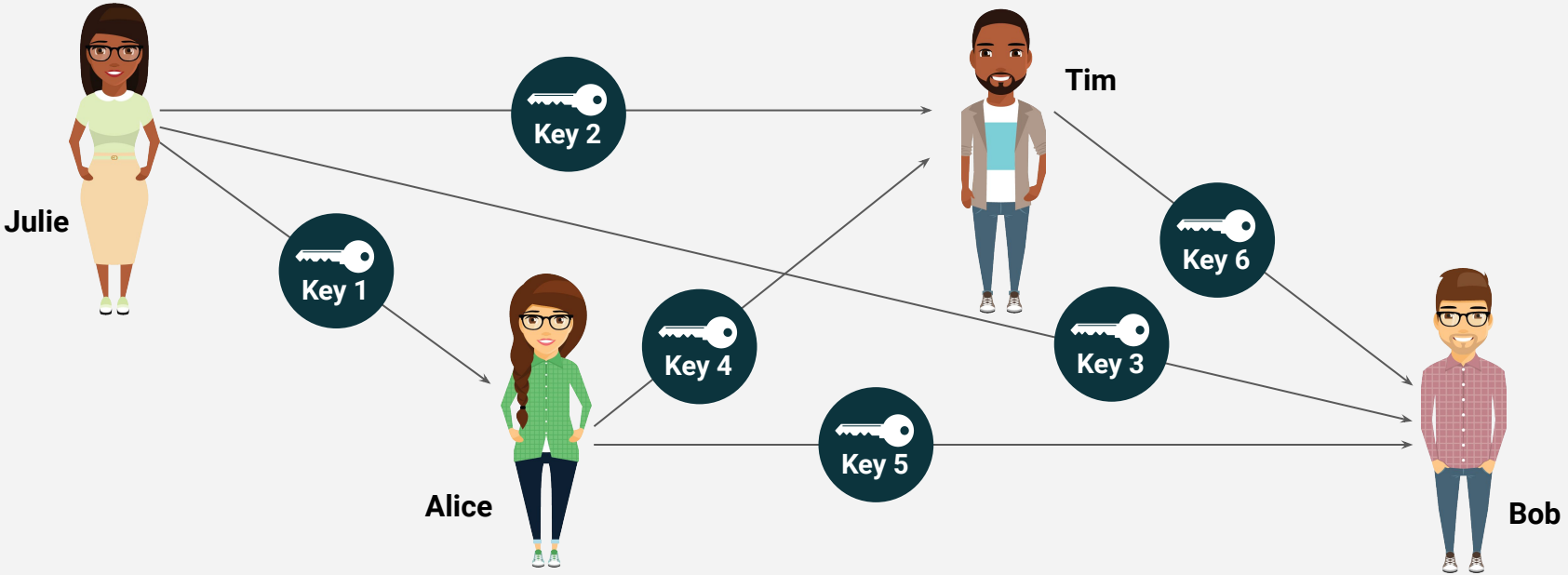
Offline Exchange (Out-of-Band Exchange)

Can include calling the recipient and reading the key, or mailing the key.

Diffie–Hellman Key Exchange

Uses complex mathematical principles to create a shared secret key between two parties over a public channel.

Disadvantage Two: Key Management



This is a small example. Since organizations often need to securely communicate with large numbers of people, the amount of symmetric keys can quickly become overwhelming.

Disadvantage Two: Key Management

Count of symmetric keys = (N * (N-1))/ 2

For an organization with 7 employees

```
(7 * 6) / 2 = 42/2 = 21
```

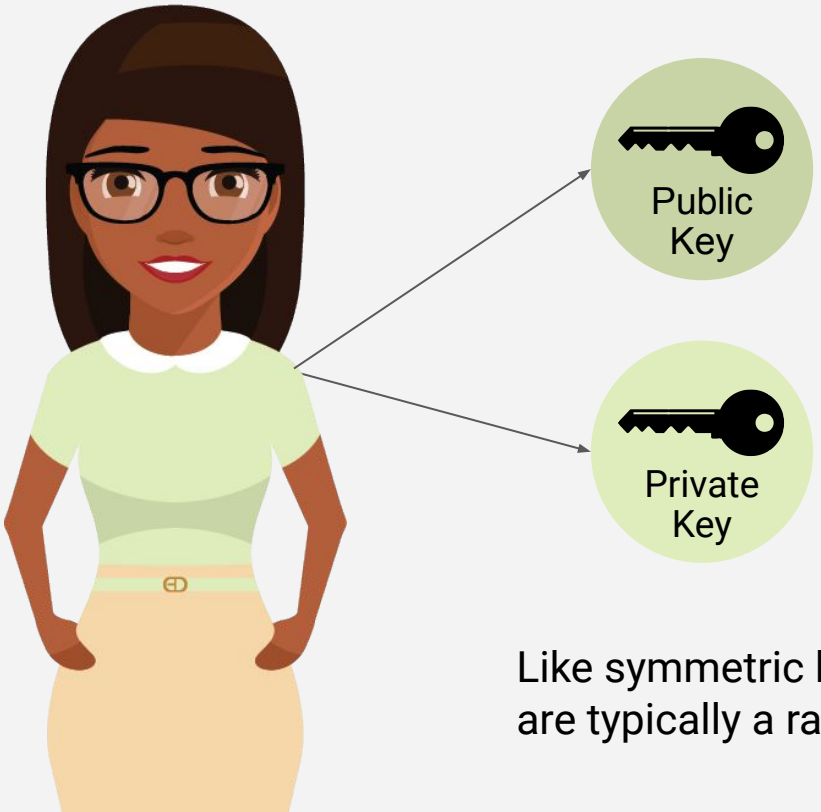
For an organization with 1,000 employees

```
(1000 * 999) / 2 = 499,500
```

These disadvantages can be addressed with asymmetric key encryption.

Asymmetric Key Encryptions

Each individual possesses a two-key pair:



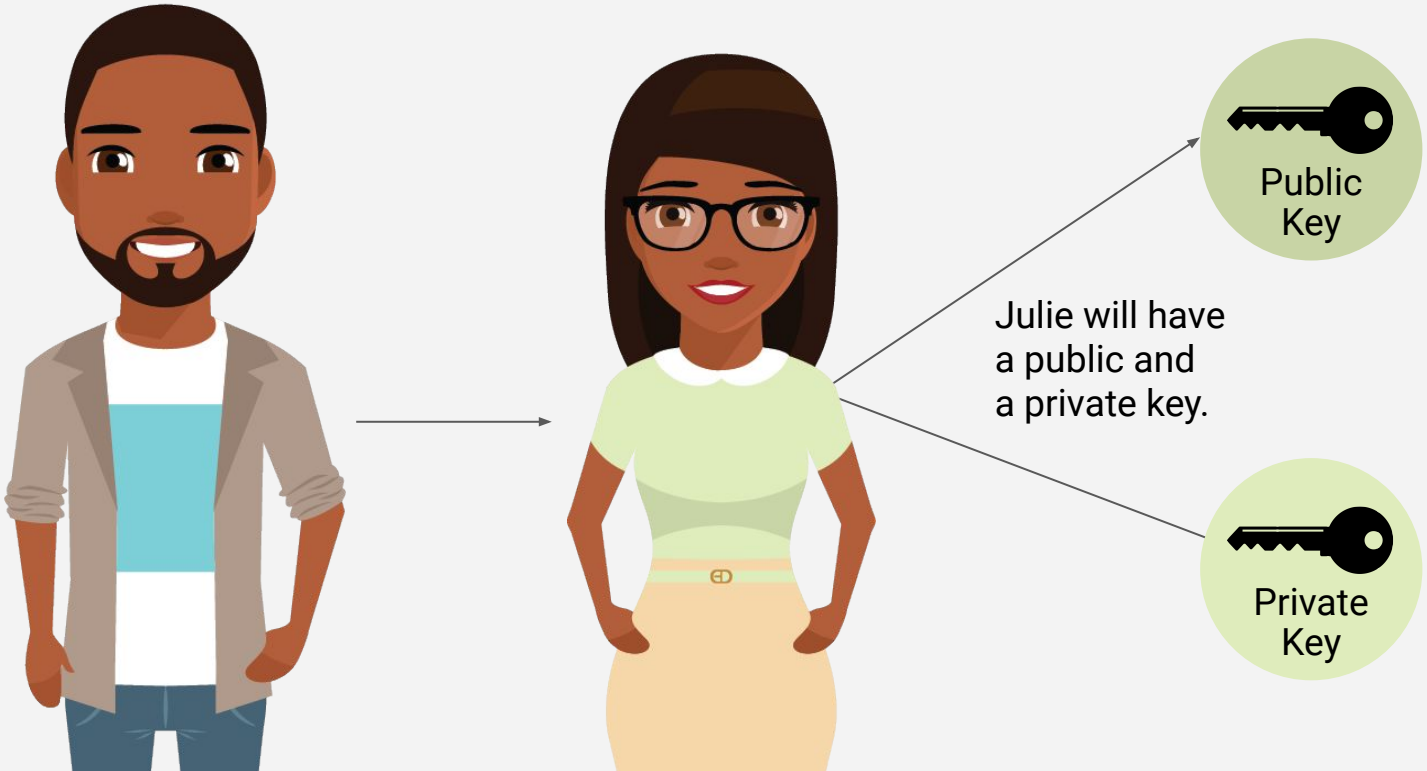
Public keys are public and accessible to all.

Private keys are kept secret and can affect confidentiality of messages if exposed.

Like symmetric keys, these public and private keys are typically a random alphanumeric string.

Asymmetric Key Encryptions

Scenario: Tim wants to send Julie his bank account number using asymmetric key encryption.



Asymmetric Key Encryptions

Scenario: Tim wants to send Julie his bank account number using asymmetric key encryption.

Tim creates a **plaintext message** containing his bank account number.

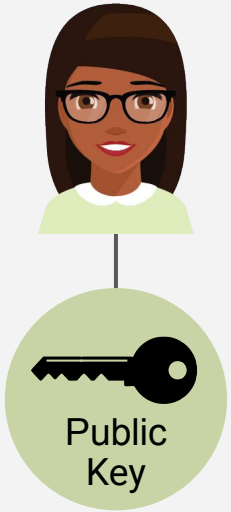
Tim goes to Julie's website, gets her **public key,** and uses it to encrypt his message.

Asymmetric Key Encryptions

Scenario: Tim wants to send Julie his bank account number using asymmetric key encryption.

Tim sends the encrypted message to Julie.

This can be sent over email or Slack without fear of interception.

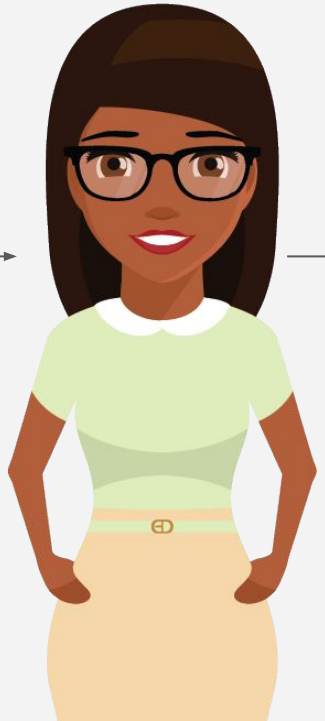


The message can only be decrypted by Julie with her matching private key.

Asymmetric Key Encryptions

Scenario: Tim wants to send Julie his bank account number using asymmetric key encryption.

LXFOPVE
FRNHR#4
&/j2



Private Key

Julie receives Tim's encrypted message and decrypts it using her matching private key.

Bank Account #

Julie can now access Tim's bank number.

Key Management

While symmetric key count grows exponentially larger as the number of employees increase, asymmetric count stays proportionate and manageable with increases.

For a company with twelve employees:

Symmetric

$(12 * 11) / 2 = 66$

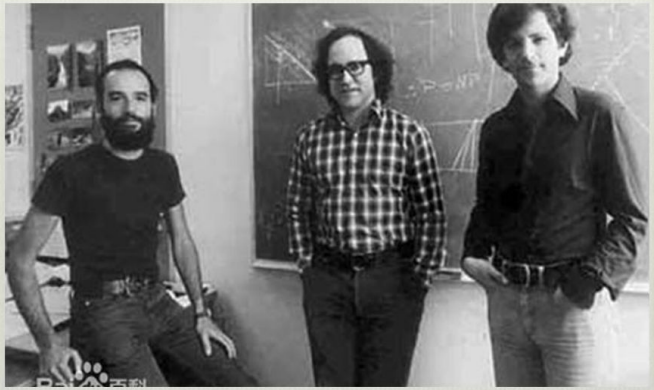
Asymmetric

$(12 * 2) = 24$

RSA

Asymmetric encryption uses an algorithm called RSA, introduced in 1977 and named after the last names of its creators.

Ron Rivest, Adi Shamir and Leonard Adleman

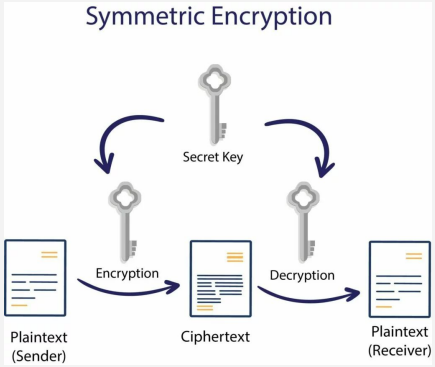


*Popularly known as **Rivest-Shamir-Adleman (RSA) algorithm,** it was built after **Whitefied Diffie, Martin Helman** and **Ralph Merkle** introduced public key cryptography in **1976**. **RSA** is an **asymmetric cryptography** popularly known as **public key cryptography.***

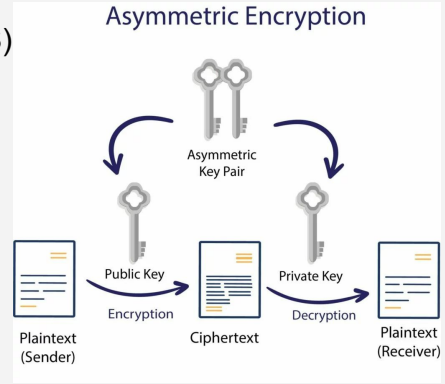
(Source: Devyani Vij, Medium)

RSA is the asymmetric algorithm standard used around the world.
It works by employing the complexity of factoring large numbers.

Symmetric vs Asymmetric Encryption



Advanced Encryption Standard (AES)
Blowfish
Twofish
Rivest Cipher (RC4)
Data Encryption Standard (DES)



Elliptic Curve Digital Signature Algorithm (ECDSA)
Rivest-Shamir-Adleman (RSA)
Diffie-Hellman
Pretty Good Privacy (PGP)

Change (N)ame, (E)mail, or (O)kay/(Q)uit?

Faster
Not Scalable
Uses the same key for both encryption and decryption (secret Key)
Data Storage
Used for bulk encryption because it is much faster than asymmetric cryptography

Slower
Highly Scalable
Public key for encryption and a private key for decryption.
Secure Web Browsing, Secure Web App usage
Digital Signing

Source: https://www.encryptionconsulting.com/education-center/symmetric-vs-asymmetric-encryption/

[Optional] Activity: Optimizing with Asymmetric Public Keys

In this activity, you will compare the key counts required for communicating with asymmetric and symmetric key cryptography.

Suggested Time:

Complete After Class

Applying Public Key Cryptography with GPG

GNU Privacy Guard (GPG)

GPG is a:

Command-line tool

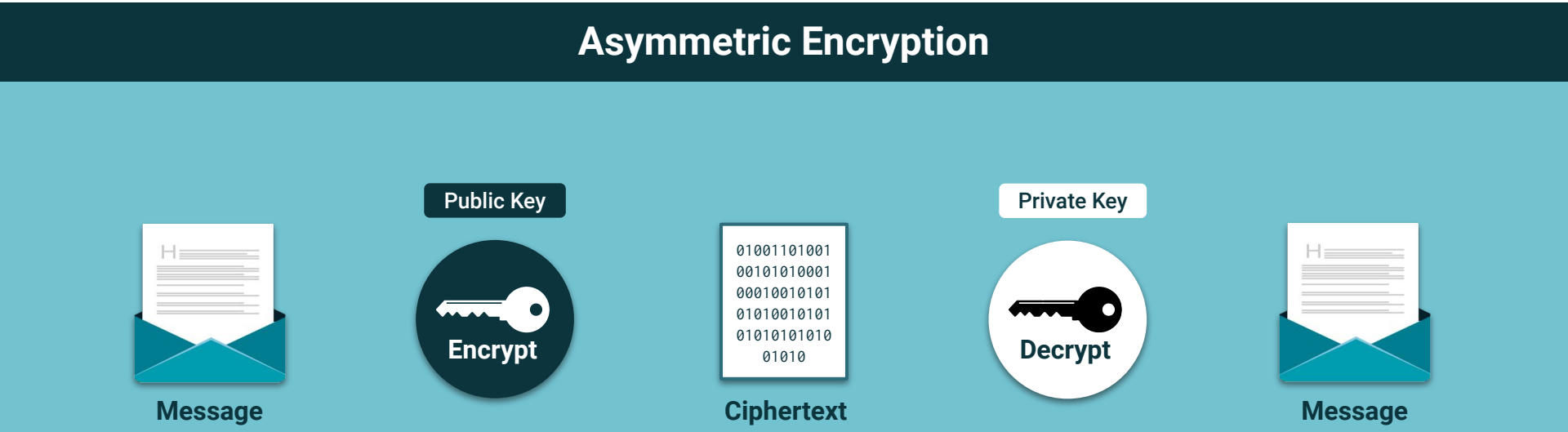
encryption, and decryption of asymmetric key cryptography.

Free program

Available on many Linux distributions that run symmetric and asymmetric encryption algorithms.

Applying Public Key Cryptography with GPG

In the following demonstration, we will use the bank account scenario with GPG to create a key pair for asymmetric encryption and decryption.



Asymmetric Key Encryptions

Creating the Pair Key

(1) gpg --gen-key

(2) Change (N)ame, (E)mail, or (O)kay/(Q)uit? Press O to confirm your information

```
gpg: key D81710193A5FC56A marked as ultimately trusted
gpg: directory '/home/instructor/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/instructor/.gnupg/openpgp-revocs.d/C4A3CFC51B1318FFD4D2C291D81710193A5FC56A.rev'
public and secret key created and signed
```

(3) gpg --list-keys

```
/home/instructor/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2019-12-11 [SC] [expires: 2021-12-10]
C4A3CFC51B1318FFD4D2C291D81710193A5FC56A
uid           [ultimate] julie <julie@email.com>
sub   rsa3072 2019-12-11 [E] [expires: 2021-12-10]
```

Asymmetric Key Encryptions

Exporting and Importing Keys

(1) **gpg --armor --output julie.gpg --export julie@email.com**

gpg	The command to run GPG.
-- armor	Puts the key in an ASCII format.
-- output julie.gpg	Creates the public key in an accessible format. In this case, we named the key **julie.gpg.**
-- export julie@email.com	References which key to use from the key ring. It is referenced by the email.

Asymmetric Key Encryptions

Viewing the key

(1) cat julie.gpg

-----BEGIN PGP PUBLIC KEY BLOCK-----

oCN2AghQUDgu5yBVAmPAx7hatvcMBR1X6NqJN4wStLB21OvHdgT2VbiHUtwkGvbJ
Hsui9eTR7bBY1YgP8PcGFjeMZ5+C7E94uYeksbwMzFWGE79M3kqEi1tgkDZTN/T8
8O31qQUgDCCbUnuvpW5pYJ2BconeNBHAZNKSKg+9U3DfCazRpky89be6W7WtjDGs
iFo5PEjBTvCJJXHvDgn2W7I7U0MWO220gyCT/Ja/eKad5GKTeMjOC4ERTwvha0ON

-----END PGP PUBLIC KEY BLOCK-----

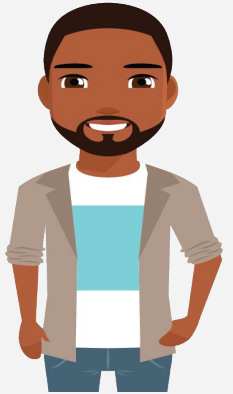
(2) gpg --list-keys

/home/instructor/.gnupg/pubring.kbx
---
pub rsa3072 2019-12-11 [SC] [expires: 2021-12-10]
C4A3CFC51B1318FFD4D2C291D81710193A5FC56A
uid [ultimate] julie <julie@email.com>
sub rsa3072 2019-12-11 [E] [expires: 2021-12-10]

(3) Julie would put this public key on her website, or she could directly share it with Tim.

Asymmetric Key Encryptions

Viewing the key



(1) gpg --import julie.gpg

-----BEGIN PGP PUBLIC KEY BLOCK-----

oCN2AghQUDgu5yBVAmPAx7hatvcMBR1X6NqJN4wStLB21OvHdgT2VbiHUtwkGvbJ
Hsui9eTR7bBY1YgP8PcGFjeMZ5+C7E94uYeksbwMzFWGE79M3kqEi1tgkDZTN/T8
8O31qQUgDCCbUnuvpW5pYJ2BconeNBHAZNKSKg+9U3DfCazRpky89be6W7WtjDGs
iFo5PEjBTvCJJXHvDgn2W7I7U0MWO220gyCT/Ja/eKad5GKTeMjOC4ERTwvha0ON

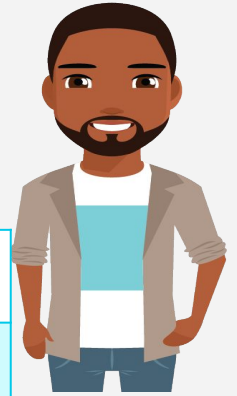
-----END PGP PUBLIC KEY BLOCK-----

(2) gpg --list-keys

pub rsa3072 2019-12-03 [SC] [expires: 2021-12-02]
39B2BD6C93E1E63E8C004183FE91AF7A7B4EC267
uid [ultimate] Julie <julie@email.com>
sub rsa3072 2019-12-03 [E] [expires: 2021-12-02]

Asymmetric Key Encryptions

Encryption



(1) **echo "Hi Julie, my bank account number is 2783492" > Tims_plainmessage.txt**

(2) **gpg --armor --output Tims_encryptedmessage.txt --encrypt --recipient julie@email.com Tims_plainmessage.txt**

gpg	The command to run GPG.
-- armor	Puts the key in an ASCII format.
-- output Tims_encryptedmessage.txt	Command for the output file, which creates the name of the encrypted file.
-- encrypt	Tells GPG to encrypt.
-- recipient julie@email.com	Tells GPG which public key to use, based on the email address of the key.
Tims_plainmessage.txt	Specifies for GPG which plaintext file to encrypt.

(3) **Run the above command - It creates a ciphertext called Tims_encryptedmessage.txt**

Asymmetric Key Encryptions

Decryption

Tim to send his encrypted message over to Julie, so she can decrypt it with her private key.

Julie receives Tim's encrypted message, she saves it in a directory where she created the key pair and then run decryption commands against the file.

(1) gpg --output Tims_decrypted_message --decrypt Tims_encryptedmessage.txt

gpg	The command to run GPG.
-- output Tims_decrypted_.txt	creates an output file, which is the decrypted message.
-- decrypt Tims_encryptedmessage.txt	indicating to decrypt and what file to decrypt.

(2) cat Tims_decrypted_message

Instructor Demonstration

GPG

Activity: GPG

In this activity, you will use asymmetric encryption with GPG to encrypt and decrypt text files.

Suggested Time:

20 Minutes

Time's Up! Let's Review.

Questions?

Countdown timer

15:00

(with alarm)

Hashing and Data Integrity

So far, we've used cryptography to protect privacy and confidentiality.

It can also be applied to protect the integrity of data.

Hashing is a cryptographic method used to verify the integrity of data.

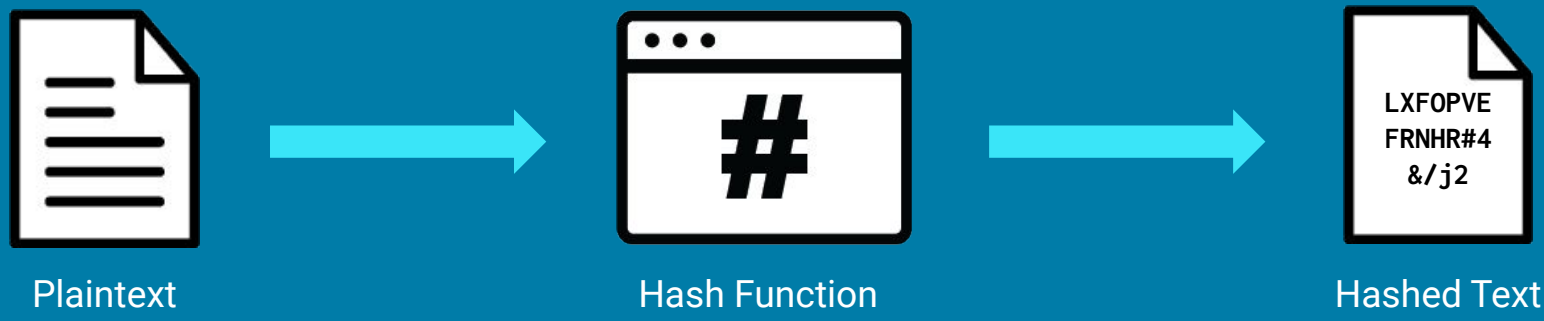
Hashing

Similar to encryption, **hashing** uses algorithms to input data and generate a unique output. While this seems similar to encryption, we'll notice that there are several significant differences.

Difference One: Keys vs. No Keys

Encryption takes plaintext and converts it to ciphertext with a key and an algorithm.

Hashing takes plaintext and converts it to a **message digest** with an algorithm, and no key.



Difference One: Keys vs. No Keys

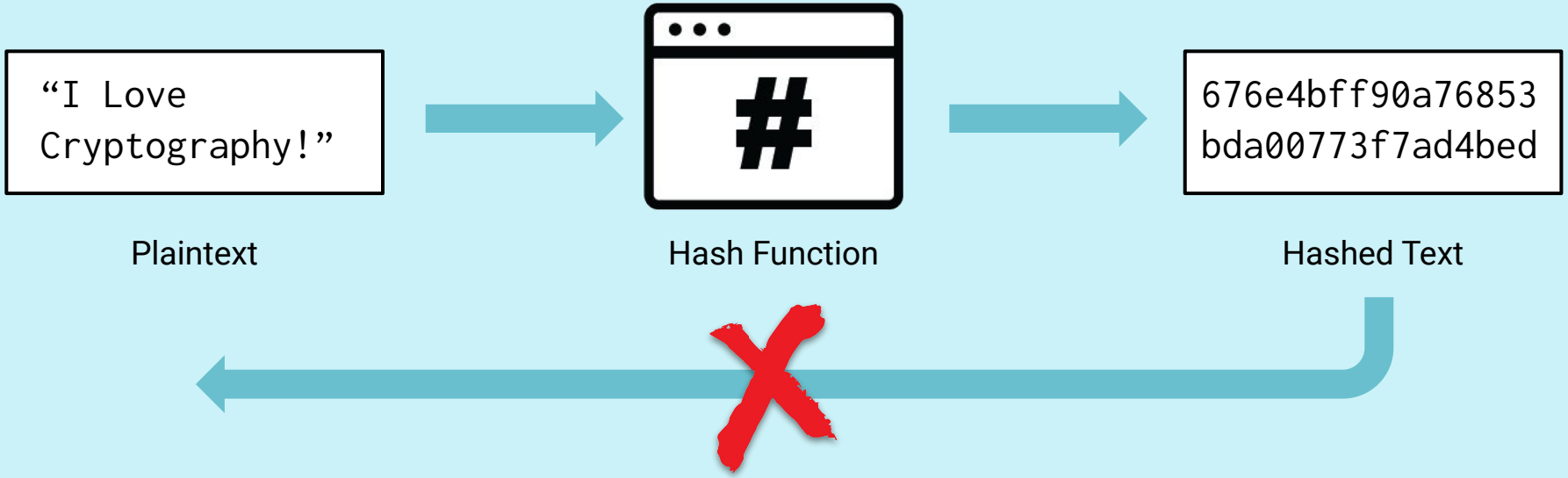
Message digests are also known as fingerprints, hashes, and checksums.

They are unique identifiers of the plaintext that is outputted from the algorithm. For example:

```
676e4bff90a76853bda00773f7ad4bed
```

Difference Two: Reversible vs. Irreversible

Encryption converts ciphertext and plaintext back and forth using encryption and decryption.



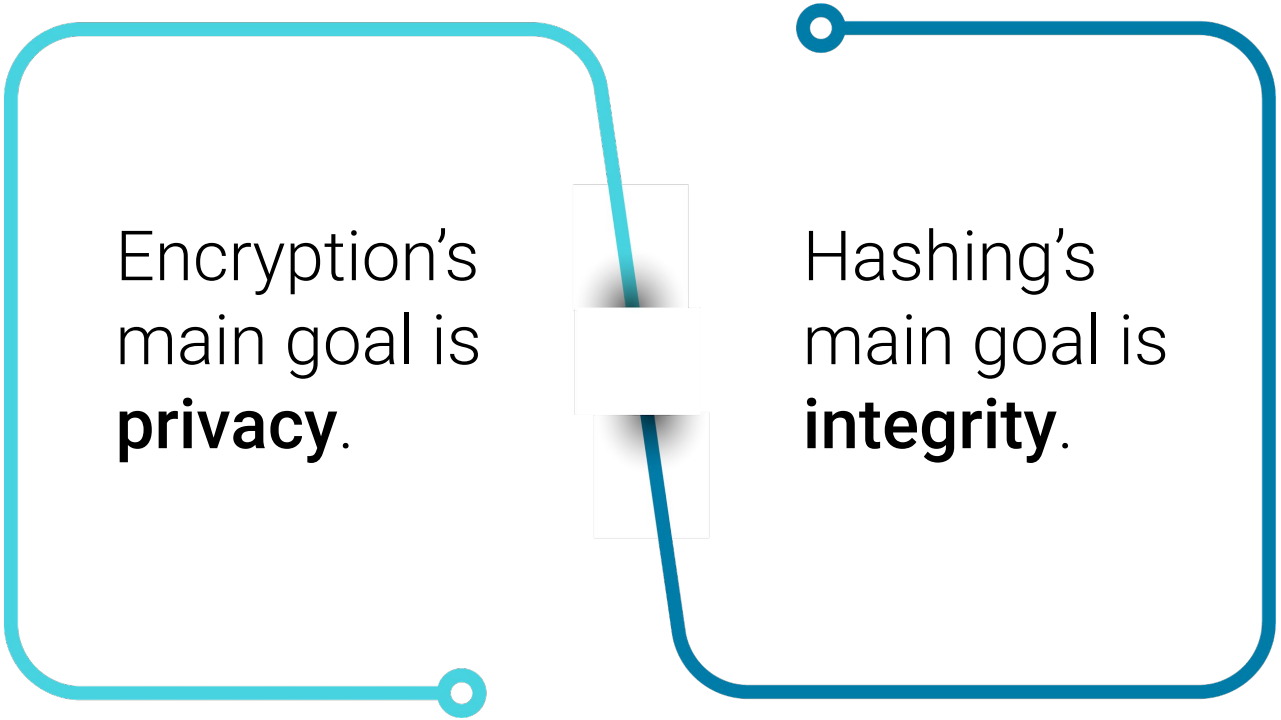
Hashing is a **one-way function**, meaning it cannot be converted back to plaintext.

Difference Three: Output Lengths

Hashing algorithms output fixed lengths. Regardless of the input length, the output length is always the same.

`I Love Cryptography!`	#	`676e4bff90a76853bda00773f7ad4bed`
`Hi`	#	`c1a5298f939e87e8f962a5edfc206918`
We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness...	#	`124650d689d47f777f715a25260a29c2` This is the hash of the entire Declaration of Independence!

Difference Four: Goals



Difference Four: Goals

If a small change is made to the input of a hashing algorithm, the message digest is completely different.

```
I Love Cryptography! = 676e4bff90a76853bda00773f7ad4bed
```

```
I Love Cryptography!! = 4e6fc433ff57a6c4a854cbbeff65f61a
```

```
I Love Cryptography? = d7a5035b338b66bf7f50dc85dbf1a279
```

```
I Luv Cryptography! = ad7e7b47b58e28a2894f5fcf9dc41691
```

Hashing Algorithms

Hashing has several algorithms:

SHA	Stands for Secure Hashing Algorithms, and includes its successors, **SHA1** and **SHA2**.
SHA2	Has variations with different security strengths: **SHA-256** and **SHA-512**.
MD	Stands for Message Digest and has several variations: **MD2**, **MD4**, and **MD5**.
LM and NTLM	Are hashes used by Windows.

Instructor Demonstration

Creating Hashes on the Command Line

Creating Hashes

Hashing with md5sum

1. **echo "This is my first hashing activity" > secretmessage.txt**
2. **md5sum secretmessage.txt > hashes.txt**

md5sum	The terminal command to run the MD5 algorithm.
secretmessage.txt	The file to be hashed..
> hashes.txt	The output file where the message digest is placed. (optional)

cat hashes.txt

Creating Hashes

Checking integrity with md5sum

1. **echo "This is my second hashing activity" > secretmessage.txt (overwrite the first file)**
2. **md5sum -c hashes.txt > md5check.txt**
3. **md5sum secretmessage.txt > hashes.txt**

md5sum	The terminal command to run the MD5 algorithm.
-c	The option to have ***md5sum*** check the hashes.
hashes.txt	The file the check is being run against.
> md5check.txt	The output file where the results of the check are placed.

md5sum: WARNING: 1 computed checksum did NOT match
cat md5check.txt
secretmessage.txt: FAILED

Activity: Generating Hashes

In this activity, you will verify the integrity of files by generating hashes of current and backup files, and then comparing them.

Suggested Time:

17 Minutes

Time's Up! Let's Review.

Questions?

Digital Signatures

Cryptography can also be used to validate authenticity using digital signatures.

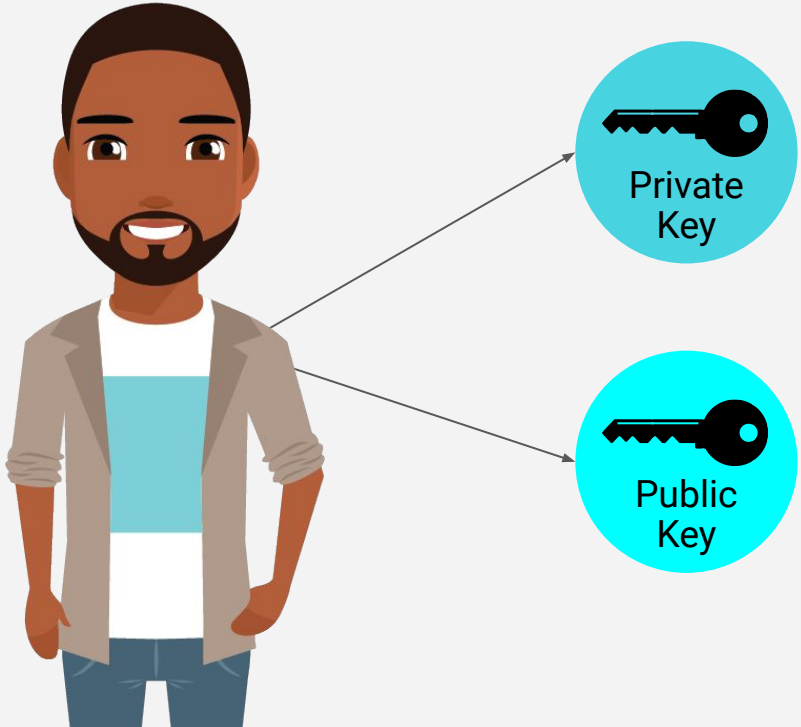
Digital Signatures

A digital signature is a mathematical scheme used to verify the authenticity of digital data.

- In the United States and several other countries, digital signatures are considered legally binding.
- Similar to asymmetric encryption, digital signatures also use public key cryptography.
- However, digital signatures use public and private keys in reverse.

Digital Signatures

Scenario: Tim wants to send Julie a message that says “Transfer $500 to the account I provided to you,” and then digitally sign the message.



STEP 1

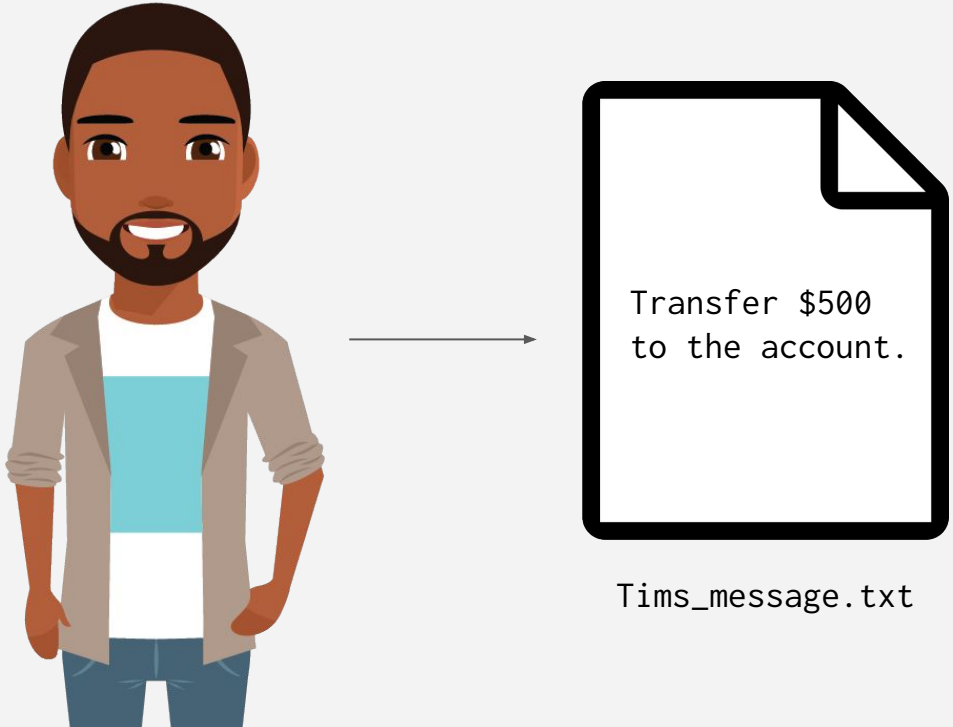
Key Creation

Tim will have a public and a private key.

The public key is truly public, and accessible on his website.

Digital Signatures

Scenario: Tim wants to send Julie a message that says “Transfer $500 to the account I provided to you.” and then digitally sign the message.



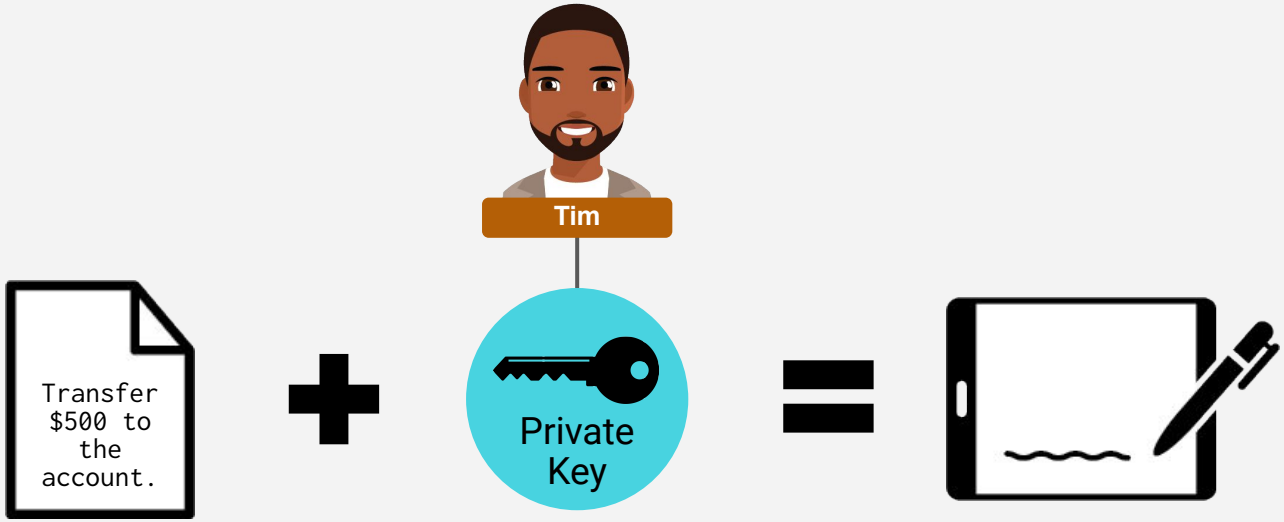
STEP 2

Creating the Message

Tim places his message inside a file.

Digital Signatures

Scenario: Tim wants to send Julie a message that says “Transfer $500 to the account I provided to you,” and then digitally sign the message.



`Tims_message.txt`

Digital Signature

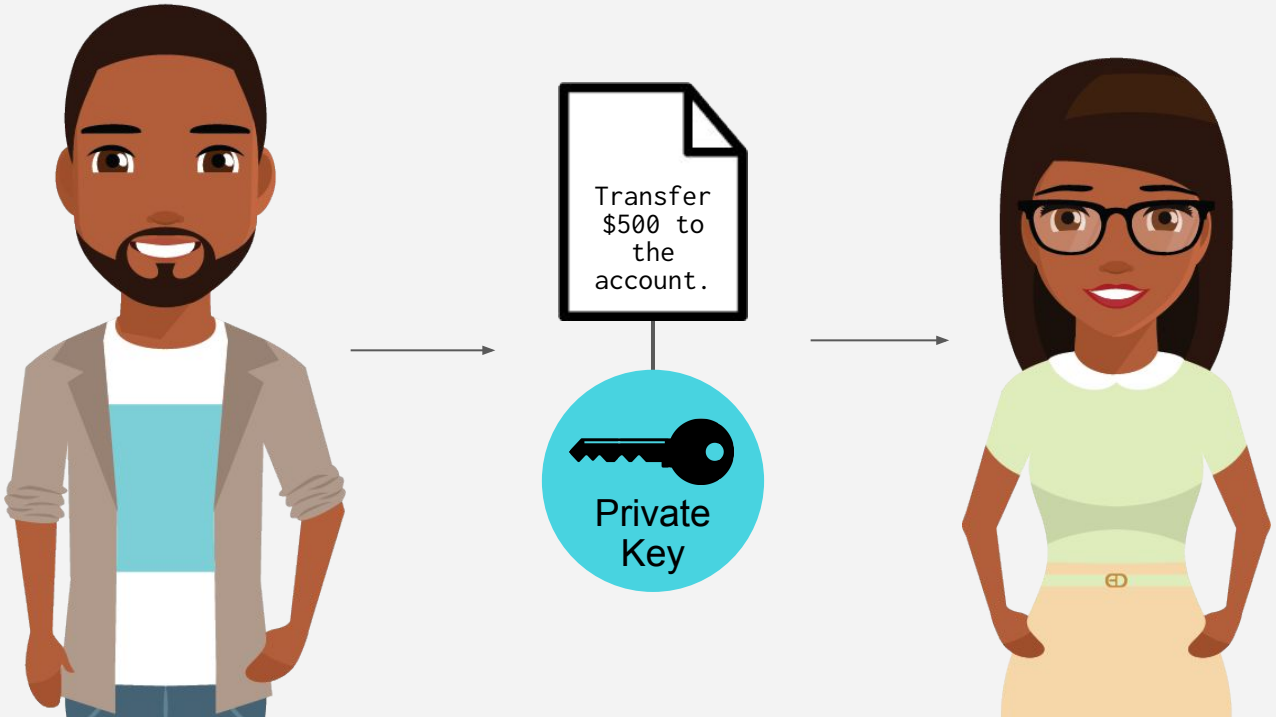
STEP 3

Signing the Message

Tim signs the message with his private key in order to create a digital signature.

Digital Signatures

Scenario: Tim wants to send Julie a message that says “Transfer $500 to the account I provided to you,” and then digitally sign the message.



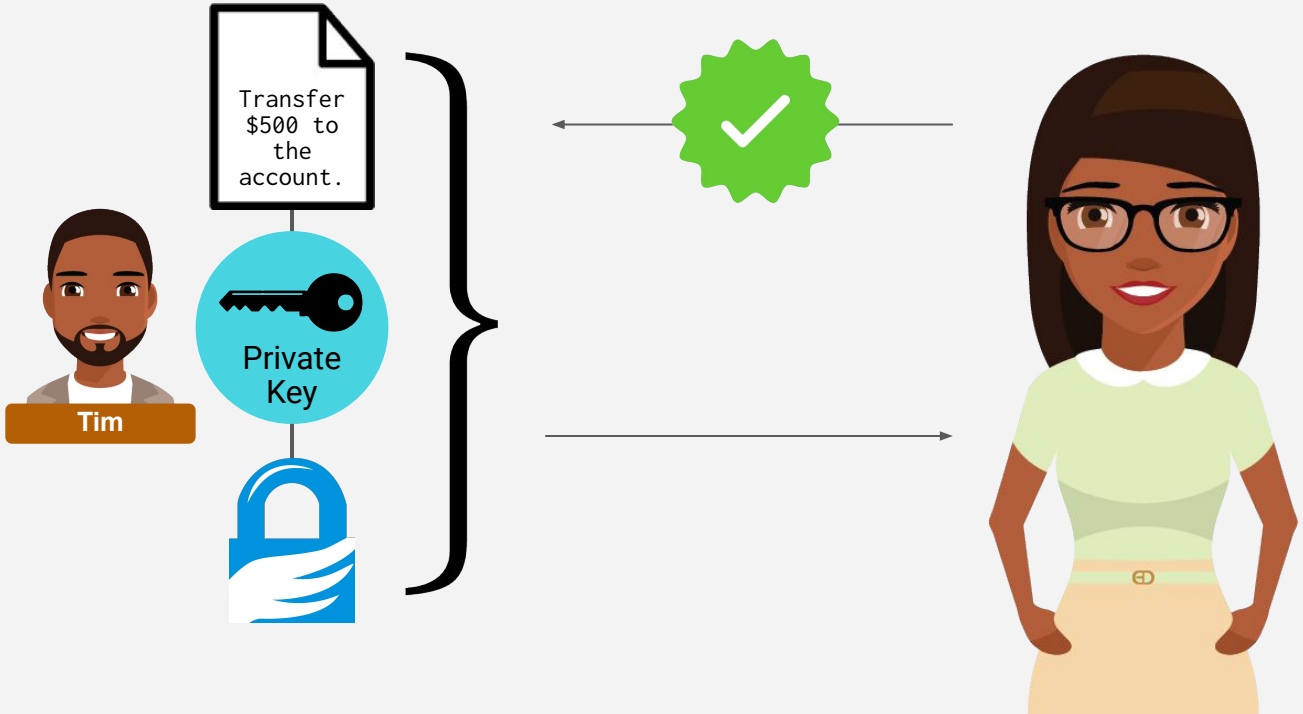
STEP 4

Sending the Message

Tim sends the digital signature to Julie along with his plaintext message.

Digital Signatures

Scenario: Tim wants to send Julie a message that says “Transfer $500 to the account I provided to you,” and then digitally sign the message.



STEP 5

Validating the Signature

Julie uses Tim's public key to validate the signature, using a validation tool such as GPG.

That scenario depicted a **detached signature**, in which the message and the signature are sent separately.

Digital Signatures

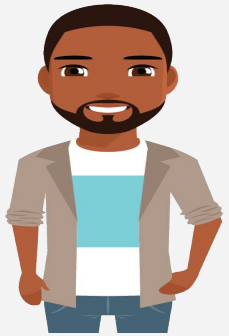
Other digital signatures include:

All at Once	Clearsigned	Signed Hash
A signature appended to an encrypted message.	A signature appended to an unencrypted message.	Instead of signing a message, a hash is created first and the hash is signed for verification.

In the next demo, we will apply a detached digital signature with GPG.

Digital Signatures

Creating Key



(1) gpg --gen-key

(2) Change (N)ame, (E)mail, or (O)kay/(Q)uit? Press O to confirm your information

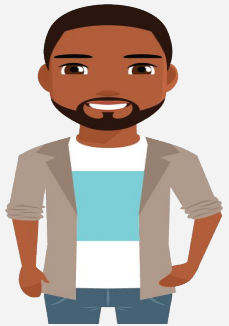
```
gpg: key D81710193A5FC56A marked as ultimately trusted
gpg: directory '/home/instructor/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/instructor/.gnupg/openpgp-revocs.d/C4A3CFC51B1318FFD4D2C291D81710193A5FC56A.rev'
public and secret key created and signed
```

(3) gpg --list-keys

```
/home/instructor/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2019-12-11 [SC] [expires: 2021-12-10]
C4A3CFC51B1318FFD4D2C291D81710193A5FC56A
uid           [ultimate] julie <julie@email.com>
sub   rsa3072 2019-12-11 [E] [expires: 2021-12-10]
```

Digital Signatures

Exporting and Importing Keys



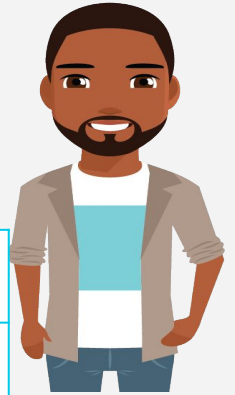
(1) **gpg --armor --output tim.gpg --export tim@email.com**

gpg	The command to run GPG.
-- armor	Puts the key in an ASCII format.
-- output tim.gpg	Creates the public key in an accessible format. In this case, we named the key **tim.gpg**.
-- export tim@email.com	References which key to use from the key ring. It is referenced by the email.

Digital Signatures

Creating the message

(1) echo "Transfer $500 to the account I provided to you" > Tims_message.txt
(2) gpg --output Tims_signature --armor --detach-sig Tims_message.txt



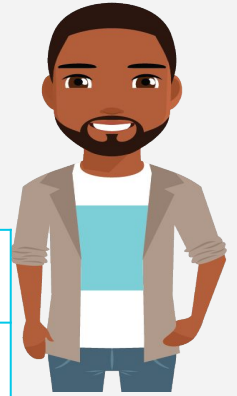
gpg	The command to run GPG.
-- output Tims_signature	specifies the output file that contains the digital signature.
-- armor	Puts the key in an ASCII format.
--detach-sig Tims_message.txt	specifies that a detached signature will be created against the file `Tims_message.txt`

(3) Run the above command - It creates a ciphertext called Tims_encryptedmessage.txt

Digital Signatures

Creating the message

(1) echo "Transfer $500 to the account I provided to you" > Tims_message.txt
(2) gpg --output Tims_signature --armor --detach-sig Tims_message.txt

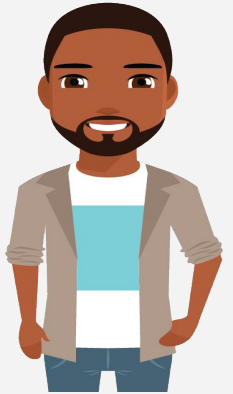


gpg	The command to run GPG.
-- output Tims_signature	specifies the output file that contains the digital signature.
-- armor	Puts the key in an ASCII format.
--detach-sig Tims_message.txt	specifies that a detached signature will be created against the file `Tims_message.txt`

(3) Run the above command - digital signatures automatically use your private key, GPG will prompt you to put in the password used to create your key pair.
(4) Enter the password and explain that a separate digital signature called *Tims_signature* has just been created.

Digital Signatures

Preview the Signature



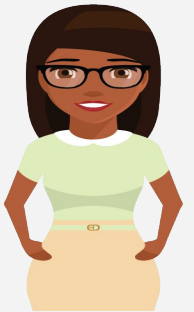
(1) cat Tims_signature
(2) gpg --output Tims_signature --armor --detach-sig Tims_message.txt

```
-----BEGIN PGP SIGNATURE-----

iQGzBAABCgAdFiEEObK9bJPh5j6MAEGD/pGventOwmcFAl3pGKoACgkQ/pGventO
wmdPfQv8CigGztcvrdbZrJVr91mPiLL5cry7nKYDAsRqkyIDltiJMxtggVbCtSPm
YLfqZATWYofBWdE4wkpmeYE96gXTeJP4VVNUpwnshg1A1q0att10S+rlv6N73g4V

-----END PGP SIGNATURE-----
```

(3) Tim will send the digital signature file to Julie along with his plaintext message.



For Julie to verify Tim's message, Julie will need to import Tim's public key

(1) gpg --import tim.gpg
(2) gpg --verify Tims_signature Tims_message.txt

```
gpg: Signature made Thu 05 Dec 2019 09:48:10 AM EST
gpg:                using RSA key 39B2BD6C93E1E63E8C004183FE91AF7A7B4EC267
gpg: Good signature from "Tim <tim@email.com>" [ultimate]
```

Digital Signatures

Integrity Validation

echo "Transfer $34,547 to the account I provided to you" > Tims_message.txt

gpg --verify Tims_signature Tims_message.txt

gpg: Signature made Thu 05 Dec 2019 09:48:10 AM EST
gpg: using RSA key 39B2BD6C93E1E63E8C004183FE91AF7A7B4EC267
gpg: BAD signature from "Tim <tim@email.com>" [ultimate]

Instructor Demonstration

Signing with GPG

Activity: Digital Signatures

In this activity, you will use GPG to determine the authenticity of messages.

Suggested Time:

20 Minutes

Time's Up! Let's Review.

Questions?


The End