Digital signature

2021-05-11

1 Digital signature

A *digital signature* is a piece of information (i.e. a sequence of bits) attached to a message which provides the three following properties:

- **Message authentication** - the receiver of the message can verify the origin of the message
- **Message integrity** - if the message gets modified the receiver is able to detect it
- **Non repudiation** - the signer cannot later claim that he or she didn't sign it[1].

1.1 Digital signature scheme

A digital signature scheme consists of the following three probabilistic algorithms:

- `Gen(n)` - generate a symmetric key pair `(pk, sk)` of `n` bits, where `n` is a security parameter.
- $\texttt{Sign}_{\texttt{sk}}(\texttt{m})$ - generate a digital signature σ from the *secret key* `sk` and the *message* `m`.
- $\texttt{Vrfy}_{\texttt{pk}}(\texttt{m},\sigma)$ - input `pk`, `m` and σ, output 1 if the signature is valid, 0 if the signature is invalid.

It's important to notice that while the key `pk` is public and available to everyone, the secret key `sk` must be kept, indeed, secret.

1.1.1 Security of the digital signature scheme

A digital signature scheme `(Gen, Sign, Vrfy)` is *secure* if an adversary knowing `pk` and other valid signatures $(m_1, \sigma_1), (m_2, \sigma_2), \dots$ is not able to produce a new message m and a valid signature σ for it.

[1]Actually, to gain non-repudiation we also need a public key certificate, but we won't explore technical details in this paper, and we'll suppose that the key used to sign is unequivocally linked to the signer.

Exercise 10.2.1 What is the difference between a MAC and a digital signature?

- MAC guarantees only authentication and integrity, while digital signature (in principle) also guarantees non-repudiation.
- A digital signature is created with a key pair `(pk, sk)`, while the MAC is based on a secret key, shared between the sender and the receiver.

1.2 Digital signature protocol

Let's consider a sender and a receiver (Alice and Bob).

Alice wants to send a message m to Bob by using her secret key $\texttt{sk}_\texttt{A}$.

Bob knows Alice's public key (which is indeed public) and is able to verify the signature.

$$\begin{tikzcd} \text{Alice} & & \text{Bob} \\ \sigma = \texttt{Sign}_{sk_A}(m) \arrow[rr, "(m, \sigma)"] & & \text{check } \texttt{Vrfy}_{pk_A}(m, \sigma) \end{tikzcd}$$

Properties:

- The signature is *authentic* - Bob knows that Alice signed the message.
- The signature is *unforgeable* - only Alice knows her private key.
- The signature is *not reusable* for any other message - because it's a function of the message.
- Any *alteration* of the message would invalid the signature - it won't be possible to verify the signature with Alice's public key anymore.
- The signature cannot be *repudiated* - Alice cannot claim not having signed the message because she was the only one knowing her private key.

1.3 Digital signature and timestamp

Digital signatures should also include timestamps (attach a timestamp to the message and sign the whole document).

Let's consider the following *example*, taken from the Bruce Schneier's book "Applied Cryptography":

> Alice sends Bob a signed digital check for \$100. Bob takes the check to the bank, which verifies the signature and moves the money from one account to another. The following week, Bob takes the same check to the bank, which again verifies the signature and moves the money from one account to another. And so on.

But if date and time of signature are attached to the message, then the bank could store this timestamp into a database, and when Bob takes the check for the second time, the bank checks the timestamp against its database.

1.4 RSA digital signature

1.4.1 Naive RSA-signature

Alice's public key $\mathtt{pk}_A = (n, e)$ and secret key is $\mathtt{sk}_A = d$.

$$\begin{array}{ccc}
\text{Alice} & & \text{Bob} \\
\sigma = \mathtt{Sign}_{sk_A}(m) = m^d (mod\ n) & & \\
 & \xrightarrow{(m, \sigma)} & \\
 & & \mathtt{Vrfy}_{pk_A}(m, \sigma) = 1 \text{ if } \sigma^e = m (mod\ n)
\end{array}$$

Exercise 10.2.2 Show that the above signature scheme is not secure. *Hint*: choose any $\sigma \in \mathbb{Z}_n$ and consider $m = \sigma^e (mod\ n)$

First, let's remember what does it mean for a digital signature scheme to be secure: an adversary should not be able to produce a new message m and a valid signature σ for it.

Now, let's pick $\sigma \in \mathbb{Z}_n^*$. We define $m := \sigma^e (mod\ n)$. Then it is true that:

$$m = \sigma^e (mod\ n) \implies \sigma^e = m (mod\ n)$$

But this means that, by definition of our scheme, $\mathtt{Vrfy}_{\mathtt{pk}_A}(m, \sigma) = 1$, and therefore σ is a valid signature for m.

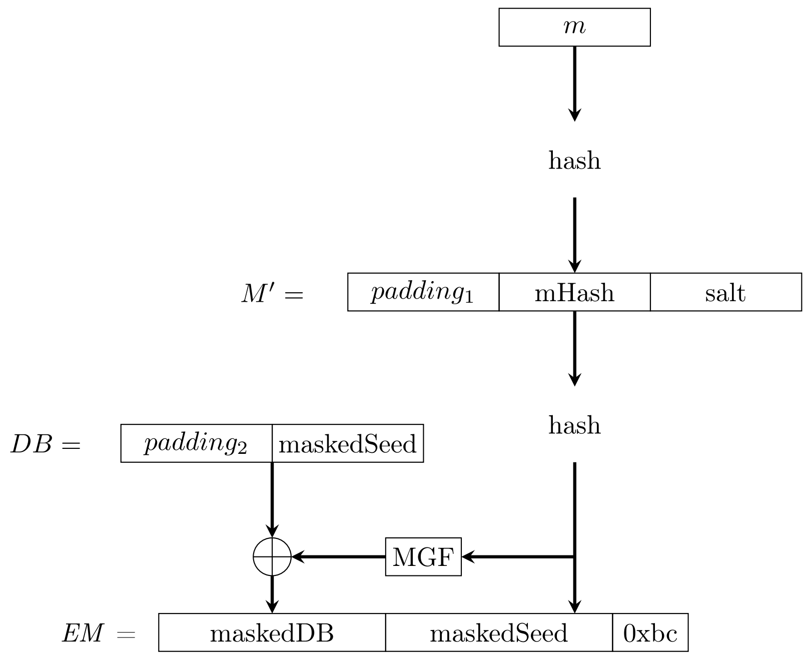
Now, some might argue that this isn't a realistic attack because the adversary has not control over m. Actually this is irrelevant to our purposes: plain RSA signature doesn't respect digital signature security requisites.

1.5 EMSA-PSS - RSA digital signature

Encoding method for signature with appendix probabilistic signature scheme. (see also `https://tools.ietf.org/rfc/rfc3447.txt`).

The following method is parametrized by the choice of:

- Hash function
- Mask generation function
- Salt length



Let's first define fundamental component:

- `emLen` = length of the encoded message (EM) (at least 8·`hLen`+8·`sLen`+9).
- `hLen` = length of `mHash` = `hash`(m).
- `sLen` = length of `salt`.
- $padding_1$ = `0x0000000000000000`
- $padding_2$ = `PS || 0x01` where `PS` is an octet string consisting of `emLen - sLen - hLen - 2` zero octets (bytes).

Encoding:

1. If the length of m is greater than the input limitation for the hash function ($2^{61} - 1$ octets for SHA-1), output "message too long" and stop.
2. Let `mHash = Hash(M)`, an octet string of length `hLen`.
3. If `emLen < hLen + sLen + 2`, output "encoding error" and stop.
4. Generate a random octet string salt of length `sLen`; if `sLen = 0`, then `salt` is the empty string.
5. Let `M'` = `0x 00 00 00 00 00 00 00 00` || `mHash` || `salt`. `M'` is an octet string of length `8 + hLen + sLen` with eight initial zero octets.

6. Let `H = Hash(M')`, an octet string of length `hLen`.
7. Generate an octet string `PS` consisting of`emLen - sLen - hLen - 2` zero octets. The length of `PS` may be 0.
8. Let `DB = PS || 0x01 || salt`; `DB` is an octet string of length `emLen - hLen - 1`.
9. Let `dbMask = MGF(H, emLen - hLen - 1)`.
10. Let `maskedDB` $=$ `DB` $\oplus$ `dbMask`.
11. Set the leftmost `8emLen - emBits` bits of the leftmost octet in `maskedDB` to zero.
12. Let `EM = maskedDB || H || 0xbc`.
13. Output `EM`.

Verification operation follows reverse steps to recover `salt`, then forward steps to recompute and compare `H`.

Exercise 10.2.4 Find $padding_2$ in the standard.

1.6 Digital signature algorithm

In 1991, The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS).

We consider a bit length of 1024, though longer bit lenghs are possible in the standard.

1.6.1 Key generation

1. Generate a prime number 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 element α with $ord(\alpha) = q$, i.e. α generates the subgroup with q elements[2]
4. Choose a random integer d with $0 < d < q$.
5. Compute $\beta \equiv \alpha^d (mod\ p)$
6. The keys are $k_{pub} = (p, q, \alpha, \beta)$ and $k_{pri} = d$.

[2]Remember that a group is a set of elements with a binary operation defined on it, and the properties of closure (i.e. the operation is closed), associativity, neutral element, inverse element, while the *order* of an element α of the group is the smallest positive integer k such that $\alpha^k = 1$, where 1 is the neutral element of the group.

Generating large prime numbers How do we generate a prime number p such that $2^{1023} < p < 2^{1024}$? We can observe that 2^{1023} is just 1 followed by 1023 zeros, i.e. 1000...0. To pick a random number between $2^{1023} and 2^{1024}$ we can just set to 1 some random zero-bits: the resulting number will be in the given range.

1.6.2 Signature

1. Choose an integer as random *ephemeral*[3] key k_E with $0 < k_E < q$.
2. Compute $r \equiv (\alpha^{k_E} (mod\ p))(mod\ q)$
3. Compute $s \equiv (SHA(x) + d \cdot r)k_E^{-1}(mod\ q)$

Where x is the message to be signed.

1.6.3 Verification

1. Compute auxiliary value $w \equiv s^{-1}(mod\ q)$
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x)(mod\ q)$
3. Compute auxiliary valid $u_2 \equiv w \cdot r(mod\ q)$
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2}(mod\ p))(mod\ q)$
5. The verification $ver_{k_{pub}}(x, (r, s))$ follows from:

$$v \begin{cases} \equiv r(mod\ q) & \Rightarrow \text{valid signature} \\ \not\equiv r(mod\ q) & \Rightarrow \text{invalid signature} \end{cases}$$

I.e. the verifier accepts a signature (r, s) only if $v \equiv r(mod\ q)$ is verified.

Exercise 10.3.6 Set $p = 59, q = 29, \alpha = 3, d = 7, \beta = \alpha^d(mod\ 59)$. Assuming that $SHA(x) = 26$ compute the DSA signature (r, s).

1. We choose $k_E = 7$
2. We compute r as:
$$(\alpha^{k_E}(mod\ p))(mod\ q) = (3^7(mod\ 59))(mod\ 29) = 4$$
3. We compute s as:
$$(SHA(x) + d \cdot r) \cdot k_E^{-1}(mod\ q) = (26 + 7 \cdot 4) \cdot 7^{-1}(mod\ 29) = 7$$

So the digital signature is $(r, s) = (4, 7)$.

[3]An ephemeral key or *session* key is a key valid only for a limited period of time, usually the time of a connection between two hosts.

Exercise 10.3.7 If an adversary is able to predict k_E DSA is broken. Read the following article: `https://www.bbc.com/news/technology-12116051`.

Exercise 10.3.8 Consider a variant of DSA in which just messages in $\mathbb{Z}_q$ are signed and the Hash function is omitted, i.e. to get the signature (r, s) of $m \in \mathbb{Z}_q$ we compute $r = \alpha^{k_E}$ (as usual), but $s = (m + d \cdot r) \cdot k_E^{-1} (mod\ q)$. Is this secure?

A possible attack can be done by setting $k_E = d$. The attacker doesn't know the secret key d, but now it has to output $s = (m + d \cdot r)d^{-1}(mod\ q)$. He or she can just choose $m = 0$, so that:

$$s = (m + d \cdot r)d^{-1}(mod\ q) = (d \cdot r) \cdot d^{-1}(mod\ q) = r(mod\ q)$$

Now s can be computed, because it only depends on known parameters, and (r, s) with $r = s$ is a valid signature for m.

1.6.4 Prime generation for DSA

The following is a slightly different generator taken from "Understanding Cryptography" by Christof Paar. The goal here is to find two primes (p, q) where $2^{1023} < p < 2^{1024}$ and $2^{159} < q < 2^{160}$ and $p - 1$ is a multiple of q:

Initialization: Set i = 1.

1. Find prime q with $2^{159} < q < 2^{160}$ using the Miller-Rabin algorithm[4].
2. For i = 1 to 4096:
   - 2.1. Generate a random integer M: $2^{1023} < M < 2^{1024}$.
   - 2.2. $M_r \equiv M(mod\ 2q)$
   - 2.3. $p - 1 \equiv M - M_r$ (note that $p - 1$ is a multiple of $2q$)
   - 2.4. if p is prime return(p, q)
   - 2.5. i = i+1
3. Go back to step 1.

[4]Miller-Rabin algorithm is a primality test for numbers.