SE 4472

Information Security

Digital Signatures and RSA

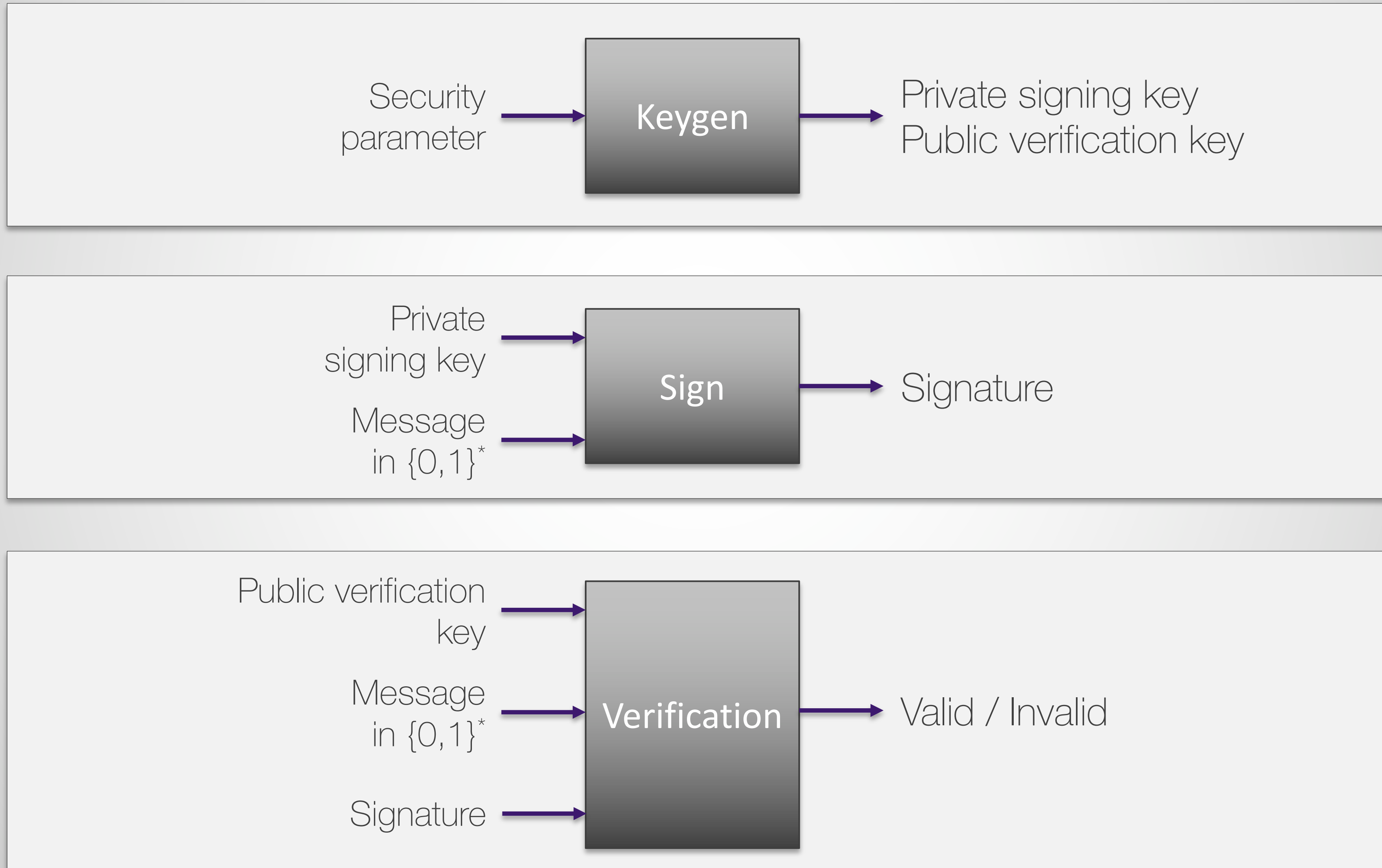
Prof. Aleksander Essex

Western Engineering

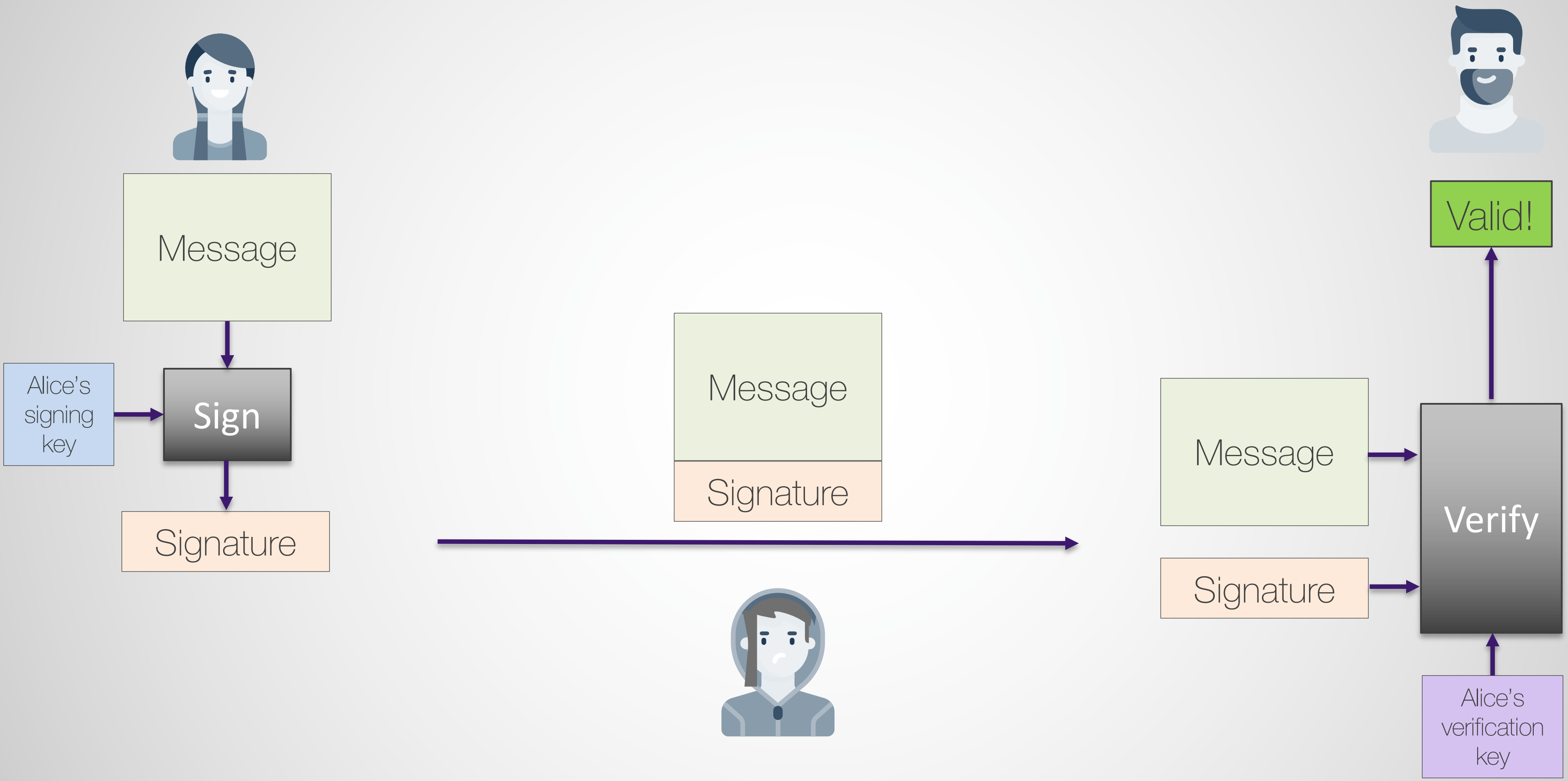
Digital Signatures

- A way to bind a message to a public key
- Two keys:
  - A private signing key
  - A public verification key
- Only the holder of the private signing key can generate valid signatures on a given message for a given public key
- Anyone can check that a given signature and message is consistent with a given public key
- Like public-key encryption, it should be hard to recover the private signing key given only the public verification key

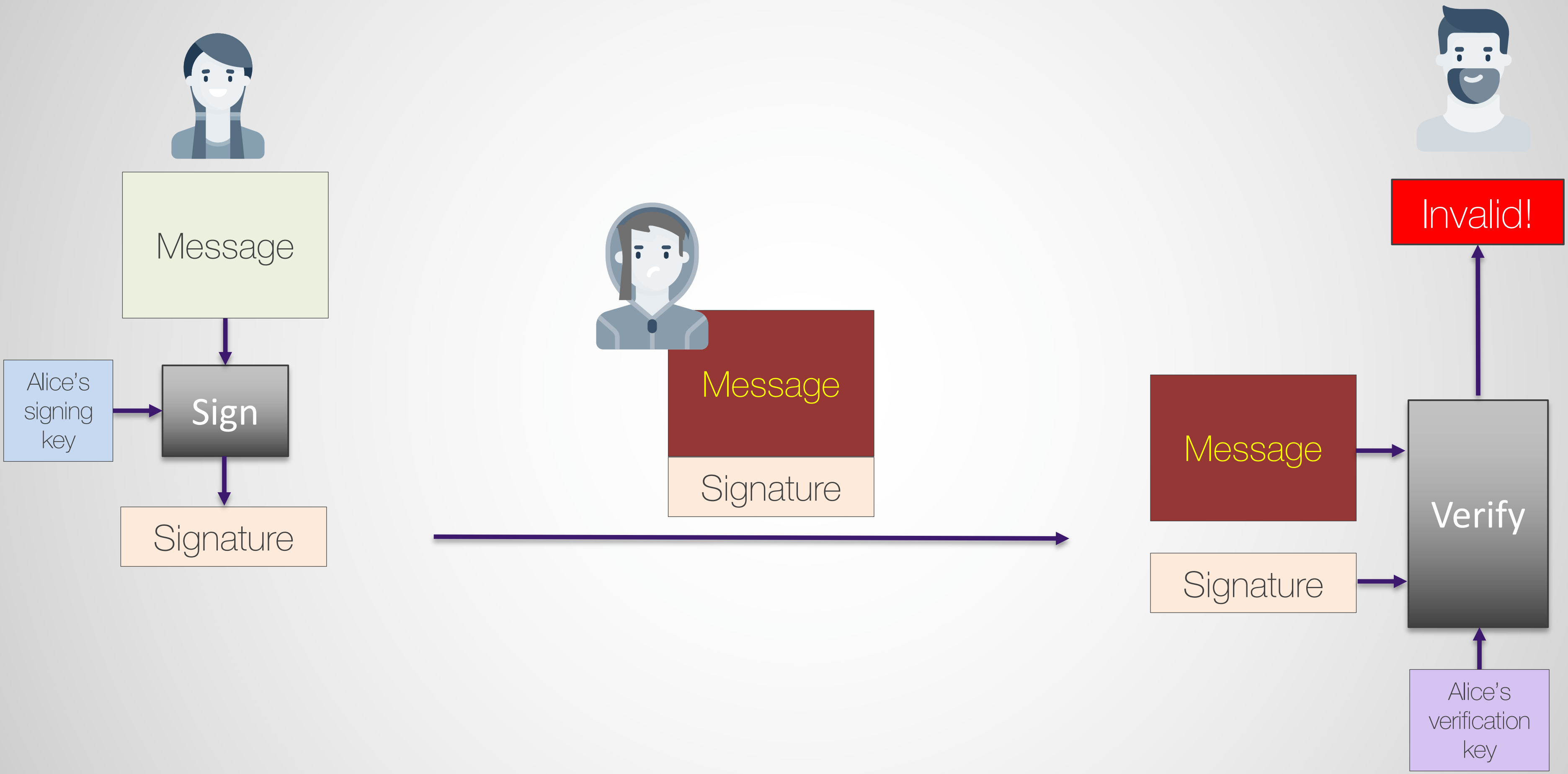
Signature Functionalities



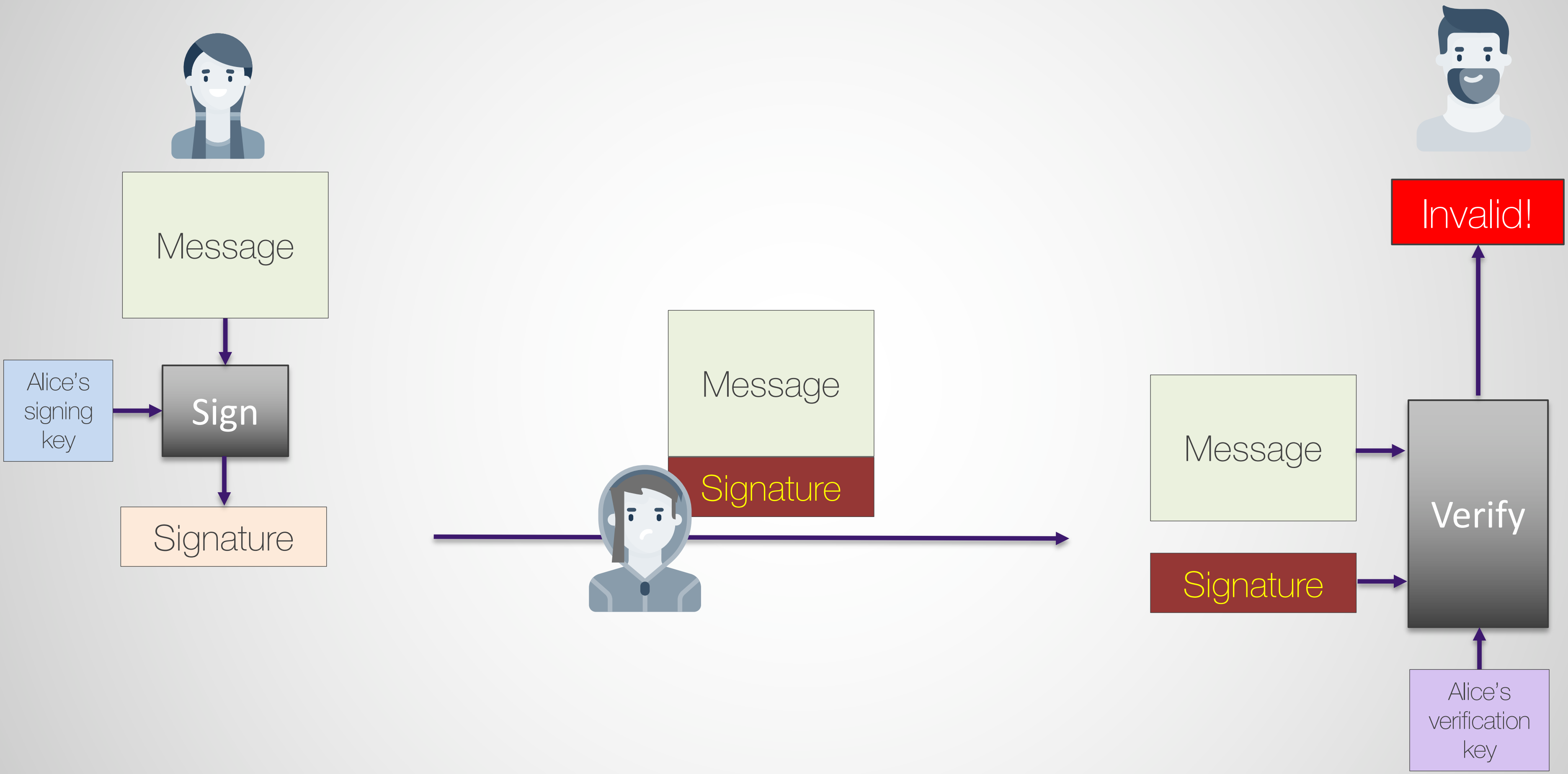
Signature workflow



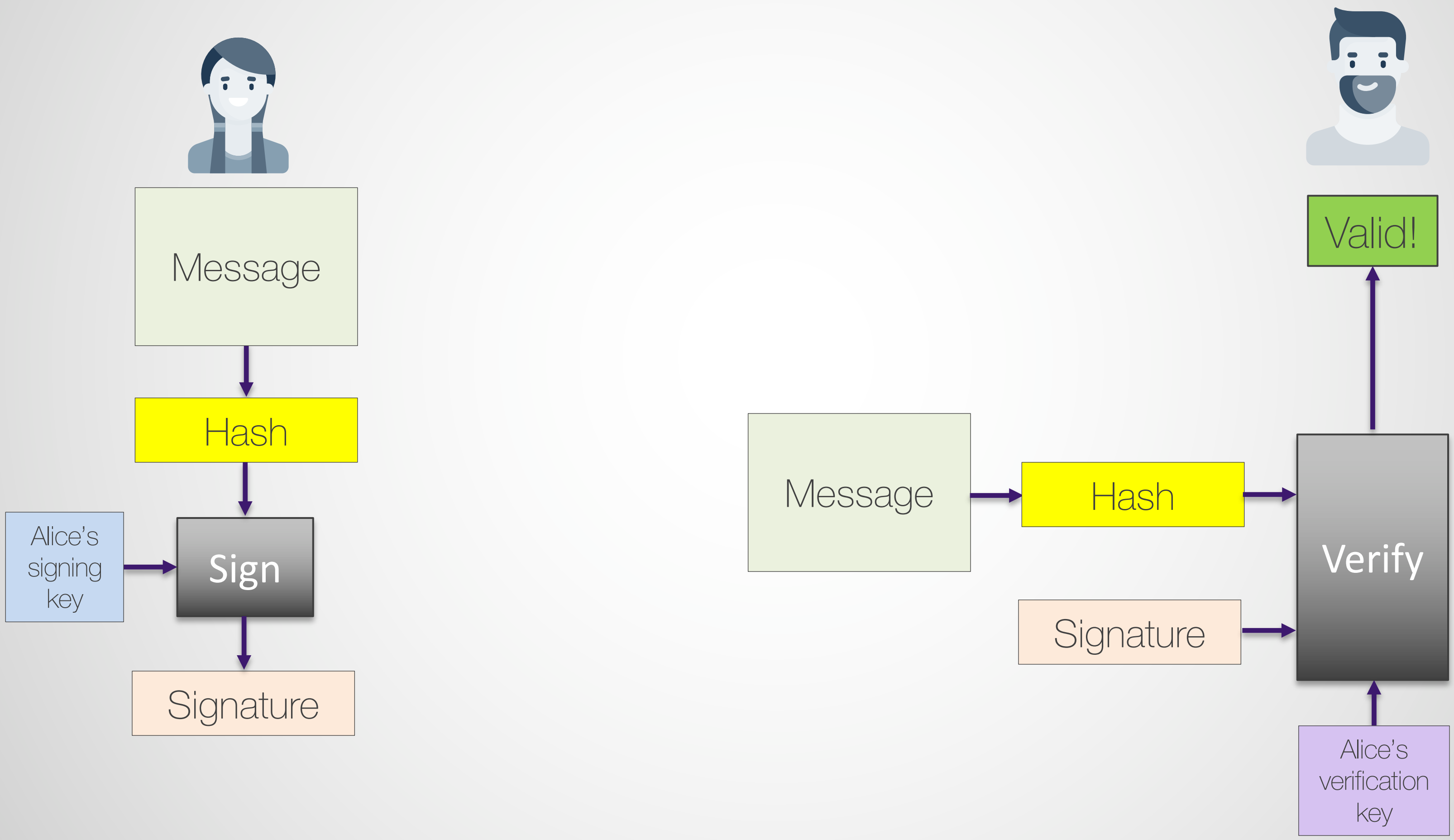
Eve modifies message



Eve modifies signature



Real signatures are done on hashes



Signatures Forgeries

- If Eve captures Alice's private signing key or can recover the signing key from the verification key, she can can create valid signatures on messages

- Forgeries are attacks creating valid signatures on a message *without* necessarily knowing the private signing key

Signatures Forgeries

- Universal Forgery
  - Eve can create a valid signature on any arbitrary message, even those provided by a challenger
- Selective Forgery
  - Eve can create a valid signature on a message she chooses ahead of time, but not necessarily on those provided by a challenger
- Existential Forgery
  - Eve can demonstrate *at least one* forgery, but the message may not be meaningful

Signatures with RSA

RSA

- First public-key encryption scheme invented by Rivest, Shamir and Adelman
- One of the most important developments in cryptography
- RSA for key agreement is not widely used these days and will be deprecated in TLS 1.3
- Still widely used in digital signatures

RSA

- Recall Fermat's Little Theorem:

  For a prime p and an integer a it is the case that:

$$a^{p-1} \equiv 1 \bmod p$$

But what if we worked modulo a composite n, i.e., where $n=p_1p_2\ldots p_k$? Euler's Theorem gives us:

$$a^{(p_1-1)(p_2-1)\ldots(p_k-1)} \equiv 1 \bmod n$$

RSA

- Let n=pq for large primes p and q and let

$$\phi = (p-1)(q-1)$$

Then as per Euler's theorem:

$$a^{\phi} \equiv 1 \bmod n$$

Which means

$$a^{\phi} \equiv a^{\phi \bmod \phi} \equiv a^{0} \bmod n$$

Or in other words, exponents are computed in the group of integers mod ϕ

RSA

- Suppose we choose two numbers, e and d such that:

$$ed \equiv 1 \bmod \phi$$

Now we have the building blocks to create a public-key encryption function

Textbook RSA

Public key: n, e

Private key: n, d

Encryption:

$$c = Enc(m) = m^e \mod n$$

Decryption:

$$m = Dec(c) = c^d \mod n$$

Textbook RSA

Why decryption works:

$$
\begin{aligned}
m = Dec(c) = c^d &\mod n \\
= (m^e)^d &\mod n \\
= m^{ed \mod \phi} &\mod n \\
= m^1 &\mod n \\
&= m
\end{aligned}
$$

Choices for e

- In the early days people chose encryption exponent to be really small, ie e=3, but there were a number of attacks
- We'd like to keep e small to keep the public-key operation efficient, but big enough to prevent attacks
- e=65537= 2^16+1 is a common choice
- d is then computed using the Euclidian algorithm as

$$d = e^{-1} \mod \phi$$

RSA Signatures

- Here's a first pass at turning RSA into a signature scheme:

Private signing key: d,n

Public verification key: e,n

Sign:

$$\sigma = \mathsf{Sign}(m) = m^d \mod n$$

Verify:

$$\mathsf{Verify}(m', \sigma):$$

$$\sigma^e \mod n \stackrel{?}{=} m'$$

Problem with Textbook RSA sigs

- Suppose you have two message/signature pairs:

$$(m_1, \sigma_1), (m_2, \sigma_2)$$

Let:

$$m_3 = m_1 m_2 \text{ and } \sigma_3 = \sigma_1 \sigma_2$$

Then:

$$(m_3, \sigma_3)$$

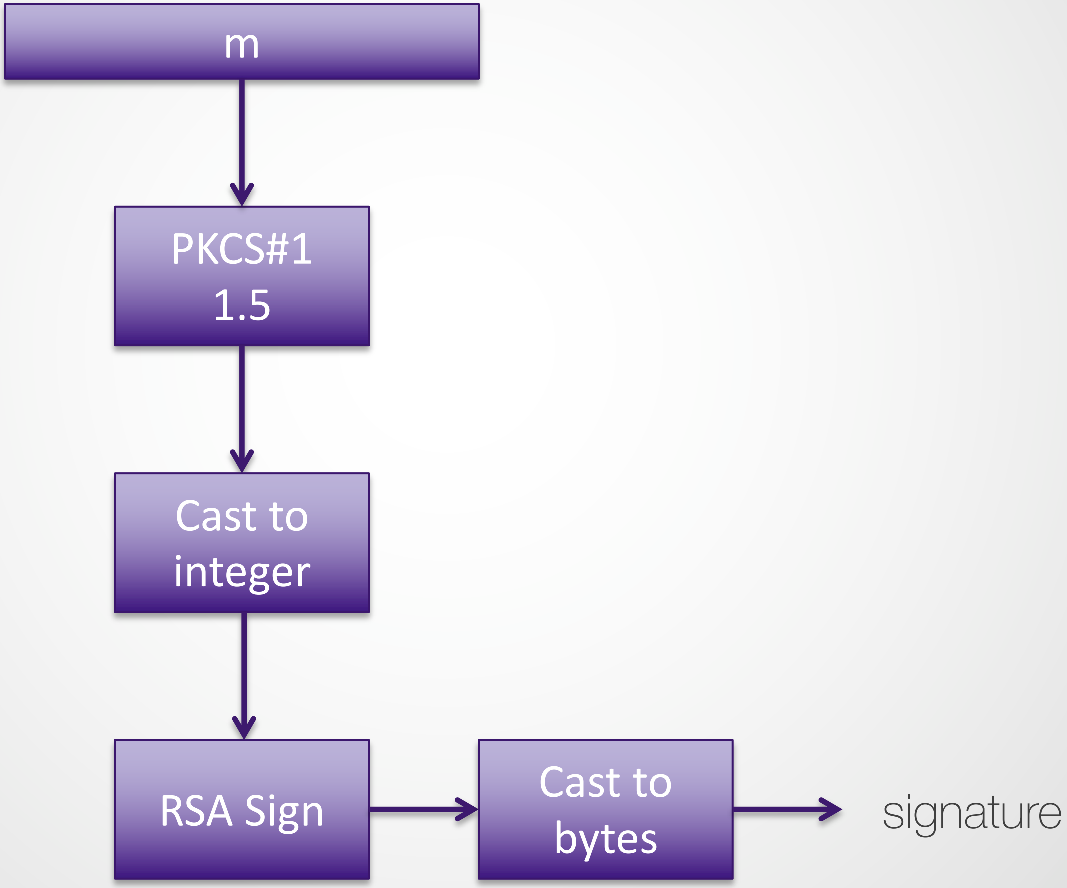
is a valid signature (see proof on board).

- Conclusion: Existential forgeries are easy to do!

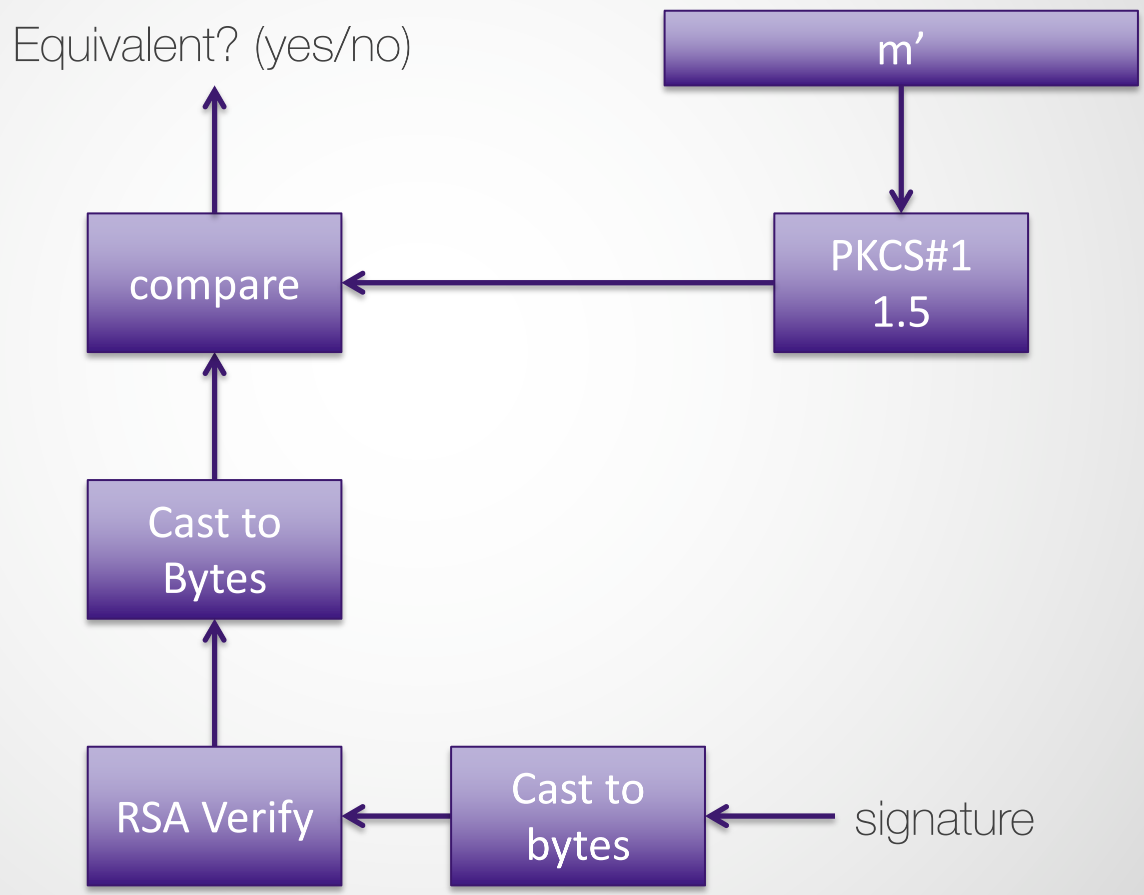
Solution: Use padding

- RSA PKCS#1 v1.5 padding:

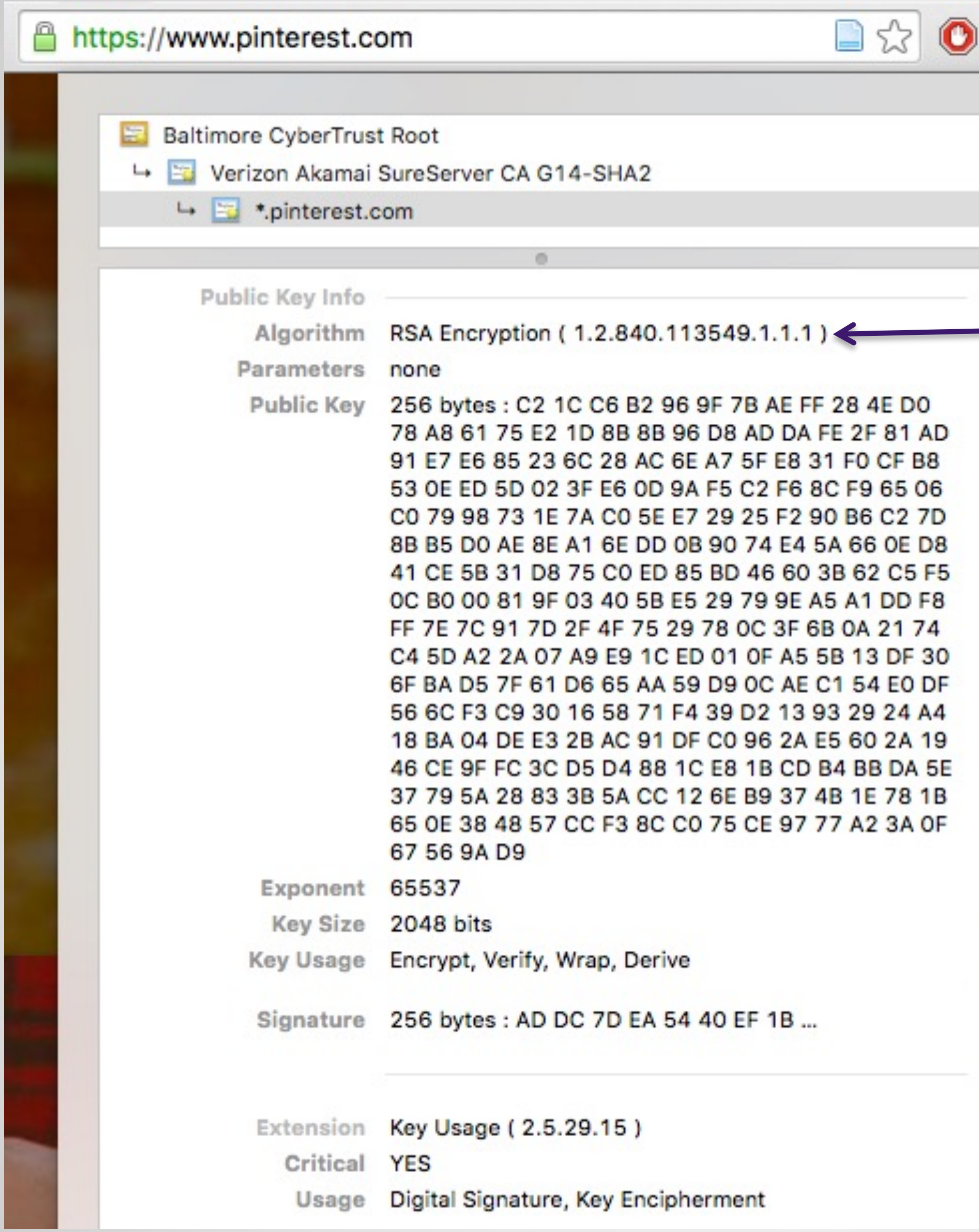
Signing



Verifying



Equivalent? (yes/no)
m’
compare
PKCS#1
1.5
Cast to
Bytes
RSA Verify
Cast to
bytes
signature



This says “we’re using PKCS#1 v1.5 padding”

Attacks

- Some implementations of this signature scheme did not check the entire padding. OOPS!
  - Loops over the FF's but doesn't count them
- Some people were using e=3
- Lets turn this into a signature forgery attack!

00 01 FF 01	H(m)	Attacker controlled

- Attacker cleverly chooses the right bytes such that the overall value is a power of 3.
- Attack computes the cube root of message (notice doesn't need the signing key!) When verifier computes the cube, the message is restored with the "correct" padding