

Digital Signatures

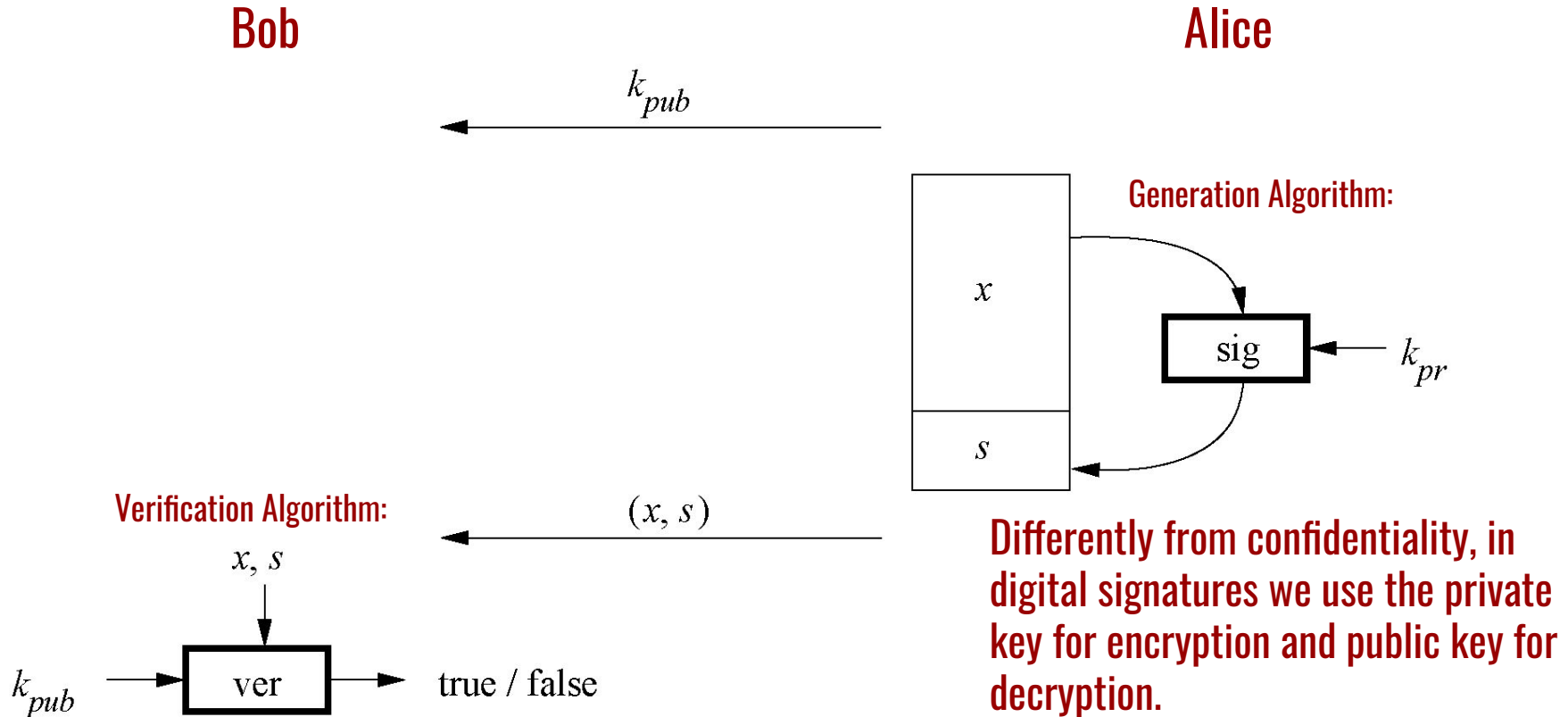
Computer and Network Security

Emilio Coppa

Motivation

- Alice orders a pink car from the car salesman Bob
- After seeing the pink car, Alice states that she has never ordered it:
- How can Bob prove towards a judge that Alice has ordered a pink car? (And that he did not fabricate the order himself)
 - This security property is called **non repudiation**
 - Symmetric cryptography fails because both Alice and Bob can be malicious: hence MAC cannot be used when non repudiation is required
 - Can be achieved with public-key cryptography

Basic Principle of Digital Signatures



Main idea

- For a given message x , a digital signature is appended to the message (just like a conventional signature).
- Only the person with the private key should be able to generate the signature.
- The signature must change for every document.
- The signature is realized as a function with the message x and the private key as input.
- The public key and the message x are the inputs to the verification function.

Security Services from Digital Signatures

- **Integrity**: ensures that a message has not been modified in transit.
- **Message Authentication**: ensures that the sender of a message is authentic.
An alternative term is data origin authentication.
- **Non-repudiation**: ensures that the sender of a message can not deny the creation of the message. (c.f. order of a pink car)

Digital Signature Forgery

Forgery is ability to create a pair consisting of a message x and a signature (or MAC) s that is valid for x , where x has not been signed in the past by the legitimate signer.

Types of forgery:

1. **Existential Forgery**
2. **Selective Forgery**
3. **Universal Forgery**

We consider the role of a “challenger” that challenges the adversary to correctly sign a message

Universal Forgery

- adversary creates a valid signature s for any given message x (chosen by the adversary or by a challenger)
- it is the strongest ability in forging and it implies the other types of forgery

Selective Forgery

- adversary creates a message/signature pair (x, s) where x has been chosen by the challenger prior to the attack
- x may be chosen to have interesting mathematical properties with respect to the signature algorithm; however, in selective forgery, x must be fixed before the start of the attack
- the ability to successfully conduct a selective forgery attack implies the ability to successfully conduct an existential forgery attack

Existential Forgery

- adversary creates at least one message/signature pair (x,s) , where s was not produced by the legitimate signer.
- adversary can choose x freely: **x need not have any particular meaning**
- existential forgery is essentially the weakest adversarial goal
- creating an existential forgery is easier than a selective forgery, because the attacker may select x for which a forgery can easily be created, whereas in the case of a selective forgery, the challenger can ask for the signature of a “difficult” message.
- therefore the strongest schemes are those which are “existentially unforgeable”

How to implement a DS scheme based on a PK scheme?

We may consider two approaches:

- sign a message x using the public key: $(x, s = E_{\text{pub}}(x))$
 - **problem: public keys are public, then everybody can forge a signature**
 - remark: sending only $s = E_{\text{pub}}(x)$ without the message x provides confidentiality but not authentication
- sign a message x using the (my) private key: $(x, s = E_{\text{priv}}(x))$
 - only the owner of the private key can generate the signature s
 - **problem: this scheme is still affected by existential forgery (see next slides)**
 - remark: sending only $s = E_{\text{priv}}(x)$ provides authentication but no confidentiality

Signing using private key

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

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

$$s = \text{sig}_{k_{\text{priv}}}(x) = x^d \bmod n$$

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

3. **To verify the signature:** “decrypt” the signature with the public key

$$x' = \text{ver}_{k_{\text{pub}}}(s) = s^e \bmod n$$

If $x=x'$, the signature is valid

Signing using private key (2)

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

Alice

Bob

K_{pub}

$K_{\text{pr}} = d$
 $K_{\text{pub}} = (n, e)$

(x, s)

Signing:

$s = \text{sig}_{k_{\text{pr}}}(x) \equiv x^d \bmod n$

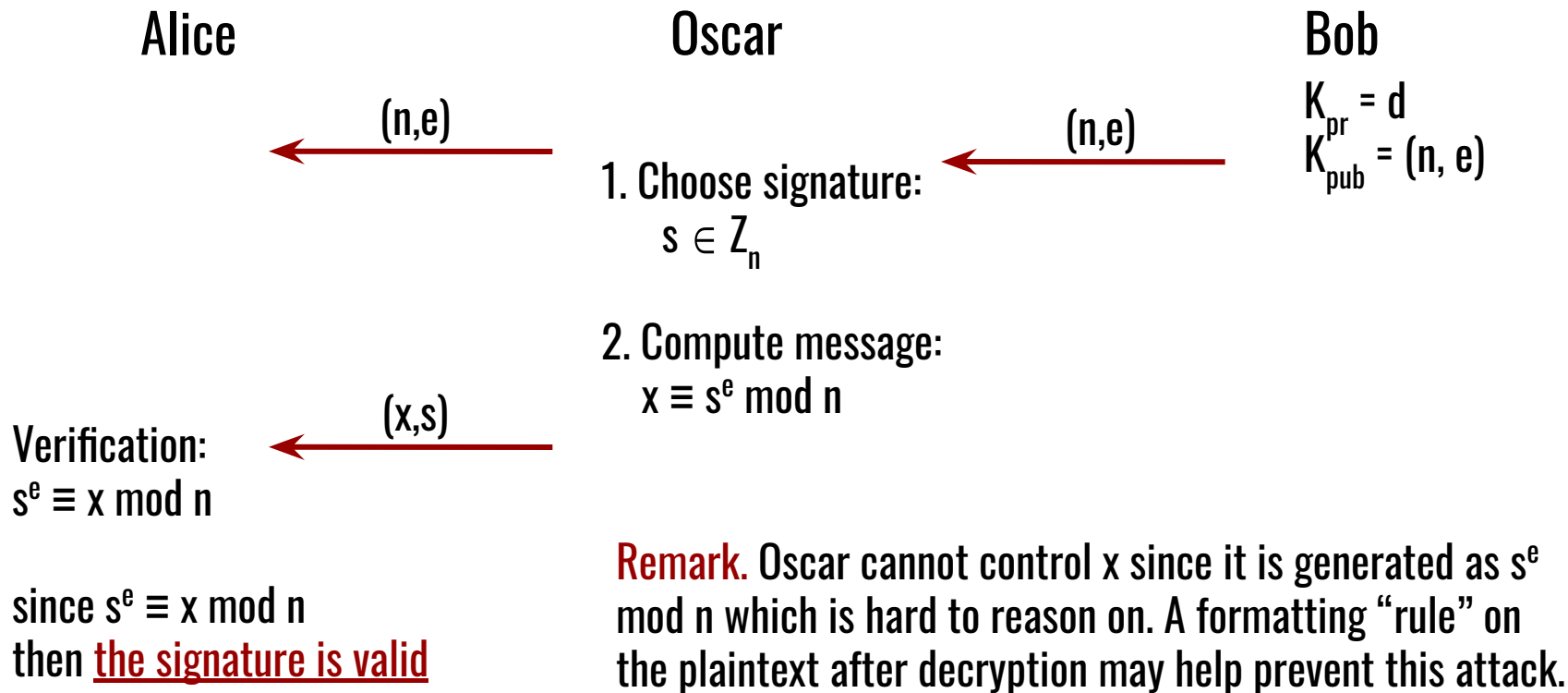
Verify signature:

$x' = s^e \bmod n$

If $x' =? x \bmod n \rightarrow$ valid signature

If $x' \neq x \bmod n \rightarrow$ invalid signature

Signing using private key: existential forgery



Other issues with PK scheme: ciphertext malleability

Several PK schemes, such as RSA, are malleable, i.e., an attacker can transform a ciphertext into another ciphertext which decrypts to a related plaintext without knowing the private key. For instance in RSA “textbook”, an attacker can exploit the property that:

$$(x_1 \cdot x_2)^e \bmod n \equiv x_1^e \cdot x_2^e \bmod n$$

Similar problem in Elgamal (see slides on asymmetric encryption).

As for encryption, textbook implementations of PK scheme are not safe for digital signature mechanism. Always use standards that define proper rules (e.g., padding).

Digital Signature

In practice, popular digital signature scheme are based on:

- **PKCS#1**: using RSA to sign hash of the message, padding with “formatting” rules.
- **Elgamal Signature Scheme**
- **Digital Signature Standard (DSS)**

PKCS#1 Digital Signature

This standard defines two schemes for digital signatures:

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**

PKCS#1 Digital Signature “with Appendix”

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

Signing the hash instead of the message provides several benefits:

1. **Efficiency**: RSA is slow, hence faster to “encrypt/decrypt” only a smaller amount of data
2. **Message overhead**: signing the message gives $2 \cdot |x|$ overhead, while signing hash gives constant overhead.

EMSA-PSS uses a MGF

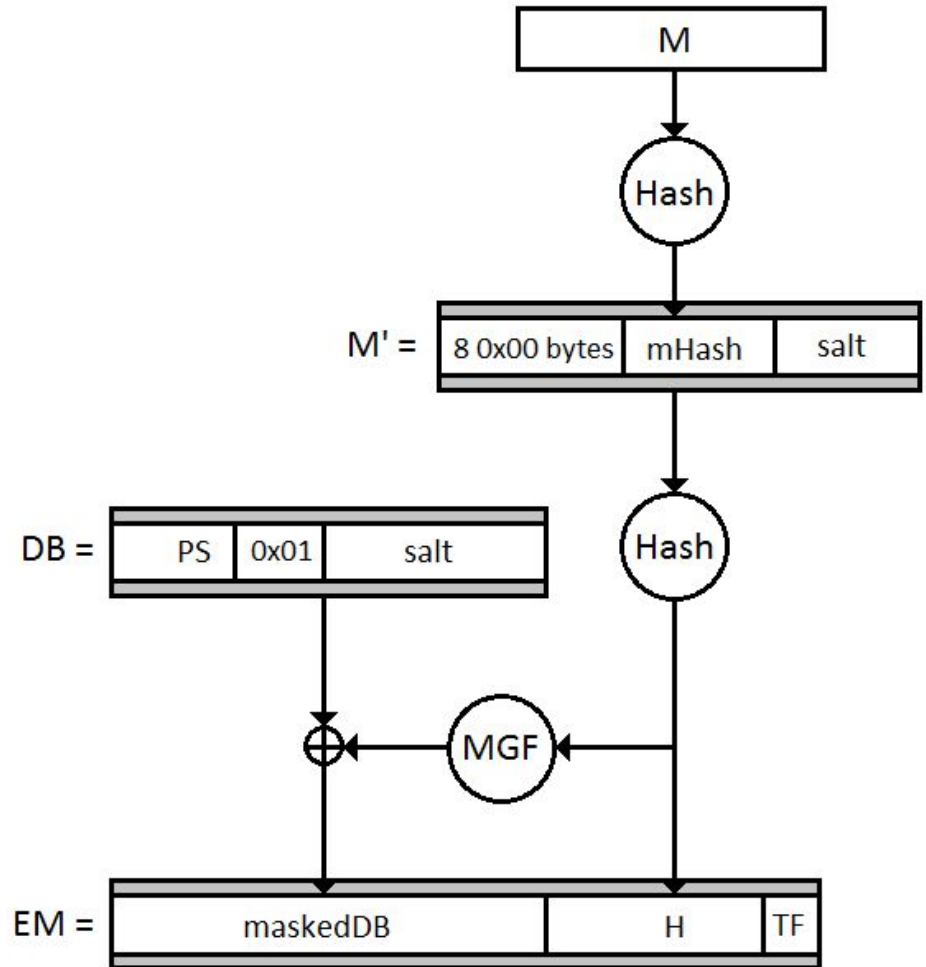
A **Mask Generation Function (MGF)** is a cryptographic primitive similar to a cryptographic hash function except that while a hash function's output is a fixed size, a MGF supports output of a variable length.

The most common mechanism to build a MGF is to iteratively apply a hash function together with an incrementing counter value, e.g., `hash(seed || counter)`. The counter may be incremented indefinitely to yield new output blocks until a sufficient amount of output is collected. This is the approach used in [MGF1](#).

RSASSA-PSS

EMSA-PSS encoding:

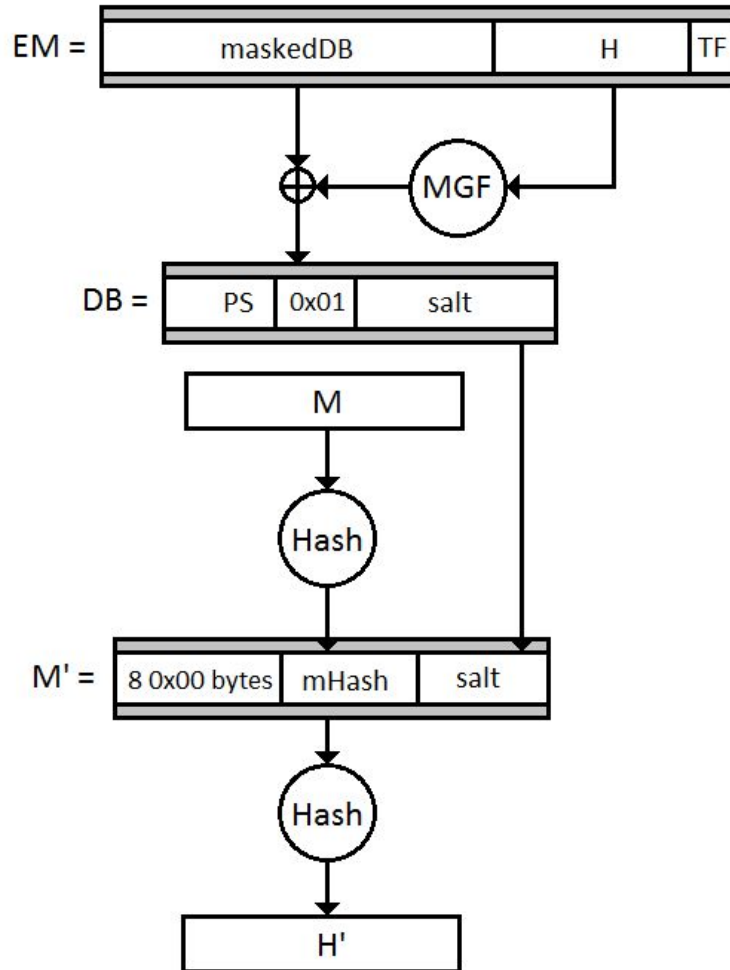
- M is the message
- salt is used as a randomizer
- PS is padding,
TF=0xBC
- Mask Generation Function (MGF)
- After encoding, RSA encryption



RSASSA-PSS (2)

EMSA-PSS verification:

- decrypt using RSA to get encoding
- given EM using the Mask Generation Function (MGF), it is possible to regenerate the salt thanks to XOR



RSASSA-PKCS1-v1_5

Message is padded as:

$$M' = 0x00 \parallel 0x01 \parallel \text{at least 8 } 0xFF \text{ bytes} \parallel 0x00 \parallel \text{hash specs} \parallel \text{hash}(M)$$

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

Elgamal Signature Scheme

Alice

Bob

choose prime p , generator α

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

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

(p, α, β)



ephemeral key:

$$k_e \in [2, p - 2]$$

such that

$$\gcd(k_e, p - 1) = 1$$

Hence, they are relatively
prime and multiplicative
inverse exists.

Elgamal Signature Scheme (2)

Alice

Bob

$$\text{signature: } \begin{cases} r \equiv \alpha^{k_e} \pmod{p} \\ s \equiv (x - d \cdot r)k_e^{-1} \pmod{p-1} \end{cases}$$

(x, s, r)



Verification:

$$t = \beta^r \cdot r^s \pmod{p}$$

check that:

$$t \stackrel{?}{=} \alpha^x \pmod{p}$$

Elgamal Signature Scheme (3)

This works because:

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

if want $t \equiv \alpha^x \pmod{p}$ then we need:

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

$$dr + k_e s \equiv x \pmod{p}$$

Given a corollary of the Fermat's Little Theorem:

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

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

Elgamal Signature Scheme (4)

we have to show that:

$$dr + k_e s \equiv x \pmod{p-1}$$

$$k_e s \equiv x - dr \pmod{p-1}$$

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

which exactly how s has been constructed.

Elgamal Signature Scheme: reuse of ephemeral key

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:

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

Elgamal Signature Scheme: reuse of ephemeral key (2)

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 Signature Scheme: 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:


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

Elgamal Signature Scheme: existential forgery attack (2)

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.

Elgamal Signature Scheme: existential forgery attack (3)

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

- Federal US Government standard for digital signatures (DSS): FIPS PUB 186
- Proposed by the National Institute of Standards and Technology (NIST)
- It defines a **Digital Signature Algorithm (DSA)**, which is based on the Elgamal signature scheme and SHA-1
- Signature is only 320 bits long
- Signing faster in DSA than in RSA, verification slower in DSA vs RSA

The Digital Signature Algorithm (DSA)

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 $\text{ord}(\alpha) = q$
4. Choose a random integer d with $0 < d < q$
5. Compute $\beta \equiv \alpha^d \pmod{p}$

The keys are:

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

$$k_{\text{pr}} = (d)$$

The Digital Signature Algorithm (DSA)

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 $r \equiv (\alpha^{k_E} \bmod p) \bmod q$
3. Compute $s \equiv (\text{SHA}(x) + d \cdot r) k_E^{-1} \bmod q$

The signature consists of (r, s)

SHA denotes the hash function SHA-1 which computes a 160-bit fingerprint of message x . Other hash functions can be used in place of SHA-1.

The Digital Signature Algorithm (DSA)

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 $u_1 \equiv w \cdot \text{SHA}(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q$

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

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

Proof of DSA

We need to show that the signature (r,s) in fact satisfied the condition $r \equiv v \bmod q$:

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

$$\Leftrightarrow k_E \equiv s^{-1} \text{SHA}(x) + d \cdot s^{-1} r \bmod q$$

$$\Leftrightarrow k_E \equiv u_1 + d \cdot u_2 \bmod q$$

We can raise α to either side of the equation if we reduce modulo p :

$$\Leftrightarrow \alpha^{k_E} \bmod p \equiv \alpha^{u_1 + d \cdot u_2} \bmod p$$

Since $\beta \equiv \alpha^d \bmod p$ we can write:

$$\Leftrightarrow \alpha^{k_E} \bmod p \equiv \alpha^{u_1} \beta^{u_2} \bmod p$$


We now reduce both sides of the equation modulo q :

$$\Leftrightarrow (\alpha^{k_E} \bmod p) \bmod q \equiv (\alpha^{u_1} \beta^{u_2} \bmod p) \bmod q$$

Since $r \equiv \alpha^{k_E} \bmod p \bmod q$ and $v \equiv (\alpha^{u_1} \beta^{u_2} \bmod p) \bmod q$, this expression is identical to: $r \equiv v$

Example of DSA

Alice

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


Verify:

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

$$u_1 \equiv 6 \cdot 26 \equiv 11 \pmod{29}$$

$$u_2 \equiv 6 \cdot 20 \equiv 4 \pmod{29}$$

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

$$v \equiv r \pmod{29} \rightarrow \text{valid signature}$$

Bob

Key generation:

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

Sign:

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

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

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.

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. The concept of multisignature systems were by no means created specifically for cryptocurrencies and have actually been around thousands of years:

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.

Common applications of digital signatures

1. **Code signing:** software vendor can easily publish a new release on a website (even from a third-party, e.g., an application store). Users can easily verify that the release has not been modified by an attacker, the code has been written by the software vendor, and the software vendor is only entity that can generate the code. Notice that:
 - a. cryptographic hash functions are not enough: anybody can generate the hash of something. They could be useful assuming the availability of “small read-only public space” from the software vendor, which however cannot be guaranteed (without using other solutions).
 - b. Message Authentication Code would require to establish a secret key between the user and the software vendor.

Common applications of digital signatures (2)

2. **Verifying the public key of an entity**: 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). For instance, in Gmail, given a message > show original: you can find the X-Google-DKIM-Signature field. The PK is available on a special DNS record.

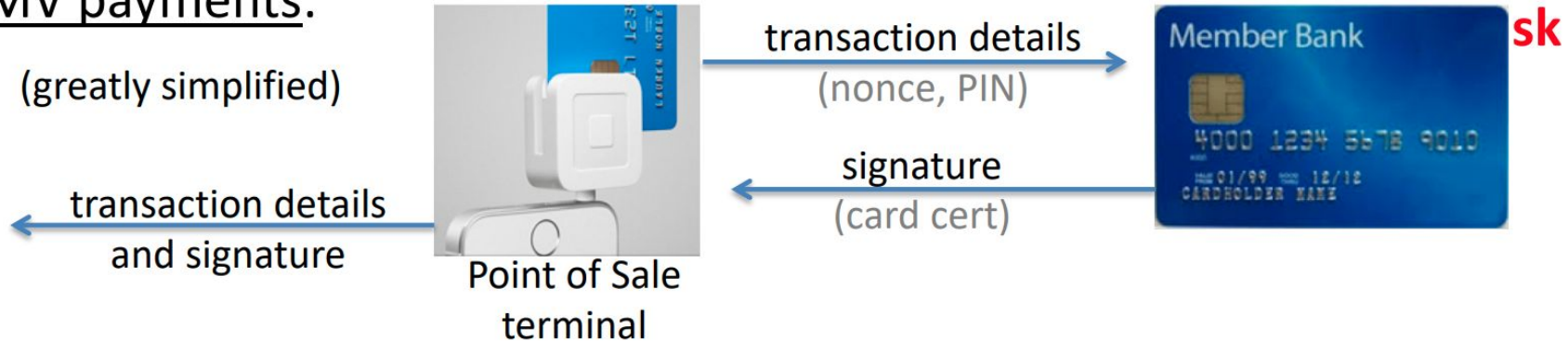
Remark. This is different from the “Italian PEC” (Posta Elettronica Certificata). DKIM is designed to authenticate the server that is sending an email, i.e., the mail infrastructure, and not the actual person that has written the message.

Common applications of digital signatures (3)

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

EMV payments:

(greatly simplified)



(credits)

Credits

These slides are based on material from:

- Slides of Prof. D'Amore from CNS 2019-2020
- Christof Paar and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. <http://www.crypto-textbook.com/>
- Wikipedia (english version)