Computer security week 5

Khaula Molapo

1.

Imagine you are downloading a critical software update from a company's website.
To ensure the software has not been tampered with, the company provides both the executable file and a digital signature file.
When you download the software, your computer uses the company's public key to verify the signature against the provided file.
The digital signature itself was created using the company's private key, applied to the hash of the software.
If the hash values match during verification, you can be confident that the software you installed is authentic and untampered.
This process prevents attackers from replacing the software with a malicious version without detection.
For example, if an attacker intercepted your download and inserted malware, the hash of the altered file would no longer match the signed hash.
In this way, hashing and signatures work together to protect users, ensuring integrity, authenticity, and trust in software distribution.

2.

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function developed by the NSA and standardized by NIST.

It produces a fixed 256-bit (32-byte) output, regardless of input size. Its role is to ensure data integrity by creating a unique digital fingerprint of information.

Even a tiny change in input drastically changes the output, making it resistant to collisions and preimage attacks.

SHA-256 is widely used in digital signatures, SSL certificates, password storage, and blockchain systems like Bitcoin.

Its reliability and strength make it one of the most trusted algorithms for securing sensitive data across modern applications.

3.

Practicing message signing in Python made the concept of digital signatures much clearer.

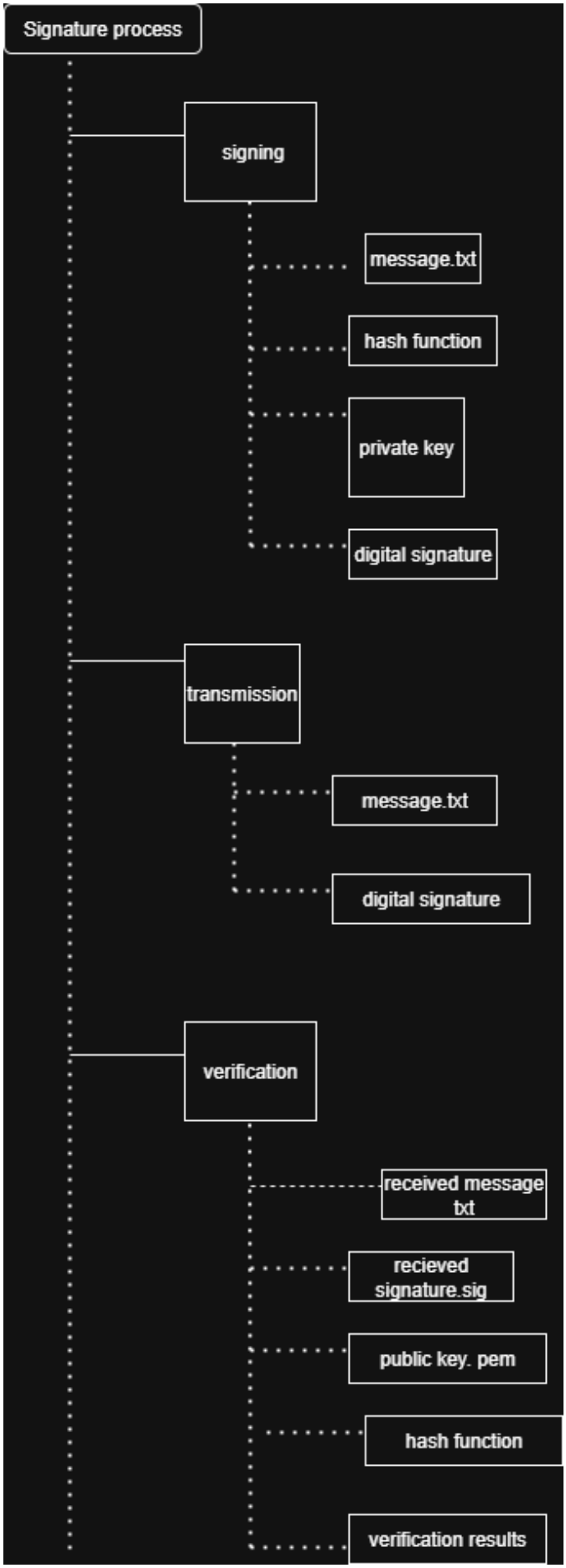
At first, the code looks complicated, but once broken down, it's straightforward: generate keys, sign a message, and verify it.

What impressed me most was how easy it is for the public key to catch even the smallest

tampering attempts.
If the message changes even slightly, verification fails immediately. This hands-on test demonstrated the real-world usefulness of hashing and signing.
It shows how technology ensures both integrity and authenticity, especially in sensitive areas like secure communications, banking, and software updates.

4.

In Draw.io, I created a simple diagram of the digital signature process.
It starts with a *Message*, which is first sent through a *Hash Function* (like SHA-256) to generate a digest.
This digest is then encrypted using the sender's *Private Key*, producing the *Digital Signature*.
The message and signature are sent together. On the receiver's end, the message is again hashed to create a digest.
Simultaneously, the signature is decrypted with the sender's *Public Key*. If both digests match, the signature is valid,
ensuring the message is authentic and unaltered. This flow ensures security and trust.


Signature process
signing
message.txt
hash function
private key
digital signature
transmission
message.txt
digital signature
verification
received message txt
recieved signature.sig
public key. pem
hash function
verification results