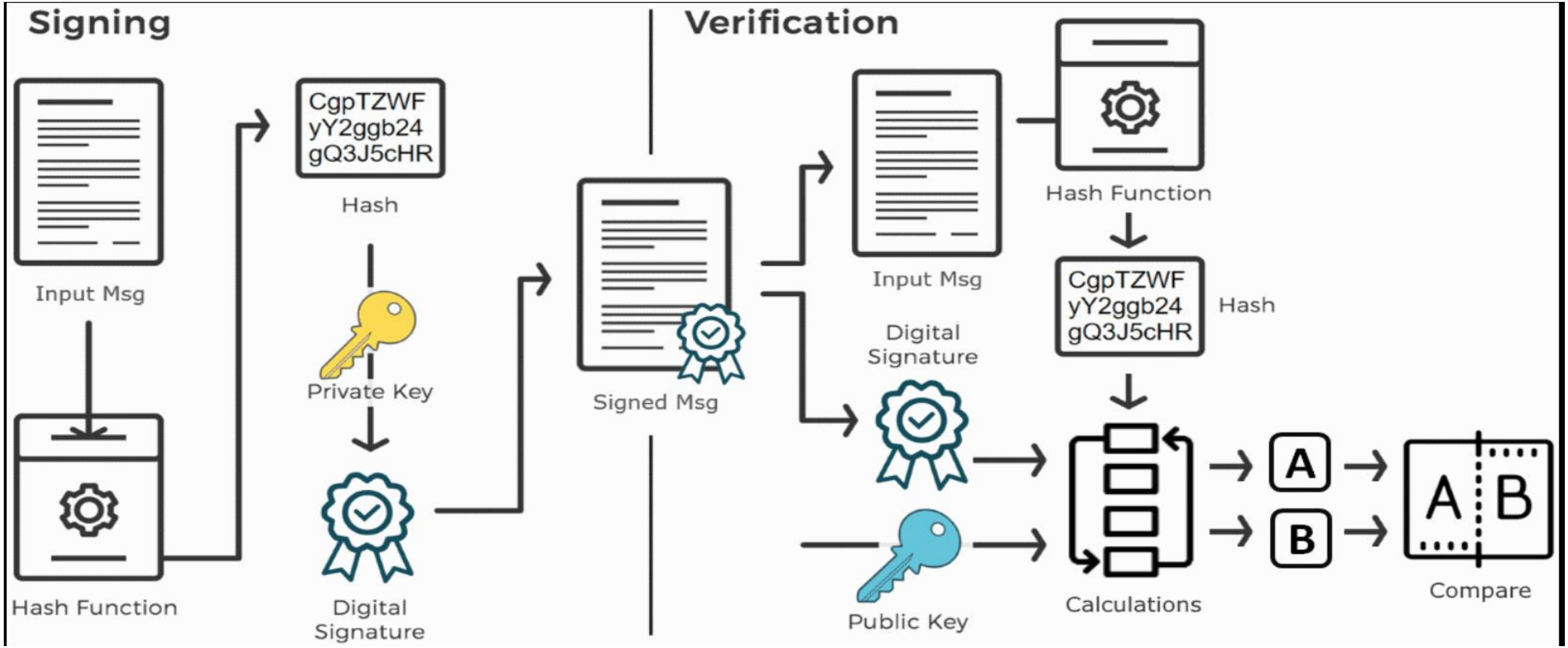
Digital signature using RSA

Signing Message and verifying Digital signature



RSA

- The RSA public-key cryptosystem provides a digital signature scheme (sign + verify),
  - based on the math of the modular exponentiations and discrete logarithms and the computational difficulty of the RSA problem (and its related integer factorization problem).
- Key Generation:
- The RSA algorithm uses keys of size 1024, 2048, 4096, ..., 16384 bits.
- RSA supports also longer keys (e.g. 65536 bits), but the performance is too slow for practical use (some operations may take several minutes or even hours).

Key Generation

The **RSA key-pair** consists of:

- public key {***n***, ***e***}
- private key {***n***, ***d***}

The numbers ***n*** and ***d*** are typically big integers (e.g. 3072 bits), while ***e*** is small, typically 65537.

By definition, the RSA key-pairs has the following property:

$(m^e)^d \equiv (m^d)^e \equiv m \pmod{n}$ for all ***m*** in the range [0...***n***)

RSA Sign

Signing a message ***msg*** with the private key exponent ***d***:

1. Calculate the message hash: ***h*** = hash(***msg***)
2. Encrypt ***h*** to calculate the signature: $s = h^d \pmod{n}$

The hash ***h*** should be in the range [0...***n***). The obtained **signature *s*** is an integer in the range [0...***n***).

RSA Verify Signature

Verifying a signature ***s*** for the message ***msg*** with the public key exponent ***e***:

1. Calculate the message hash: ***h*** = hash(***msg***)
2. Decrypt the signature: $h' = s^e \pmod n$
3. Compare ***h*** with ***h'*** to find whether the signature is valid or not

If the signature is correct, then the following will be true:

$$h' = s^e \pmod n = (h^d)^e \pmod n = h$$

The **RSA sign / verify algorithm** is pretty simple. Let's implement it with some code.

Summary

- How to sign the message
- How RSA used to sign the message
- Verification of digital signature