Cryptographic Signature Scheme

Elena Kirshanova

Course “Information and Network Security”
Lecture 9
4 мая 2020 г.

Motivation

Key Exchange protocol + Symmetric Encryption = confidentiality

But

- as described Diffie-Helleman key exchange is vulnerable to active attacks
- it does not offer integrity of the communication
- nor does it offer authenticity

We'd like to achieve integrity, authenticity, as in MACs, in public key setting.

We can do it with cryptographic signature schemes

Signature scheme: definition

A Signature Scheme consists of three efficient algorithms

- Key generation: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  vk – verification key (public), sk – signing key (secret)
- Signature generation: $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk})$
- Verification: $\mathsf{Ver}(m, \sigma, \mathsf{vk})$ outputs $\{\mathsf{accept}, \mathsf{reject}\}$.

Here, $m \in \mathcal{M}$ is a message to be signed.

Correcntess: $\forall m, \forall (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}()$:

$$\mathsf{Ver}(m, \mathsf{Sign}(m, \mathsf{sk}), \mathsf{vk}) = \mathsf{accept}$$

Signature scheme: security

Two types of chosen message attacks:

I. Existential forgery

- the attacker can request the signature on any message of his choice
- he should not be able to output a valid message-signature pair (m, σ) for a new m (i.e., he did not previously request a signature for m

II. Strong Existential forgery

- the attacker can request the signature on any message of his choice
- he should not be able to output a valid signature on a previously signed message, i.e., (m, σ') is a valid attack even if the adversary saw (m, σ).

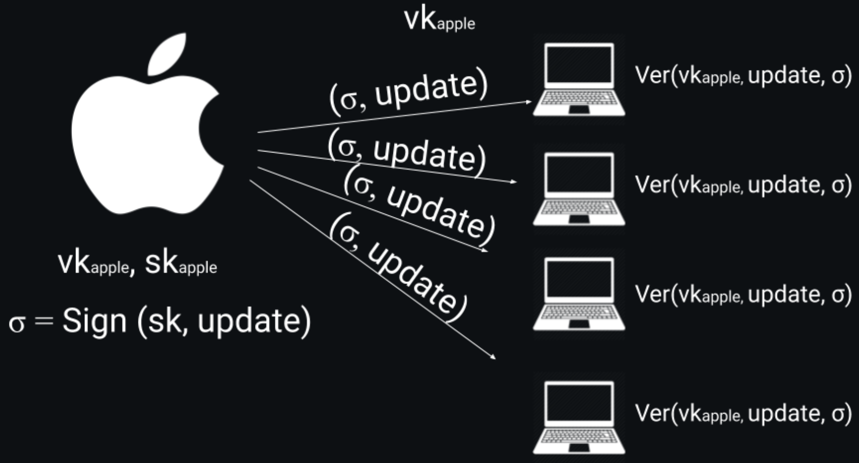
A signature that is secure in the first (weaker) model can be turned into a strongly secure signature.

Security caveats

- Non-repudiation (неотказ от авторства).
  – The signer be bound to messages she signs.
  – In the definition we gave, this property is not required and may not be useful: the signer could claim his vk to be stolen or leaked

- Duplicate Signature Key Selection (DSKS).
  – If an attacker, who sees (m, σ), can generate a key pair $(\mathsf{vk}', \mathsf{sk}')$ s.t. (m, σ) is also valid with respect to $(\mathsf{vk}', \mathsf{sk}')$.
  – ensures that the attacker cannot modify the signature. (e.g., the attacker cannot re-randomize a valid signature)
  – To prevent such attacks: the signer attaches her public key to the message before signing it.

Real life use-cases: software updates



Practical signature schemes

1. RSA
   – hardness is based on factoring
   – fast Ver, slow Sign, KeyGen, much larger keys, signatures

2. (EC)DSA= (Elliptic Curve) Digital Signature Algorithm
   – hardness is based on dlog
   – slower Ver, fast Sign, KeyGen
   – for ECDSA much shorter keys, signatures

3. ГОСТ Р 34.10-2012
   – same as ECDSA
   – old ГОСТ Р 34.10-94 was the same as DSA

Key sizes (in bits):

Security lvl.	ECDSA / ГОСТ'12	RSA/DSA
80	160	1024
128	256	3072
256	512	15360

Math crash course I: arithmetic in a ring

Let $N = p \cdot q$, where p, q are large primes

- $\mathbb{Z}_N = \{0, 1, \ldots, n-1\}$ – ring
- elements in $\mathbb{Z}_N$ are added and multiplied modulo N, i.e., for $x, y \in \mathbb{Z}_N$ Ex.: $N = 15$

$$11 + 6 \bmod N = \text{rem}(17, 15) = 2$$
$$6 \cdot 7 \bmod N = \text{rem}(42, 15) = 12$$

Math crash course I: arithmetic in a ring

Let $N = p \cdot q$, where p, q are large primes

- $\mathbb{Z}_N = \{0, 1, \ldots, n-1\}$ – ring
- elements in $\mathbb{Z}_N$ are added and multiplied modulo N, i.e., for $x, y \in \mathbb{Z}_N$ Ex.: $N = 15$

$$11 + 6 \bmod N = \mathrm{rem}(17, 15) = 2$$
$$6 \cdot 7 \bmod N = \mathrm{rem}(42, 15) = 12$$

- Not every non-zero $x \in \mathbb{Z}_N$ has inverse! The set of invertible elements is denoted $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N \mid \gcd(x, N) == 1\}$.
  $\gcd$ – greatest common divisor (НОД).

  Ex.: $3, 6, 9, 5, 10, 12 \notin \mathbb{Z}_N^*$.
  $\mathbb{Z}_N^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

Math crash course I: structure of $\mathbb{Z}_N^*$

- Denote $\phi(N) = |\mathbb{Z}_N^*|$. $\phi(N)$ is known as Euler function
  – if N – prime, $\phi(N) = N - 1$ (see prev. lecture)
  – if $N = p_1^{e_1} \cdot p_n^{e_n}$, $\phi(N) = N \cdot \prod_i \left(1 - \frac{1}{p_i}\right)$.
  – for $N = p \cdot q$, $\phi(N) = (p-1)(q-1)$.
  Ex.: $|\mathbb{Z}_N^*| = |\{1, 2, 4, 7, 8, 11, 13, 14\}| = 2 \cdot 4 = 8$.

- Euler's theorem: for all $a \in \mathbb{Z}_N^*$

$$a^{\phi(n)} = 1 \bmod N$$

recall Fermat's: $a^{p-1} = 1 \bmod p$ for p-prime.

Easy and hard problems in $\mathbb{Z}_N$

$N = p \cdot q$, p, q are of $\approx$ 1024 bits each.

In $\mathbb{Z}_N$ it is easy to

– add, multiply, find inverse (if exists, or check if does not)
– compute $g^r \bmod N$

It is believed to be hard to

– find p, q
– compute square roots in $\mathbb{Z}_N$ (as hard as factoring)
– compute e^{th} roots module N when $\gcd(e, \phi(N)) = 1$

RSA Key Generation

Let $\ell > 2$ be an integer and $e > 2$ be an odd integer.

RSAGen(ℓ, e) :

1. Generate an ℓ-bit integer p s.t. $\gcd(p-1, e) = 1$
2. Generate an ℓ-bit integer $q \neq p$ s.t. $\gcd(q-1, e) = 1$
3. $N = p \cdot q$, $\phi(N) = (p-1)(q-1)$
4. $d = e^{-1} \bmod \phi(N)$
5. Output $\mathsf{vk} = (N, e), \mathsf{sk} = (N, d)$

- there exist efficient probabilistic algorithms to generate primes
- Step 4 is correct since $d \in \mathbb{Z}_N^*$ since

$$\gcd(p-1, e) = \gcd(q-1, e) = 1 \implies \gcd((p-1)(q-1), e) = 1.$$

- there is plenty of conditions on p, q to make the above secure

Do not try to implement RSAGen yourself.

RSA Signature Generation and Verification

$\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_N^*$– a cryptographic hash-function

I. RSASign(sk $= (N, d), m)$:

1. $y = \mathcal{H}(m) \in \mathbb{Z}_N^*$
2. $\sigma = y^d \bmod N$

II.
RSAVerify(vk $= (N, e), m, \sigma)$:

1. $y' = \sigma^e \bmod N$
2. $\texttt{return}(y' == \mathcal{H}(m))$

$$(y^d)^e = y^{ed} = y \bmod N$$

Correctness: For $N = pq$ and e, d s.t. $ed = 1 \bmod \phi(N)$ and for all $x \in \mathbb{Z}$

$$x^{ed} = x \bmod N$$

Proof: for $k \in \mathbb{Z}$

$ed = 1 + k\phi(N) = 1 + k(p-1)(q-1)$

$x^{p-1} = x \bmod p$ (Ferma't thm.)

$x^{ed} = x^{1+k(p-1)(q-1)} =$

$x \cdot (x^{p-1})^{q-1} = x \bmod p$

Analogously, $x^{ed} = x \bmod q$

$\implies p, q \mid x^{ed} - x$

$\implies x^{ed} = x \bmod p \cdot q$

Without $\mathcal{H}$ the scheme is trivially insecure!

RSA security

- $\text{RSAGen}(\ell, e) \to (\text{vk} = (N, e), \text{sk} = (N, d))$
- $\text{RSASign}(\text{sk}, m) \to \sigma = \mathcal{H}(m)^e \bmod N$
- $\text{RSAVerify}(\text{vk}, m, \sigma) \to \{0, 1\}$

RSA Assumption: There does not exist a ppt adversary that given (N, m, m^e) for a random $m \in \mathbb{Z}_N^*$, outputs m.

Factoring $N \implies$ computing e^{th}–roots.
The inverse is not known!

Theorem: The signature scheme (RSAGen, RSASign, RSAVerify) is secure in exsistensial forgery CMA model under the RSA Assumption and the assumption that $\mathcal{H}$ is a Random Oracle.

Informally, a Random Oracle model is an heuristic way to say that $\mathcal{H}$ behaves like a black-box that replies with random (but consistent) outputs.

Standards

Variants of RSA Signatures are standardized at PKCS (Public Key Cryptography Standards) by RSA Security LLC
`https://en.wikipedia.org/wiki/PKCS`

Version 2.2 (latest) of PKCS #1 includes

- RSASSA-PSS
  SSA = Signature Scheme with Appendix
  PSS = Probabilistic Signature Scheme
- RSASSA-PKCS1-v1_5 (attacks exist)

The standards describe data type conversions, how to represent the data, hash functions, etc.

Usages of signatures schemes

Practical signature schemes

1. RSA
   long keys, signatures; fast verification

2. ECDSA, GOST
   short keys, signatures; slower verification

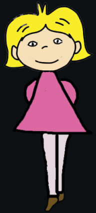
RSA is good for Certificates, ECDSA/GOST are good for e-mails.

Certificates bind a public key to an identity.

Certificates and PKI

Alice

$\mathsf{pk}_{\mathtt{A}}$



Certificate Signing Request $\longrightarrow$

`id, email,` $\mathsf{pk}_{\mathtt{A}}$

$\longleftarrow$ `cert`, $\sigma = \mathsf{Sign}(\mathsf{sk}_{\mathtt{CA}}, \mathtt{cert})$

Certificate Authority (CA)

$\mathsf{vk}_{\mathtt{CA}}, \mathsf{sk}_{\mathtt{CA}}$

Certifying Alice's identity
Creating a certificate **`cert`**
Signing **`cert`**

Anyone, who needs to communicate securely with Alice, first runs $\mathsf{Ver}(\mathsf{vk}_{\mathtt{CA}}\mathtt{cert}, \sigma)$. If verification passes, $\mathsf{pk}_{\mathtt{A}}$ can be used to communicate with Alice.

Example: X.509 certificate

Certificate chains

Root CAs	$\longrightarrow$	Intermediate CA
certifies Interm. CAs		certifies clients

There are currently thousands of intermediate CAs operating on the Internet

To avoid malicious CAs: certificate pinning:

1. Every browser maintains a pinning database: $(\mathtt{domain}, \mathtt{hash}_0, \mathtt{hash}_1, \ldots)$
2. The data for each record is provided by the domain owner
3. When the browser connects to a domain, domain sends its certificate chain $\mathtt{cert}_0, \mathtt{cert}_1, \ldots$
4. The browser computes $\mathcal{H}(\mathtt{cert}_i)$ and verifies against $\mathtt{hash}_i$.

The last PA

Task: implement KeyGen, Sign, Ver for RSA and ECDSA.

Compare the speed of Sign, Ver for 1000 messages (standard comparison)