Mastering Asymmetric Cryptography

Text

A Practical Guide to OpenSSL in the Command Line

Vaithee

The Two Pillars of Digital Trust

How do we keep information **private?**

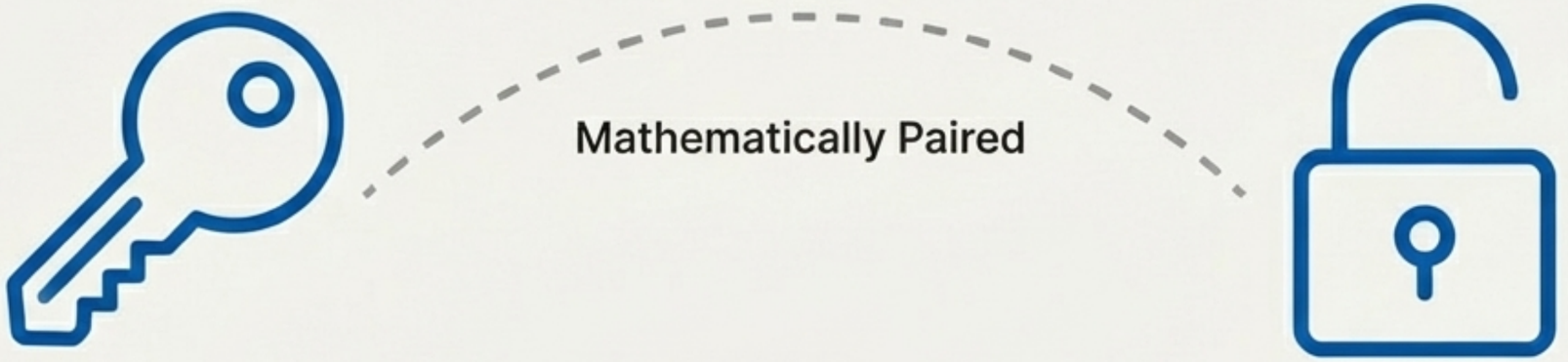
Ensuring that only the intended recipient can read a message, even if it is intercepted by others.

How do we **prove** information is **genuine?**

Verifying that a message truly comes from the claimed sender and has not been altered in transit.

Asymmetric cryptography provides an elegant solution to both.

The Solution: A Mathematically Linked Pair of Keys



Private Key

SECRET. Known only to you.
Used for decrypting and signing.

Public Key

SHARABLE. Distributed to the world.
Used for encrypting and verifying.

Your First Step: Generating an RSA Key Pair

Prerequisites

- A Linux environment with OpenSSL installed. (Check with `openssl version`).
- A dedicated directory for this exercise.

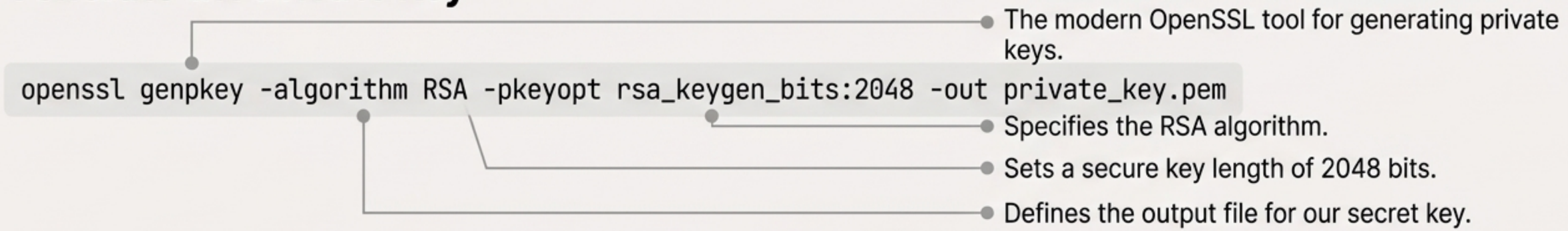
Setup Commands

```
mkdir asym-lab
cd asym-lab
```

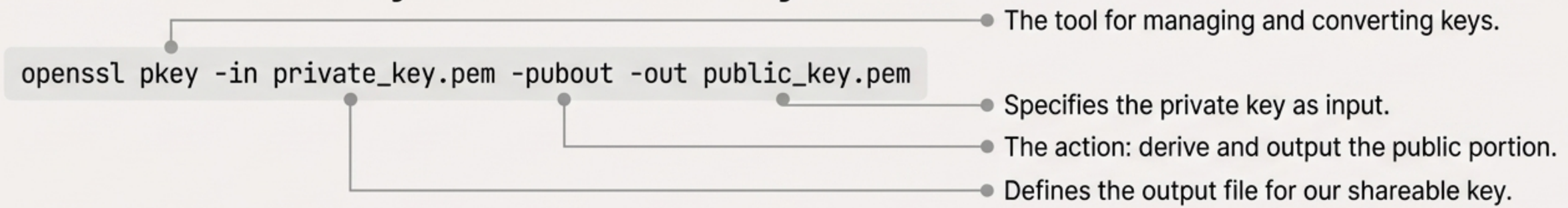
We will use OpenSSL to generate a 2048-bit RSA key pair. This is the foundation for all subsequent operations.

Anatomy of the Key Generation Commands

Generate the Private Key



Extract the Public Key from the Private Key



Pro Tip

To protect your private key with a passphrase, add `-aes-256-cbc -passout pass:yourpassphrase` to the `genpkey` command.

Mission 1: Achieving Confidentiality with Encryption

Anyone can lock a message using your public key, but only you can unlock it with your private key.

The public key encrypts. The private key decrypts.

Encryption in Practice: Locking the Message

Step 1: Create a Sample Message

```
echo "This is a secret message for the lab." > message.txt
```

Step 2: Encrypt the Message with the Public Key

```
openssl pkeyutl -encrypt -pubin -inkey public_key.pem -in message.txt -out encrypted.bin
```

- `pkeyutl` — The utility for public key cryptographic operations.
- `-encrypt` — Specifies the action is to encrypt.
- `-pubin` — Indicates that the input key is a public key.
- `-inkey public_key.pem` — The public key to use for encryption.

> Direct RSA encryption is only suitable for small data blocks. In real-world applications like TLS, it is used to securely exchange a key for faster, symmetric encryption (this is called 'hybrid encryption').

Decryption in Practice: Unlocking the Secret

Step 1: Decrypt the Message with the Private Key

```
openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted.bin -out decrypted.txt
```

Note that if the key were passphrase-protected, you would need to add `-passin pass:yourpassphrase`.

Step 2: Verify the Result

```
cat decrypted.txt
```

```
This is a secret message for the lab.
```

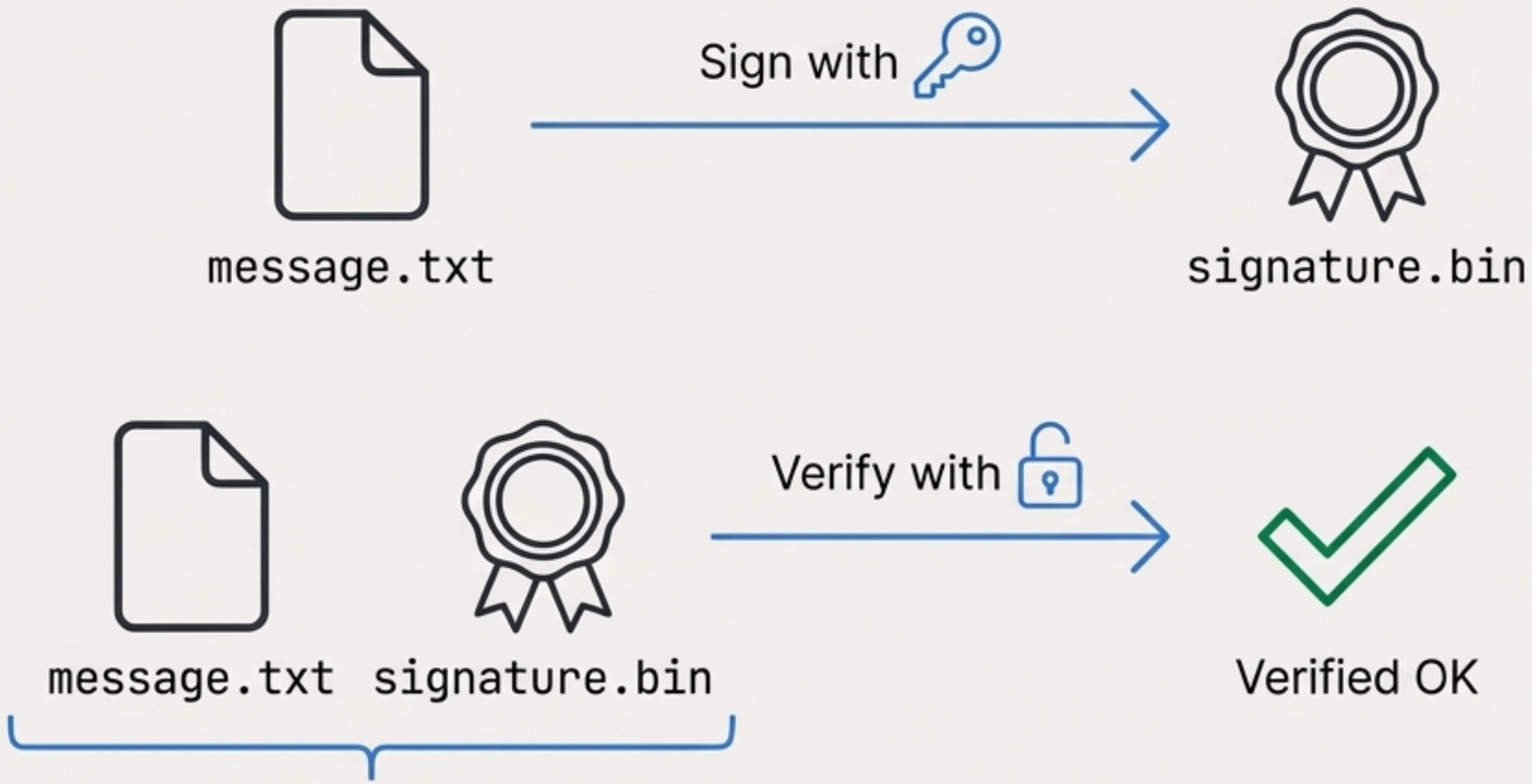
```
diff message.txt decrypted.txt
```

No output

Success. The decrypted file is identical to the original.

Mission 2: Ensuring Authenticity with Digital Signatures

You sign a message with your private key to prove it's from you. Anyone can use your public key to check the signature.



The private key signs. The public key verifies.

Signing in Practice: Creating the Digital Seal

Sign the Message Digest with the Private Key

```
openssl dgst -sha256 -sign private_key.pem -out signature.bin message.txt
```

The OpenSSL message digest (hashing) utility.

Specifies creating a SHA256 hash of the message before signing. This ensures efficiency and security.

Signs the hash using your private key.

The file containing the resulting binary signature.

The original message to be signed.

Verification in Practice: Checking the Seal

Verify the Signature against the Original Message

```
openssl dgst -sha256 -verify public_key.pem -signature signature.bin message.txt
```

#0057B8

Specifies the action is to verify, providing the public key to check against.

#0057B8

Points to the signature file file we created.

#0057B8

The original message, which OpenSSL will re-hash and compare.

The Integrity Test: What Happens When Data is Tampered With?

A digital signature fails if even a single bit of the original message is changed. Let's demonstrate.

Step 1: Tamper with the Message File

```
echo "Tampered!" >> message.txt
```

Step 2: Attempt Verification Again with the Original Signature

```
openssl dgst -sha256 -verify public_key.pem -signature signature.bin message.txt
```

Verification Failure

The Duality of a Key Pair: A Summary

For Confidentiality

Goal:
Keep data secret.

Analogy:
A public letterbox that only you have the key to open.

Process:

Encrypt with: **Public Key** 

Decrypt with: **Private Key** 

For Authenticity

Goal:
Prove origin and integrity.

Analogy:
A unique signature that anyone can check against a public record.

Process:

Sign with: **Private Key** 

Verify with: **Public Key** 

Beyond the Lab: Exploration and Real-World Context

These OpenSSL commands demonstrate the core primitives of asymmetric cryptography. In production systems like **SSH** and **TLS**, these concepts are foundational to building secure communication channels.

Challenge Yourself

1. What happens if you try to decrypt with the wrong private key, or verify with the wrong public key?
2. Generate a 4096-bit key. Compare the generation time.
3. Try signing a larger file. Does the signature size change? Why or why not?

```
To learn more: man openssl
To clean up your directory: rm *.pem *.txt *.bin
```