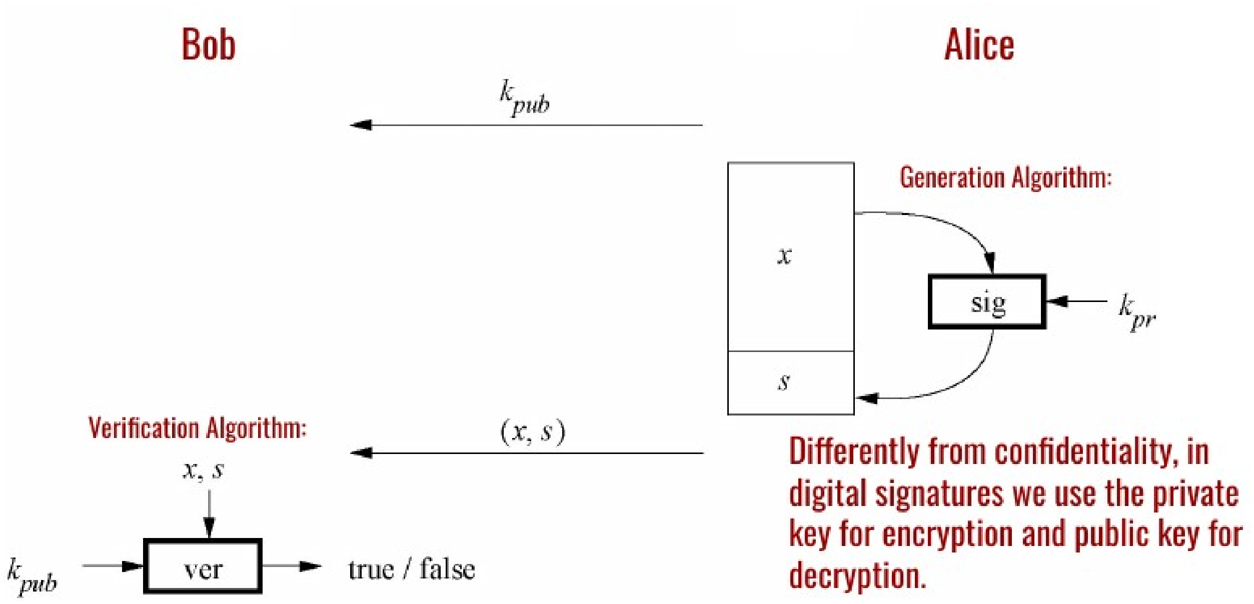
10. Digital Signatures

To ensure non repudiaion; symmetric can fails and MAC cannot ensure non repudiation

Use public-key.

10.1 Principles



Bob
Alice
k_{pub}
Generation Algorithm:
x
sig
k_{pr}
s
Verification Algorithm:
(x, s)
x, s
k_{pub}
ver
true / false
Differently from confidentiality, in digital signatures we use the private key for encryption and public key for decryption.

Signature must change for every document.

X and S input vor verification.

Security services

Integrity: ensures no modifications.

Message authentication: encuse the sender is authentic.

Non-repudiation: ensure not deny creation

10.2 DS Forgery

Ability to create a pair of message and signature where x has not been signed in the past by the legitimate signer.

10.2.1 Universal Forgery

Adversary creates a valid signature s for any given message x .

It is the strongest ability in forging and it implies the other types of forgery.

10.2.2 Selective Forgery

adversary creates a message/signature pair (x, s) where x has been chosen by the challenger prior to the attack.

Implies Existential

10.2.3 Existential Forgery

adversary creates at least one message/signature pair (x,s), where s was not produced by the legitimate signer.

Can choose x freely; easier than selective. The strongest is Existentially unforgeable.

10.3 Implementation

2 approaches:

- sign a message x using the public key: (x, s = E_{pub}(x)) (everybody can forge a signature)
- sign a message x using the (my) private key: (x, s = E_{priv}(x)) (still affected by existential)

10.3.1 Signing using pivate key

Assume RSA

1. To generate the private and public key: use the same key generation as RSA encryption
2. To generate the signature: "encrypt" the message x with the private key

$$s = sig_{Kpriv}(x) = x^d \bmod n$$

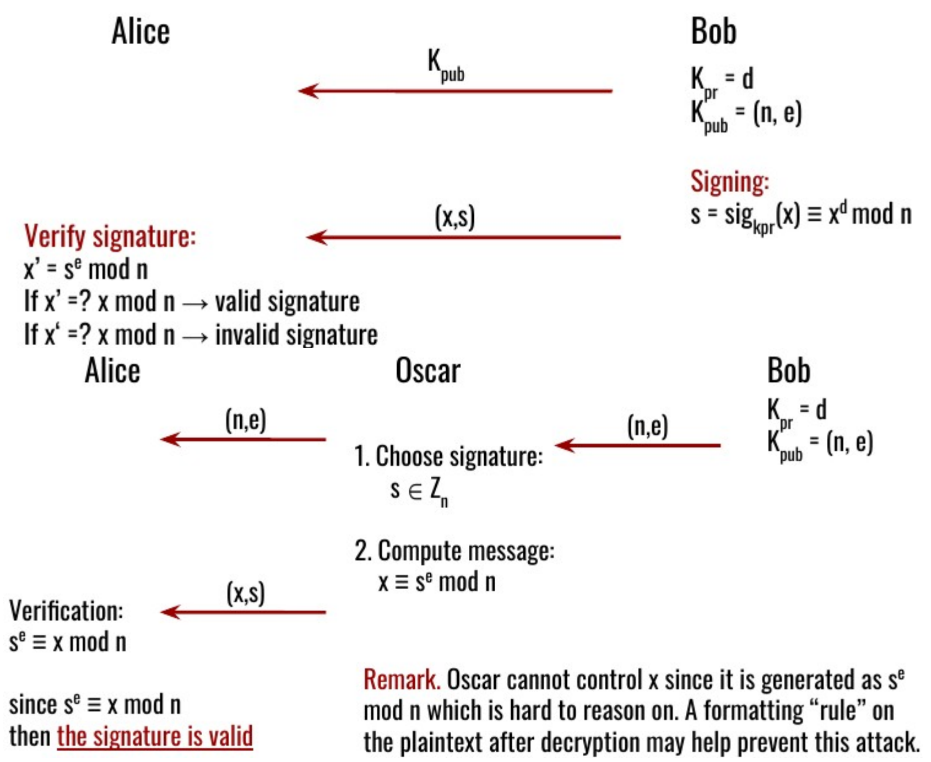
then append s to x by sending (x, s)

3. To verify the signature: "decrypt" the signature with the public key

$$x' = ver_{Kpub}(s) = s^e \bmod n$$

If x=x', the signature is valid.

Assume we are using RSA as PK scheme for implementing the digital signature scheme:



Remark. Oscar cannot control x since it is generated as s^e mod n which is hard to reason on. A formatting "rule" on the plaintext after decryption may help prevent this attack.

Issue: Malleability

an attacker can transform a ciphertext into another ciphertext which decrypts to a related plaintext without knowing the private key.

f.i.: $(x1\ x2)^e \bmod n = x1^e x2^e \bmod n$

10.3.2 PKCS#1 DS

2 Schemes:

1. RSASSA-PSS: Signature Scheme with Appendix based on a Probabilistic Signature Scheme using encoding EMSA-PSS.

2. RSASSA-PKCS1-v1_5 (older): Signature Scheme with Appendix using encoding EMSA-PKCS1-v1_5.

“Appendix” means that instead of signing the message, they sign the hash of the message

it provides:

1. Efficiency (RSA is slow)

2. Message overhead (the message is longer)

Use of MGF: **A Mask Generation Function (MGF)** is a cryptographic primitive similar to a cryptographic hash function except that while a hash function's output is a fixed size, a MGF supports output of a variable length.

The most common mechanism s iteratively apply hash.

RSASSA-PSS Encoding

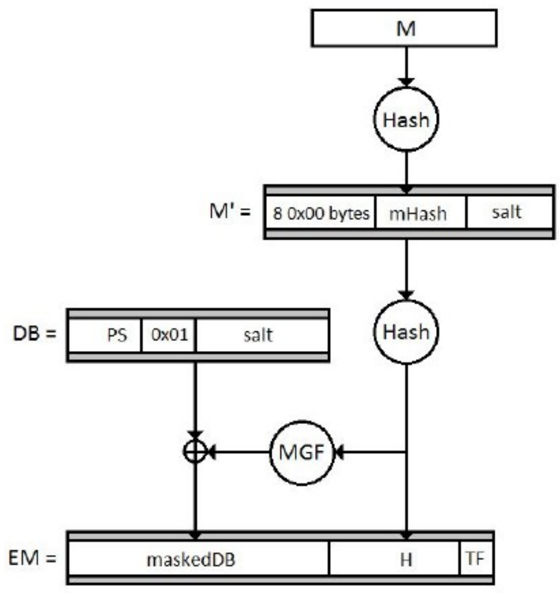


Figure 2: EMSA-PSS encoding

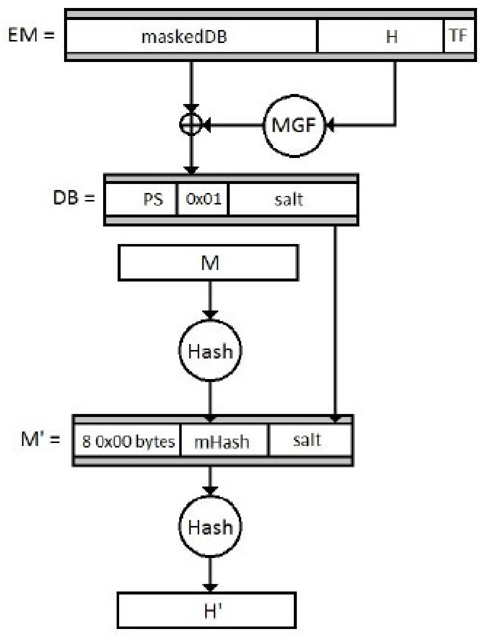


Figure 1: EMSA-PSS verification

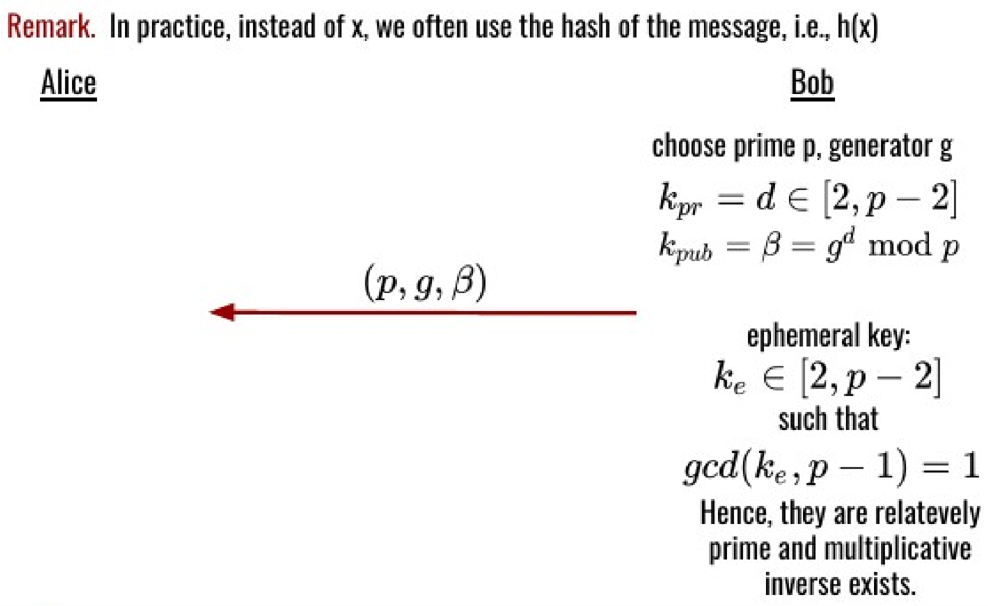
RSASSA-PKCS1-v1_5

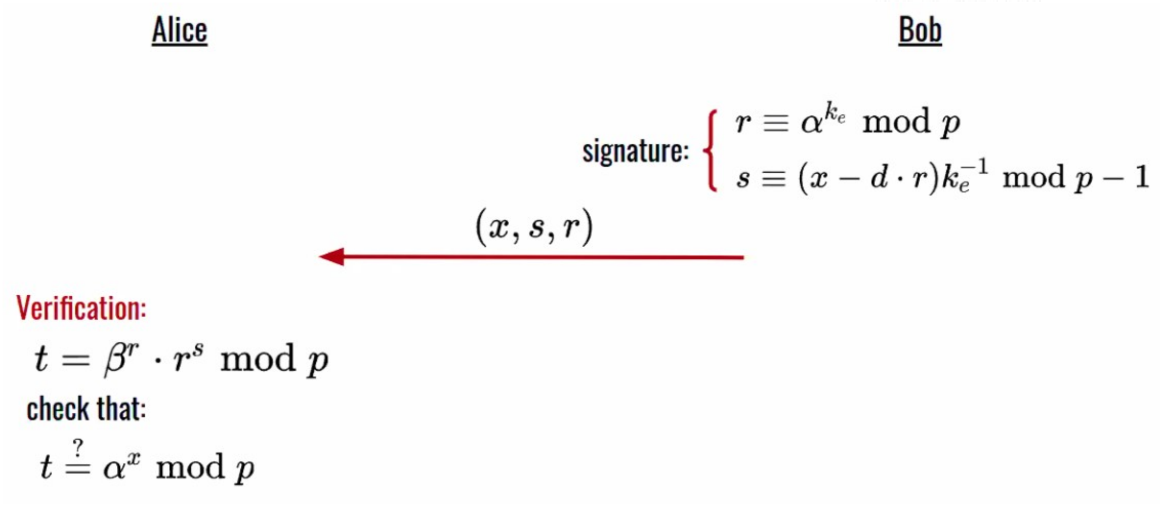
Message is padded as:

M' = 0x00 ‖ 0x01 ‖ at least 8 0xFF bytes ‖ 0x00 ‖hash specs‖ hash(M)

After decryption of a ciphertext, check expected patterns in the plaintext (e.g., look for 0x01).

10.3.3 Elgamal Signature scheme


Remark. In practice, instead of x, we often use the hash of the message, i.e., h(x)
Alice
Bob
choose prime p, generator g
$k_{pr} = d \in [2, p-2]$
$k_{pub} = \beta = g^d \bmod p$
(p, g, β)
ephemeral key:
$k_e \in [2, p-2]$
such that
$gcd(k_e, p-1) = 1$
Hence, they are relatevely prime and multiplicative inverse exists.


Alice
Bob
signature: $\begin{cases} r \equiv \alpha^{k_e} \bmod p \\ s \equiv (x - d \cdot r)k_e^{-1} \bmod p-1 \end{cases}$
(x, s, r)
Verification:
$t = \beta^r \cdot r^s \bmod p$
check that:
$t \overset{?}{=} \alpha^x \bmod p$

alfa is the generator that we are using for our cyclic group.

k_e is the ephemeral key

This works because:

$$t = \beta^r \cdot r^s \equiv (\alpha^d)^r \cdot (\alpha^{k_e})^s \bmod p$$

if want $t \equiv \alpha^x \bmod p$ then we need:

$$(\alpha^d)^r \cdot (\alpha^{k_e})^s \equiv \alpha^x \bmod p$$
$$dr + k_e s \equiv x \bmod p$$

Given a corollary of the Fermat's Little Theorem:

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

This means that we can do mod arithmetic p-1 on the exponent when doing mod arithmetic p (prime!)

we have to show that:

$$dr + k_e s \equiv x \bmod p-1$$
$$k_e s \equiv x - dr \bmod p-1$$
$$s \equiv (x - dr)k_e^{-1} \bmod p-1$$

which exactly how s has been constructed.

Given two signatures using the same ephemeral key, we have:

$$s_1 \equiv (x_1 - d\,r)k_E^{-1} \bmod p-1$$
$$s_2 \equiv (x_2 - d\,r)k_E^{-1} \bmod p-1$$

This system can be solved as:

$$s_1 - s_2 \equiv (x_1 - x_2)k_E^{-1} \bmod p-1$$

from which:

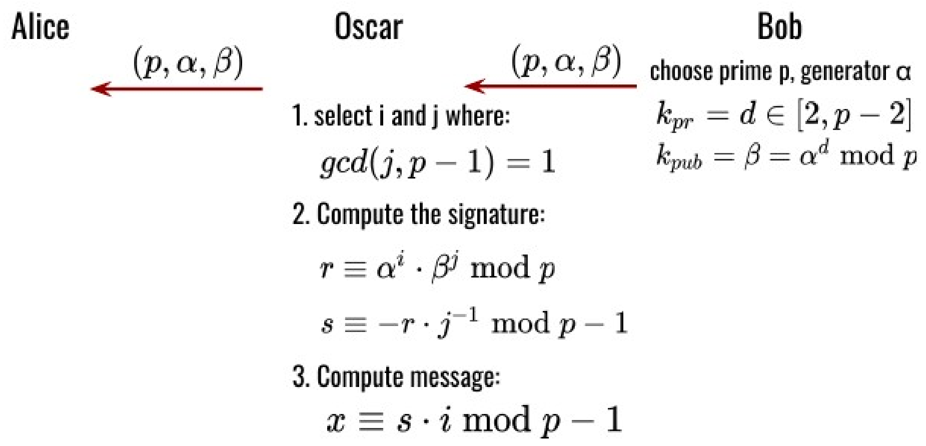
$$k_E \equiv \frac{x_1 - x_2}{s_1 - s_2} \bmod p-1$$

If: $\gcd(s_1 - s_2, p-1) \neq 1$

Then there are multiple solutions for the ephemeral key, which must be checked to find the correct one. The private key can then found as:

$$d \equiv \frac{x_1 - s_1 k_E}{r} \bmod p-1$$

Elgamal: existential forgery attack


Alice

Oscar

Bob

(p, α, β)

(p, α, β)

choose prime p, generator α

$k_{pr} = d \in [2, p-2]$

$k_{pub} = \beta = \alpha^d \bmod p$

1. select i and j where:

$gcd(j, p-1) = 1$

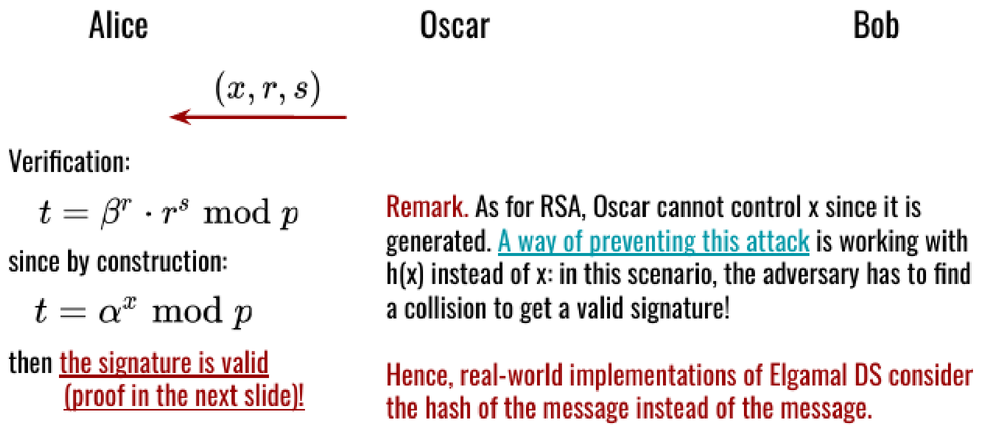
2. Compute the signature:

$r \equiv \alpha^i \cdot \beta^j \bmod p$

$s \equiv -r \cdot j^{-1} \bmod p-1$

3. Compute message:

$x \equiv s \cdot i \bmod p-1$


Alice

Oscar

Bob

(x, r, s)

Verification:

$t = \beta^r \cdot r^s \bmod p$

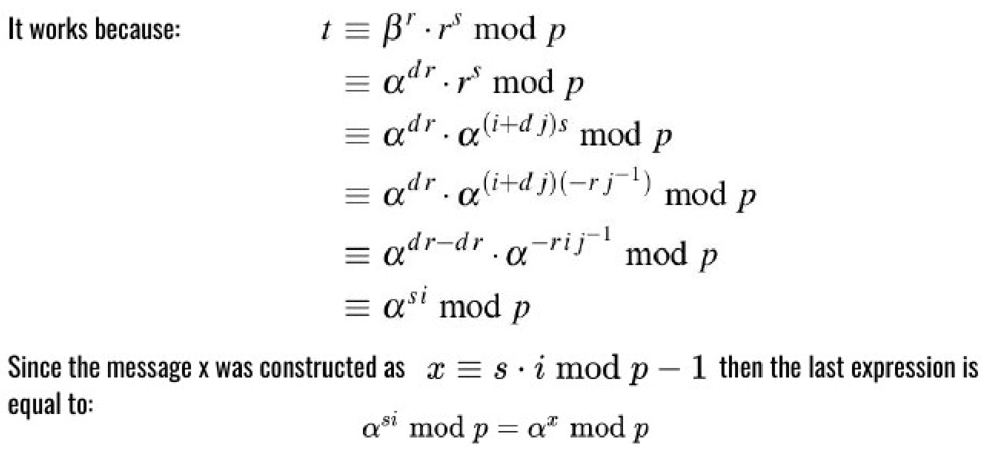
since by construction:

$t = \alpha^x \bmod p$

then the signature is valid (proof in the next slide)!

Remark. As for RSA, Oscar cannot control x since it is generated. A way of preventing this attack is working with h(x) instead of x: in this scenario, the adversary has to find a collision to get a valid signature!

Hence, real-world implementations of Elgamal DS consider the hash of the message instead of the message.


It works because:

$$
\begin{aligned}
t &\equiv \beta^r \cdot r^s \bmod p \\
&\equiv \alpha^{dr} \cdot r^s \bmod p \\
&\equiv \alpha^{dr} \cdot \alpha^{(i+dj)s} \bmod p \\
&\equiv \alpha^{dr} \cdot \alpha^{(i+dj)(-rj^{-1})} \bmod p \\
&\equiv \alpha^{dr-dr} \cdot \alpha^{-rij^{-1}} \bmod p \\
&\equiv \alpha^{si} \bmod p
\end{aligned}
$$

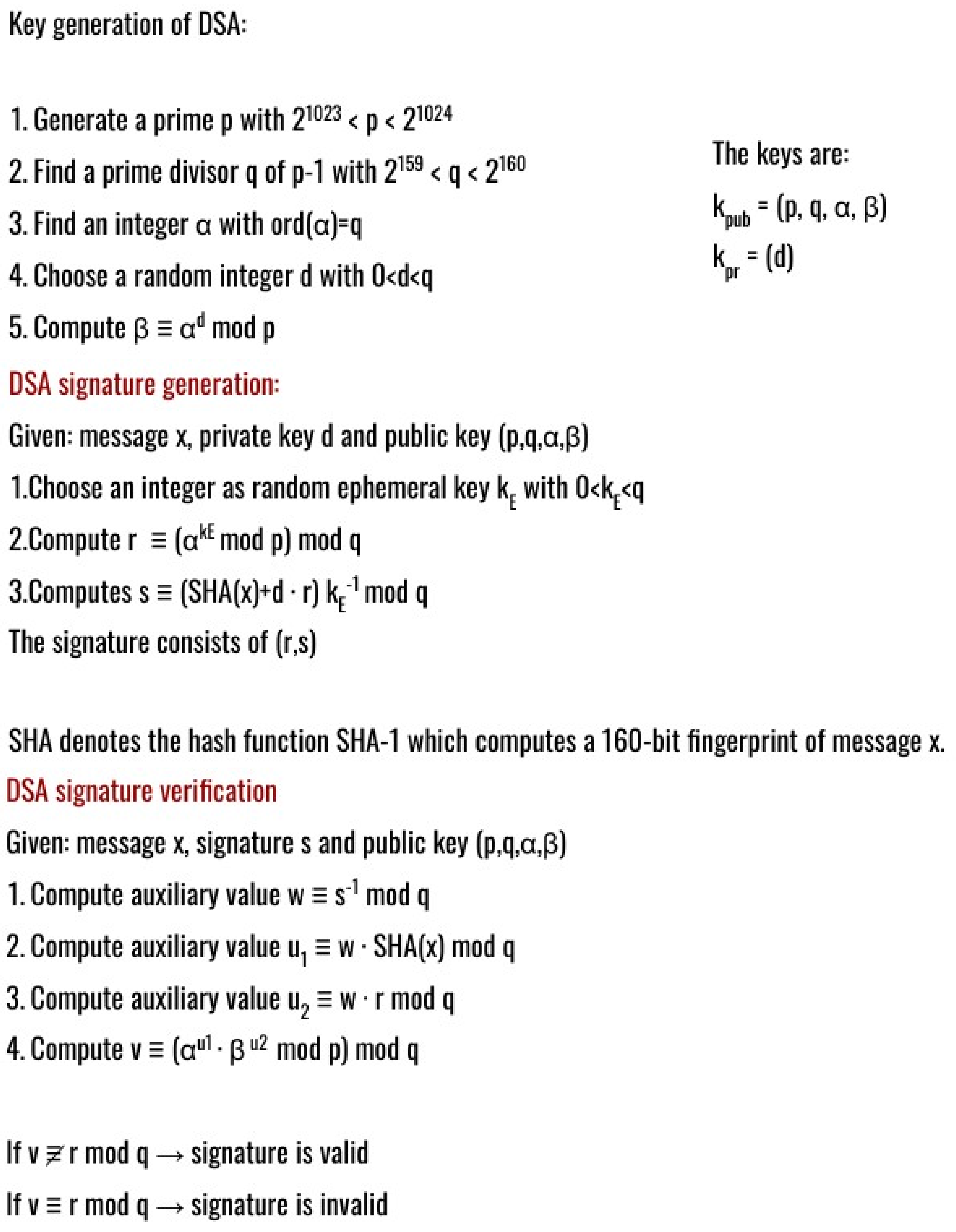
Since the message x was constructed as $x \equiv s \cdot i \bmod p-1$ then the last expression is equal to:

$$\alpha^{si} \bmod p = \alpha^x \bmod p$$

Digital signature Standard

NIST defined **Digital Signature Algorithm (DSA)**, which is based on the Elgamal signature scheme and SHA-1. Signature is only 320 bits long and it is faster in DSA than in RSA, verification slower in DSA vs RSA.

The algorithm



Key generation of DSA:

1. Generate a prime p with $2^{1023} < p < 2^{1024}$
2. Find a prime divisor q of p-1 with $2^{159} < q < 2^{160}$
3. Find an integer α with $ord(\alpha)=q$
4. Choose a random integer d with $0<d<q$
5. Compute $\beta \equiv \alpha^{d} \bmod p$

The keys are:

$k_{pub} = (p, q, \alpha, \beta)$

$k_{pr} = (d)$

DSA signature generation:

Given: message x, private key d and public key (p,q,α,β)

1.Choose an integer as random ephemeral key k_E with $0<k_E<q$

2.Compute $r \equiv (\alpha^{kE} \bmod p) \bmod q$

3.Computes $s \equiv (SHA(x)+d \cdot r)\, k_E^{-1} \bmod q$

The signature consists of (r,s)

SHA denotes the hash function SHA-1 which computes a 160-bit fingerprint of message x.

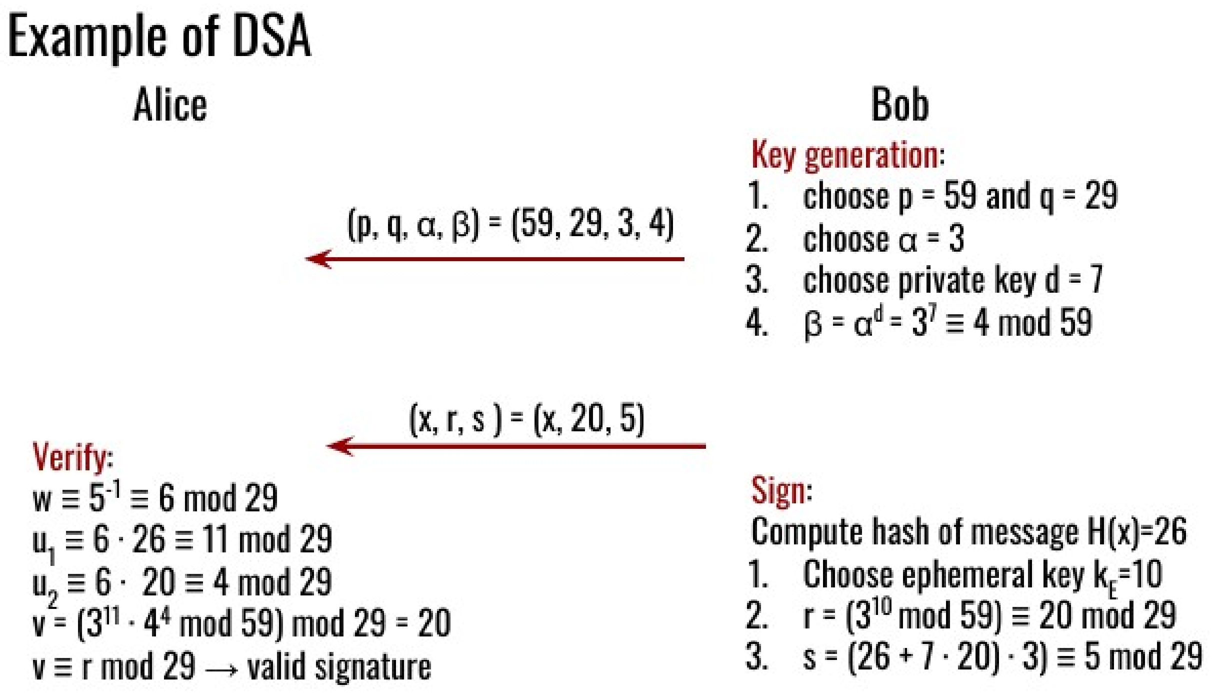
DSA signature verification

Given: message x, signature s and public key (p,q,α,β)

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $v \equiv (\alpha^{u1} \cdot \beta^{u2} \bmod p) \bmod q$

If $v \not\equiv r \bmod q \rightarrow$ signature is valid

If $v \equiv r \bmod q \rightarrow$ signature is invalid



Example of DSA

Alice

Bob

Key generation:

1. choose $p = 59$ and $q = 29$
2. choose $\alpha = 3$
3. choose private key $d = 7$
4. $\beta = \alpha^d = 3^7 \equiv 4 \bmod 59$

$(p, q, \alpha, \beta) = (59, 29, 3, 4)$

$(x, r, s) = (x, 20, 5)$

Verify:

$w \equiv 5^{-1} \equiv 6 \bmod 29$

$u_1 \equiv 6 \cdot 26 \equiv 11 \bmod 29$

$u_2 \equiv 6 \cdot 20 \equiv 4 \bmod 29$

$v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$

$v \equiv r \bmod 29 \longrightarrow$ valid signature

Sign:

Compute hash of message $H(x)=26$

1. Choose ephemeral key $k_E=10$
2. $r = (3^{10} \bmod 59) \equiv 20 \bmod 29$
3. $s = (26 + 7 \cdot 20) \cdot 3) \equiv 5 \bmod 29$

DSA: reuse of ephemeral key

As in Elgamal Signature Scheme, there exists an attack to recover the private key when two messages are signed using the same ephemeral key. Never do that!

Elliptic curves vs digital signatures

The Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.

10.4 Timestamping a document

TimeStamping Authority (TSA): guarantees timestamp of a document.

Alice (A) wants to timestamp a document:

1. A compute hash of document and sends to TSA

2. TSA adds timestamp, computes new hash (of timestamp and received hash) and sign the obtained hash; sends back to A

3. A keeps TSA's signature as a proof

Everybody can check the signature. TSA does not know Alice's document.

Two or more parties signing the same document?

Multisignature (multi-signature) is a digital signature scheme which allows a group of users to sign a single document. Usually, a multisignature algorithm produces a joint signature that is more compact than a collection of distinct signatures from all users.

Nowadays, multisignature has applications in cryptocurrency transactions.

Monks at Mt Athos would secure their crypts with multiple keys, with more than one being needed to unlock the crypt. This meant that no single monk could access any precious relics without the awareness of at least one other monk.

Even a single user may use multisignature: several private keys that are stored in different ways.

10.5 Common applications

1. Code signins: to ensure that sw versions are not modified.

2. Verifying public key: we look into this in another lecture (X509).

3.Signing emails: mail providers use protocols, such as DomainKeys Identified Mail (DKIM), to perform email authentication and detect forged sender addresses in emails (email spoofing). (Only in Italy there is PEC…)

4. Credit-card payments: e.g., EMV (Europay, Mastercard, and Visa).