FH JOANNEUM
Elektronik und
Computer Engineering

Security and Testing

Cryptography / Encryption of Data at Rest

FH JOANNEUM
Institute of Electronic Engineering
System Test Engineering
Egon Teiniker

Version: 1.3.0

CC BY SA

Introduction

Outline

- Introduction
  - Context of Cryptography
  - Recommendation for Using Cryptography

Introduction

Context of Cryptography

- **Mathematical Problems:** Factoring, finite sets, …
- **Cryptographic Primitives:** RSA, Diffie-Hellman, AES, SHA-256, …
- **Protocols:** TLS, SSH, PGP, …
- **Library Implementations:** OpenSSL, BSAFE, NaCl, …
- **Software Applications:** Apache, Firefox, Chrome, …

Introduction

Recommendations for Using Crypto

- Don’t implement your own crypto library
  - “Nobody ever got fired for using AES.”
- Avoid obsolete crypto
  - DES: Broken – don’t use
  - 3DES: Still used for legacy apps, don’t use for new apps
  - MD5: Don’t use for signatures
  - SHA-1: Will soon be questionable for signatures
- Use reasonable key lengths
  - AES-256
  - SHA-256
  - DSA/RSA-4096

Introduction

Recommendations for Using Crypto

- Use a good random source
  - /dev/random
  - /dev/urandom
  - Don’t use: Time of day, random()
- Use the right cipher mode
  - Don’t use ECB, use CBC, CTR
  - Don’t reuse IVs, use a fresh random IV
- Listen to cryptographers

Encryption of Data at Rest

Outline

- Pseudo-Random Number Generator
- Message Digest
  - Secure Password Storage
- Key Derivation Functions
- Message Authentication Code
  - JSON Web Tokens
- Symmetric Encryption
- Asymmetric Encryption
- Digital Signatures
- Digital Certificates

Encryption of Data at Rest

Pseudo-Random Number Generator

A Pseudo-Random Number Generator (PRNG) is an algorithm used to generate numbers that **approximate true randomness**. PRGN approximate randomness by using a seed from which subsequent values are generated.

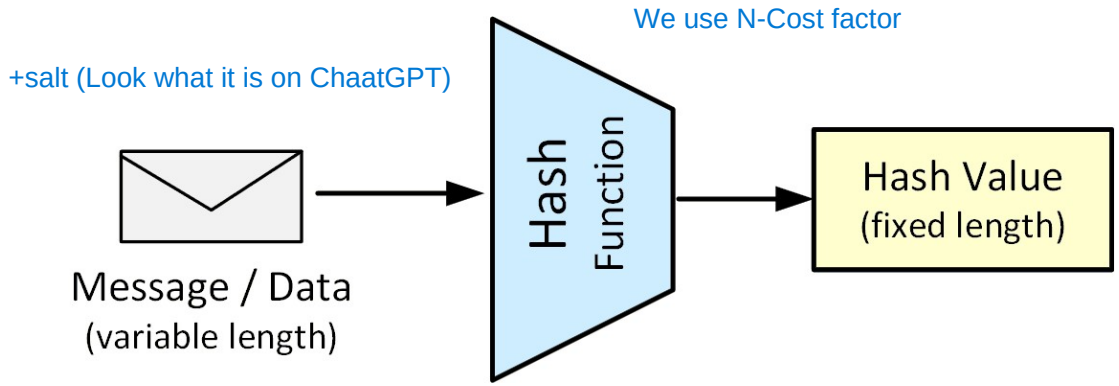
- In Python, the pseudo-random generators of the **random module should not be used for security purposes**.
  For security or cryptographic uses, see the **secrets module**.

- We can also use the output of the **/dev/urandom** PRNG that is provided by the OS.

Encryption of Data at Rest

Message Digest

Message digests compute a **cryptographic hash**, or secure checksum, for a particular message.



Different **algorithms** can be used: ~~MD5~~, ~~SHA-1~~, **SHA-256, SHA-512**

Encryption of Data at Rest

Message Digest

Hash values can be used for:

- **Verify data integrity:** When we download a binary file, the website hosting the download often publishes a checksum on the site as well. We can generate a hash value of our downloaded file and ensure that it matches the values published on the download site. If the values match, we have verified that the downloaded file contains valid data.
- **Store user passwords:** Security best practice state that user passwords should only be known by the user himself. This can be accomplished by storing the hash value of the user's password instead of the password itself.

Encryption of Data at Rest

Secure Password Storage

A **salt** consists of a number of bits of random data this is concatenated to the input of a cryptographic function.

The salt **should be unique for each input value** so that a unique hash value will be generated even if the input data is identical.

It is beneficial to add a salt to a message digest to make the hash values unique and **to mitigate brute force attacks**.

An attacker who has access to a non-salted hash value can use **rainbow tables** to find the string that created the hash.

Encryption of Data at Rest

Secure Password Storage

Hashing a Password:

1. Generate a random salt
2. Concatenate salt + password
3. Hash (salt + password)
4. Concatenate (salt + hash(salt + password)) = hash value

In step 3, we can run the hash algorithm 100.000 times to **increase the calculation time**.

Encryption of Data at Rest

Secure Password Storage

Checking a Password:

1. Extract salt from hash
2. Concatenate salt + password
3. Hash (salt + password)
4. Compare the both hash values

We have to calculate the hash value (using the same salt) for the entered password before we can compare the values.

Encryption of Data at Rest

Key Derivation Functions

Key derivation functions (KDF) derive bytes suitable for cryptographic operations from passwords or other data sources using a pseudo-random function.

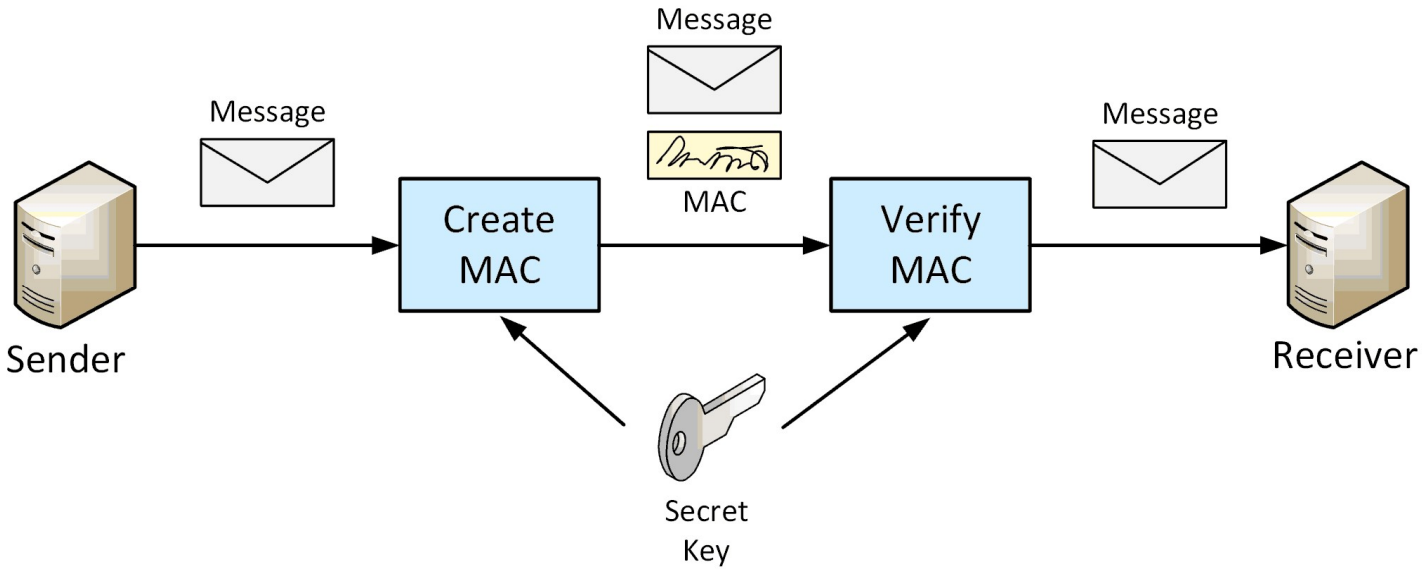
Different KDFs are suitable for different tasks:

- **Cryptographic key derivation**: Deriving a key suitable for use as input to an encryption algorithm. This process is typically known as key stretching. *Examples:* PBKDF2, HKDF.
- **Password storage**: When storing passwords we want to use an algorithm that is computationally intensive. Ideal password storage KDFs will be demanding on both computational and memory resources. *Example:* Scrypt

Encryption of Data at Rest

Message Authentication Code

A Hash Message Authentication Code (HMAC) is different from a regular message digest calculation because it incorporates **a secret key** as well.



Encryption of Data at Rest

JSON Web Tokens

JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.

Structure of a JWT:

```
<base64-encoded header>.<base64-encoded payload>
.<base64-encoded signature>
```

- The first part is the **JSON Object Signing and Encryption Header (JOSE)**
- The second part is the **claims set** (or **payload**)
- The third part is the **signature**

Encryption of Data at Rest

JSON Web Tokens

Encryption of Data at Rest

JSON Web Tokens

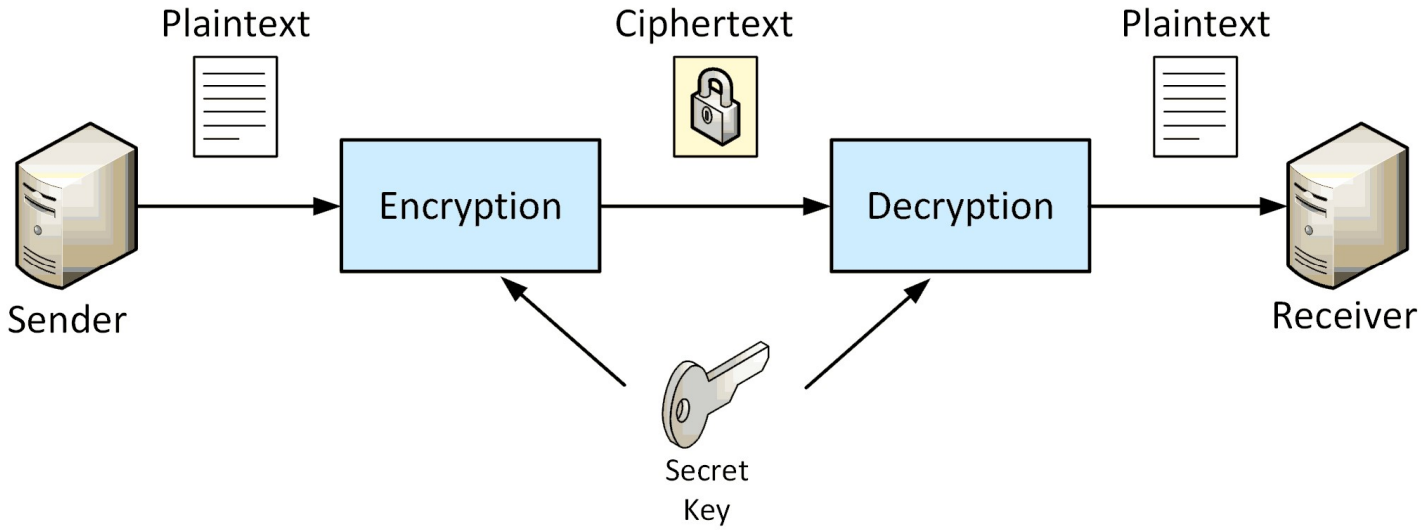
The claim attributes are:

- **iss**: The issuer of the token
- **sub**: The subject of the token
- **aud**: The audience of the token
- **qsh**: query string hash
- **exp**: Token expiration time defined in Unix time
- **nbf**: "Not before" time that identifies the time before which the JWT must not be accepted for processing
- **iat**: "Issued at" time, in Unix time, at which the token was issued
- **jti**: JWT ID claim provides a unique identifier for the JWT

Encryption of Data at Rest

Symmetric Encryption

Secret key cryptography is when a plaintext message is **encrypted with a private, or symmetric, key** to create ciphertext. The message can be then **decrypted using the same secret key** that was used for encryption.



Encryption of Data at Rest

Symmetric Encryption

A **cipher** is an algorithm for performing encryption and decryption.

Common algorithms are: **AES**, ~~DES~~, ~~TripleDES~~, ~~Blowfish~~, among others.

There are two primary **types of ciphers**:

- **Block ciphers** process blocks of data of fixed size. If there isn't enough data to make a complete block, the data must be padded before it is encrypted. When it is decrypted, this padding is removed.
- **Stream ciphers** do not require padding because they encrypt continuous streams of data, typically a byte or even a bit at a time.

Encryption of Data at Rest

Symmetric Block Cipher Modes

Block ciphers can operate on different modes:

- ~~**Electronic Code Book (ECB)**~~

  ECB mode describes the use of a symmetric cipher in its rawest form. The problem with ECB mode is that if there are patterns in the data, there will be patterns in the encrypted data as well.

  Example:

  input : 000102030405060708090a0b0c0d0e0f0001020304050607

  cipher: e1b246e5a7c74cbc92c9db45300b932fe1b246e5a7c74cbc
  e481a8d39714d0de

Encryption of Data at Rest

Symmetric Block Cipher Modes

Block ciphers can operate on different modes: (continued)

- **Cipher Block Chaining (CBC)**

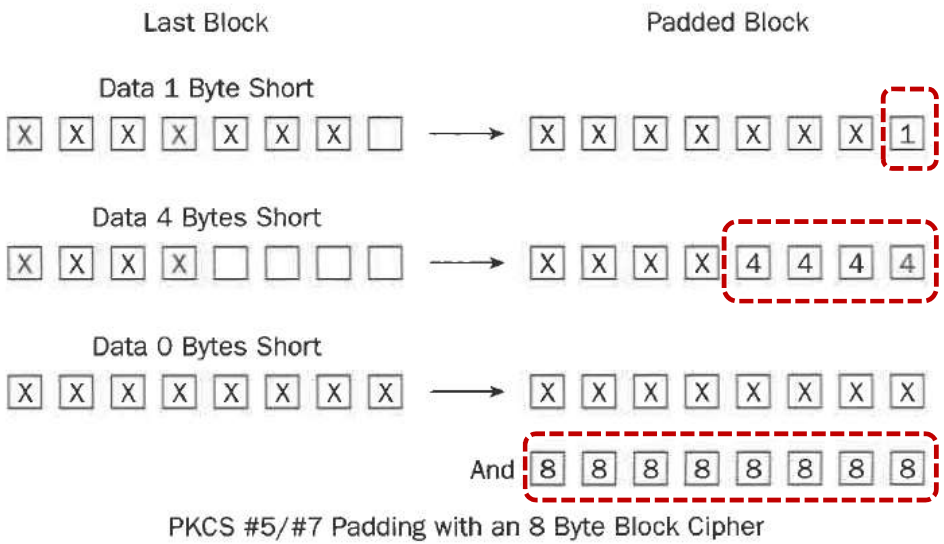
  CBC mode reduces the likelihood of patterns appearing in the cipher text by XOR-ing the block of data to be encrypted with the last block of cipher text produced and then applying the raw cipher to produce the next block of cipher text.

  It is the **Initialization Vector (IV)** that provides the initial block of cipher text that is XOR-ed with the first block of input.

Encryption of Data at Rest

Symmetric Block Cipher Padding

PKCS#5 / PKCS#7 Padding:



PKCS #5/#7 Padding with an 8 Byte Block Cipher

Encryption of Data at Rest

Symmetric Block Cipher Modes

Block ciphers can operate on different modes: (continued)

- **Segment Integer Counter (CTR)**

  CTR or Counter mode is defined in RFC3686. We don't have to specify any padding because the mode allows you to work with any length of data.

  Example: "**AES/CTR/NoPadding**"

  The advantages of the CTR mode are:

  - It is a **streaming mode**, so we don't have to worry about padding.
  - It allows for random access to the encrypted data.

Encryption of Data at Rest

Symmetric Stream Ciphers

Stream ciphers are basically just ciphers that, by design, **behave like block ciphers using the streaming modes**. The idea is for the cipher to create a stream of bits that are then XOR-ed with the plain text to produce the cipher text.

Common algorithms: ~~RC4~~, ChaCha20, AES in CTR mode

Stream ciphers **do not have modes or require padding.**

They will always produce **output of the same length as the input**.

Encryption of Data at Rest

Asymmetric Encryption

Asymmetric Encryption is a form of Encryption where keys come in pairs. These two keys are generated simultaneously in the same process. **Encryption uses the public key**, while **decrypting requires the private key**.

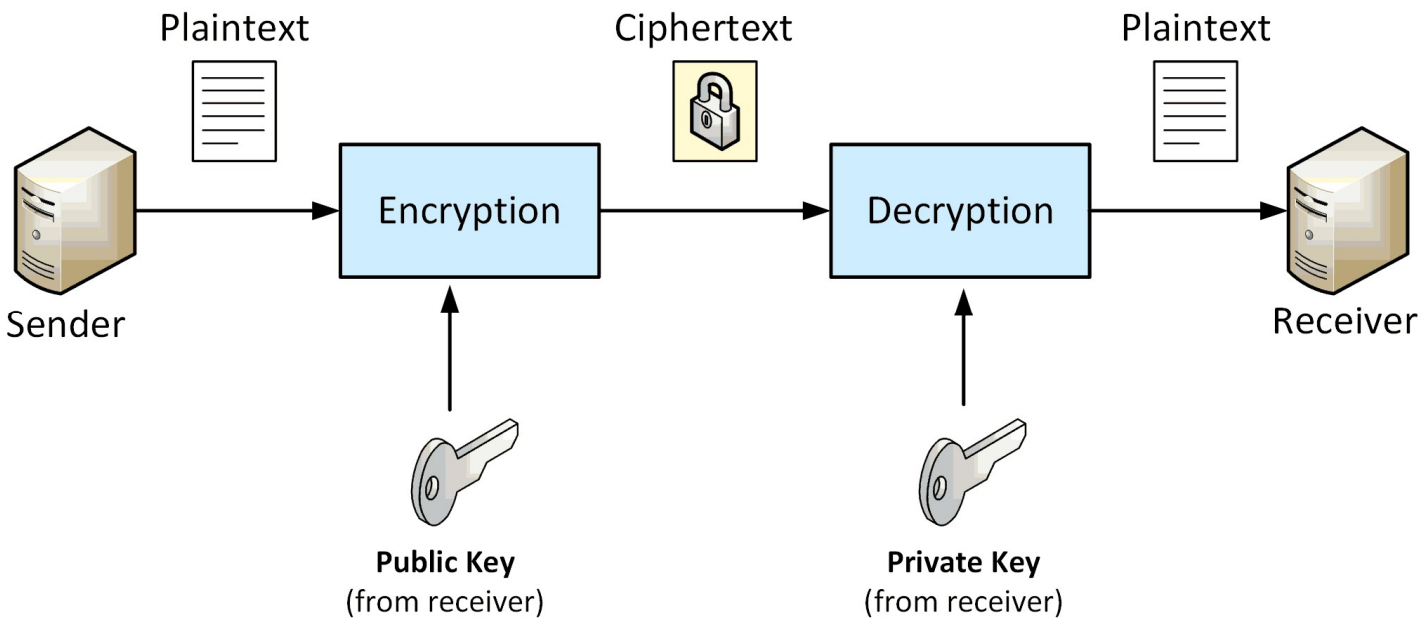
While the private key is a secrete, the public key can be distributed arbitrarily.

There is no need for secure exchange of keys.

Anyone who wants to send a secure message uses the freely available public key.

Encryption of Data at Rest

Asymmetric Encryption



Encryption of Data at Rest

Asymmetric Encryption

Common algorithms are:

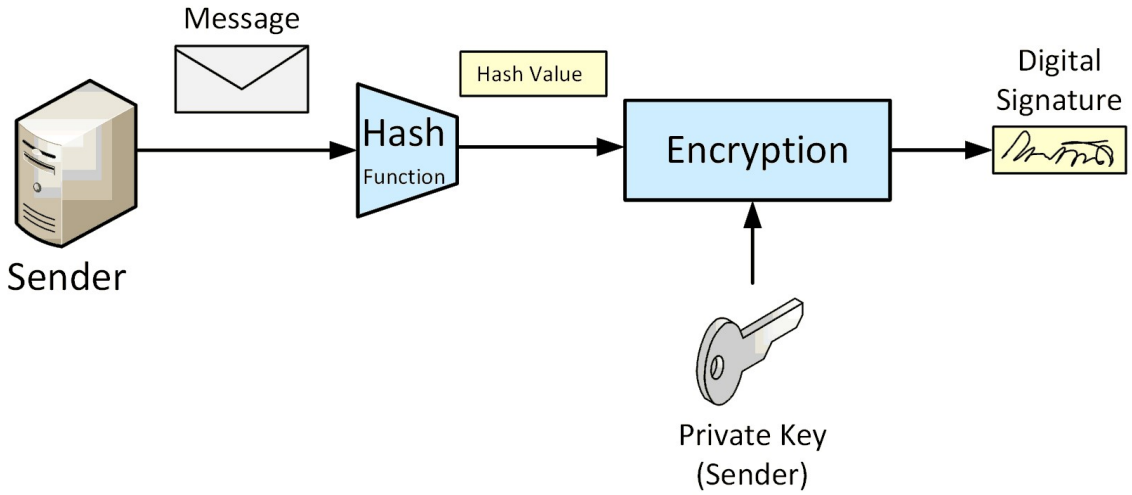
- **RSA** (named after the inventors Rivest, Shamir and Adleman)
- **ElGamal** (based on the Diffie–Hellman key exchange)
- **ECC** (Elliptic-Curve Cryptography)

After having created a **key pair**, we can do the encryption and decryption.

Encryption of Data at Rest

Digital Signatures

Digital signatures are used to ensure the **integrity of the message** sent to a recipient by ensuring the **identity of the sender** of the message. The solution is to hash a message and encrypt the hash value with the sender's private key.

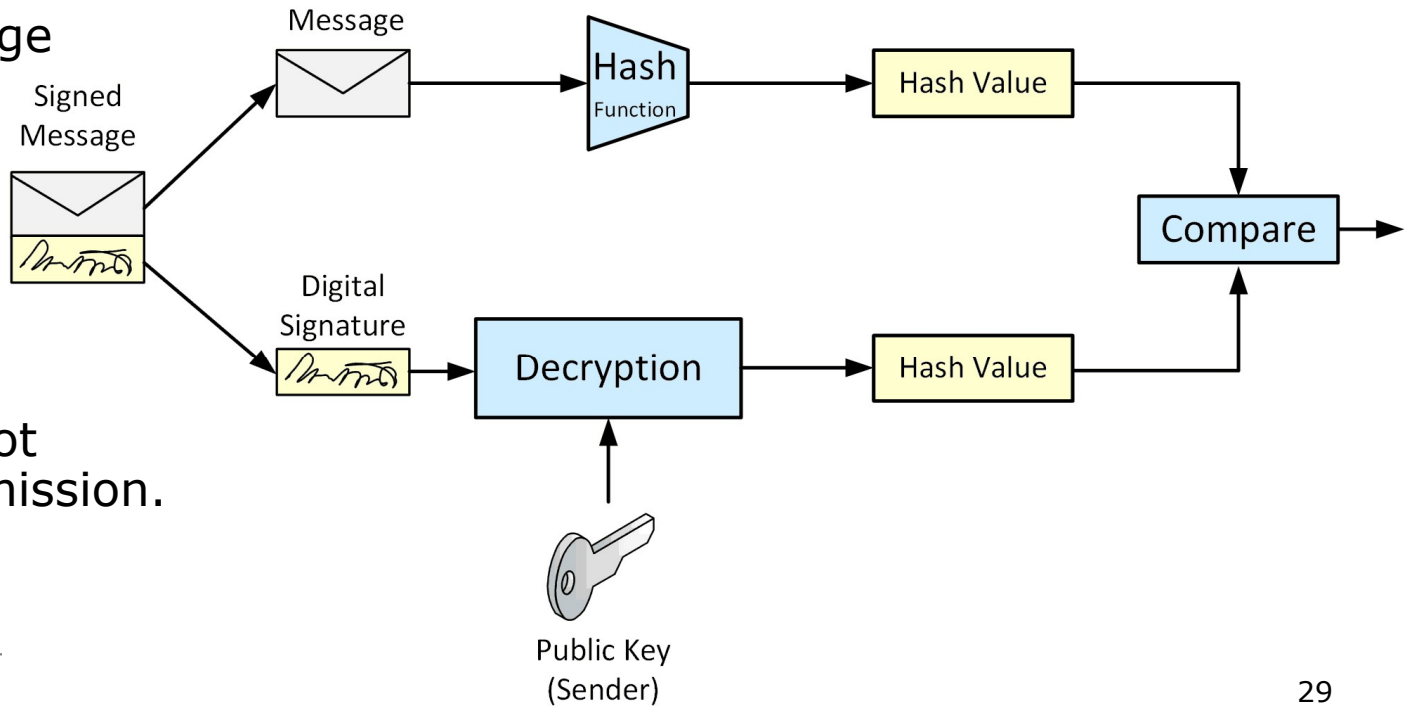


Encryption of Data at Rest

Digital Signatures

The receiver calculates the hash of the message and uses its public key to decrypt the encrypted hash.

If both hash values match, the message comes from the right transmitter and was not changed during transmission.



Encryption of Data at Rest

Digital Certificates

A digital certificate (**X.509 certificate**) is a document used to prove the **ownership of a public key** and includes **identification information** that is **digitally signed** by a trusted third party, also known as a **Certificate Authority (CA)**.

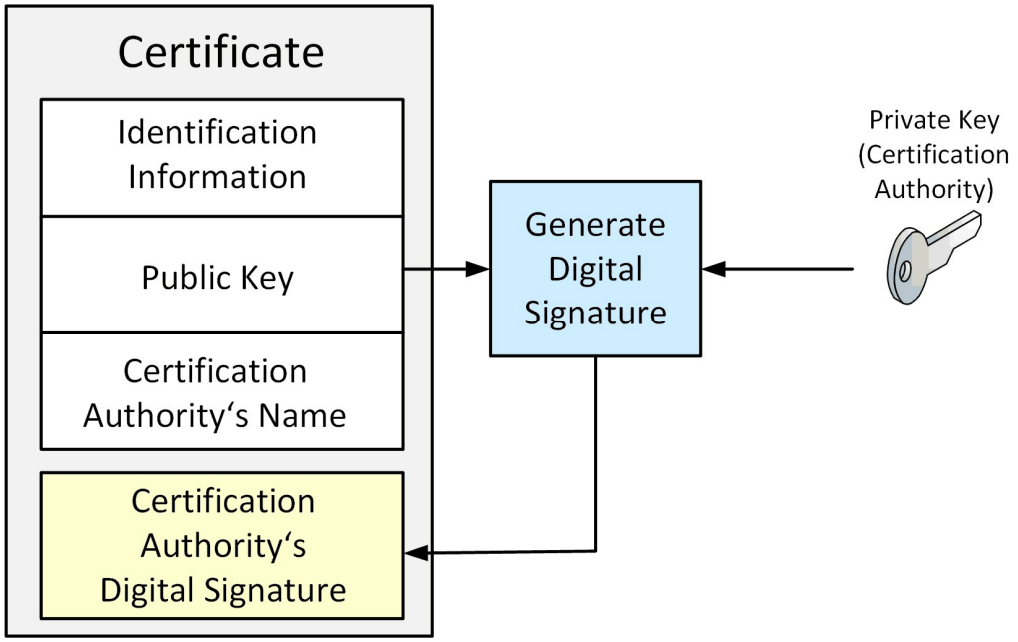
A digital certificate is commonly used to solve key management issues.

Digital certificates also play an important role in web applications. Therefore, root certificates are shipped together with the web browsers.

Encryption of Data at Rest

Digital Certificates

X.509 certificates also have an **expiration date** that indicates how long a certificate is valid.



Encryption of Data at Rest

FAQs

- Describe the **concept** and possible use cases for the following crypto primitives:
  - **Message digest**
  - **Key derivation functions**
  - **Message authentication codes**
  - **Symmetric encryption**
  - **Asymmetric encryption**
  - **Digital signatures**
- Explain the **benefits of using salt** in the context of storing passwords.
- Describe the **differences** between **block- and stream ciphers**.
- Describe the **concept** of **block cipher padding**.
- Describe the **differences** between **ECB and CBC block cipher mode**.
- Describe the **internal structure** of a **digital certificate**.

References

- *Christof Paar, Jan Pelzl, et al.*
  Understanding Cryptography: A Textbook for Students and Practitioners
  Springer 2010

- *Jean-Philippe Aumasson*
  Serious Cryptography
  No Starch Press 2018