1 RSA-PSS

To define an electronic signature with RSA, you have to apply it in reverse. Take Alice's public key $K_e = (e, n)$ and d her secret exponent. Roughly, when Alice wants to sign a document M, she follows the steps:

- She computes a hash $h_M = H(M)$ (we suppose now that $0 \leq h_M < n$).
- She generates the signature $s_M = (h_M)^d \mod n$ (Alice is the only one to be able to generate this signature as she is the only one to own d).
- The signed document is thus the pair $(M||s_M)$, and this pair is sent to Bob.

1. In this case, how can Bob verify the signature ?

In practice the function H is more complicated than a hash function. It is rather a non-deterministic process like PSS. This latter scheme uses zero paddings (`padding1`, of exactly 8 bytes, and `padding2`), together with a randomly chosen salt. Then it uses a hashing extenion called a MGF (Mask Generating Function) which extend a hash on more than the number of bits of the classical output of the hash function. for instance if the output of the hash function is on 256 bits, the MGF can extend this fingerprint on 2048 bits to reach the number of bits of RSA. An easy way to acheive this is to append several times its input while re-hashing it until reaching more than the desired number of bits.
Precisely, PSS proceeds as shown below and on Figure 1:

i. $M' = (padding1||Hash(M)||salt)$;

ii. $H = Hash(M')$;

iii. $g' = (padding2||0x01||salt)$;

iv. $mask = g' \oplus MGF(H)$;

v. $h_M = (mask||H||0xbc)$.

2. After deciphering s_M, how can Bob recover the salt used by Alice ?
3. How can he then verify the signature ?
4. On Figure 1, what can you say about the different block sizes when the hash function outputs 256 bits, the chosen salt is on 512 bits and the cipher works on 2048 bits outputs ? What is the maximal salt size ?

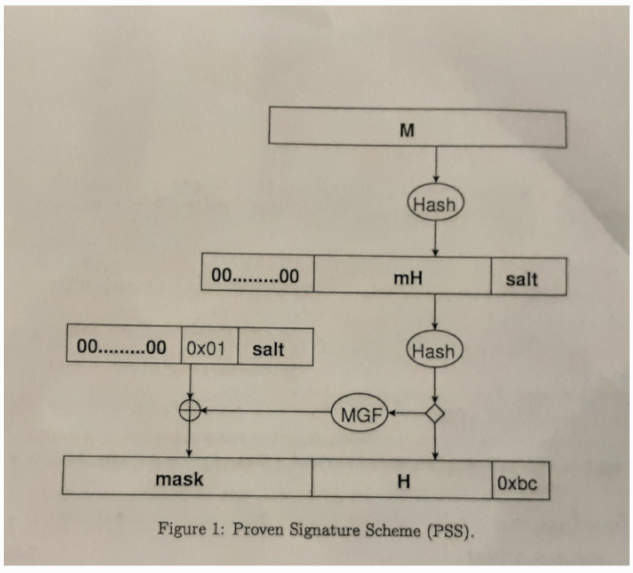


Figure 1: Proven Signature Scheme (PSS).

(1)

The signature is generated by : $M \rightarrow H \xrightarrow{H(m)} S \rightarrow S_{K_{PRIVA}}(H(m))$

K_{PRIVA}

so Bob can simply use the public key of Alice to decipher the S_m that he has received. Now he can hash the document and check if the result matches with what he has received from Alice.

(2)

He has to follow the same steps but in reverse order to get back to the salt. Specifically :

h_m = mask + H + 0xbc ⇝ he can isolate the mask

$mask = g' \oplus MGF(H)$

$g' = mask \oplus MGF(H)$ ⇝ he can apply MGF(H) and XOR it with the mask to obtain g'

$g' = (padding \,||\, 0x01 \,||\, salt)$

↑ The size of the padding is unknown but it is all 0x00, therefore it is easy to find it at thanks to 0x01

At this point he has isolated the salt value

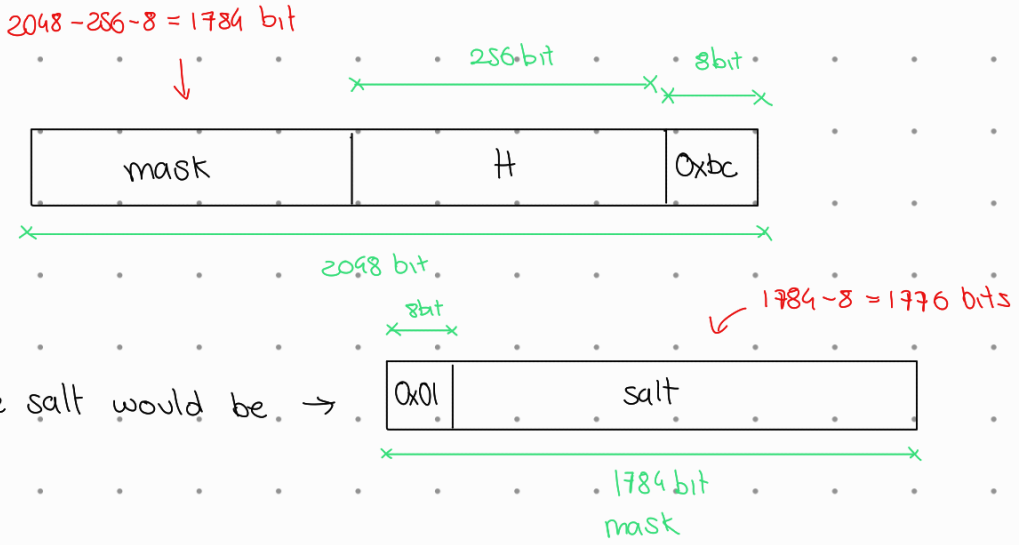
③

To verify the signature he can check if M' is correct.

$h_m = (mask \,||\, H \,||\, 0xbc)$ → $H = Hash(M')$ ⟷ ? ⟷ $H = Hash(M')$ ← calculate $M' = (padding1 \,||\, H(M) \,||\, salt)$

If they are equal the signature is valid

④



3 Signature numérique GOST

GOST is a russian signature standard. It uses the following parameters:

Partagés : q prime between 254 and 256 bits, p prime between 509 and 512 bits (or between 1020 and 1024 bits) such that q divides $p-1$, $a \neq 1$ such that $a^q \equiv 1[p]$ and a hash function h.

Privé : x.

Public : $y \equiv a^x[p]$.

To sign a message M, one has to:

i. Alice generates a random number k.

ii. Alice computes and sends $r = (a^k \mod p) \mod q$ and $s = (xr + kh(M)) \mod q$.

iii. Bob computes $v = h(M)^{q-2} \mod q$; $z_1 = sv \mod q$; $z_2 = ((q - r)v) \mod q$ and $u = ((a^{z_1} y^{z_2}) \mod p) \mod q$. if $u == r$ Then the signature is verified.

1. How do we build a ?
2. What is the size of the private key ?
3. What is the size of the random number k ?
4. Show that the verification is valid.
5. What happens when $r = 0$ and how to correct the protocol inthis case ?
6. What if $h(M) \equiv 0[q]$?

①

a has to be built such that $a^q \equiv 1 [p]$

②

The private key is at most the size of p

③

The size of the random number k is at most the one of q

④

$$g^{z_1} g^{z_2} = g^{sv} g^{xz_2} = g^{xrv+k} g^{xz_2} = g^{xrv+k} g^{xqv-xrv} = g^k \mod p$$

4 Secure Shell

The SSH software enables a secure telnet session between two hosts.
Upon a request from the client, the server sends his public key K_p in clear.
The client stores it in his memory (in the file `$HOME/.ssh/known_hosts` ; if already stored, the client compares it with the stored one and warns the user if it has changed).
The client picks a session key k and sends it to the server in an encrypted way.
The server decrypts the session key k. The client and the server can now communicate with a common secret key k with symmetric encryption.

1. Why do we want to secure the session with symmetric encryption instead of asymmetric encryption?
2. Assuming that all the messages in the protocol above are authenticated, explain why the subsequent connections are confidential and authenticated.

3. If the first connection is not authenticated, explain that an active adversary can impersonate the server.
4. Why does the client need to warn the user when the public key has changed?
5. Why is SSH useful?
6. It is possible to add a certificate in the file `$HOME/.ssh/authorized_keys` on a server (or directly a public key). Then all the accesses made by the client to the server will use a locally stored private key, encrypted by a symetric algorithm using a hashed password as a key. What is the main difference between this system and a direct passwd/login pair ?

(1)

The main reason is to have a fast communication. Using an asymmetric encryption schema would slow the communication down. Using symmetric key allows us to have confidentiality and speed.

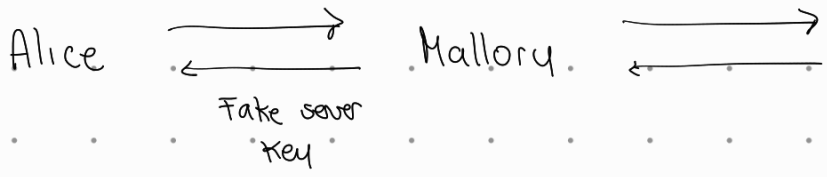
(2)

If the first message from the user to the server is authenticated and the server response is authenticated, the user can be sure about the server key and vice-versa. Subsequently all other connections are authenticated.

and confidential. Confidentiality depends on the key strength of the symmetric algorithm.

(3)

If the first connection is not authenticated a MITM may happen. This is because an attacker could impersonate a server by sending its key as sever key. However, the user should be prompted with a message if he was already connected to that sever before.



(4)

This is what was previously mentioned in answer 3. If the user is not prompted with any message the key could be overwritten without checking the signature.

(5)

Because it allows for secure and fast connection with other host and servers.

(7)

If we trust the key that we are adding we may avoid using password at all. This removes the possibility of intercepting the password or brute forcing it to gain access to a system.

5 Authenticated Diffie-Hellman Key Agreement

Let us consider a public-key Diffie-Hellman key agreement protocol derived from the simple Diffie-Hellman protocol. In this protocol, we have the following public parameters:

- a large prime p
- a large prime factor q of p -1
- an element g of order q in $\mathbb{Z}/p\mathbb{Z}^*$.

Each user U has a random secret key $X_U \in \mathbb{Z}/q\mathbb{Z}$, uniformly distributed and a public key $Y_U = g^{X_U} \mod p$. All the users' public keys are stored in an authenticated database (e.g., using a trusted third party), which is publicly readable. We propose the following key agreement protocol between users A and B:

- A generates $a \in \mathbb{Z}/q\mathbb{Z}$ using a pseudorandom number generator, computes $v = g^a \mod p$, and sends v to B.
- B generates $b \in \mathbb{Z}/q\mathbb{Z}$, using a pseudorandom number generator, computes $w = g^b \mod p$ and sends w to A.

In the end, A and B share the secret key $K = g^{aX_B + bX_A} \mod p$.

1. Explain how A can compute K.
2. Assume the pseudorandom number generator of B is biased in the sense that it only generates small numbers (eg., of length around 40 bits) instead of generating numbers almost uniformly in $\mathbb{Z}/q\mathbb{Z}$. Show how an adversary A* can impersonate A to set up a key with B. Suggest a countermeasure.
3. Assume that $b = ac$ for some small c. Show that the adversary A* can impersonate A and set up a key with B. Suggest a countermeasure.

(1)

It would be enough to generate $v = g^a \mod p$ and to receive $w = g^b \mod p$ from Bob. Then Alice has to retrive from the sever $Y_B = g^{X_B}$, we assume that she has $Y_A = g^{X_A}$

$$K = g^{aX_B + bX_A} = g^{aX_B} \cdot g^{bX_A} = (g^a)^{X_B} \cdot (g^b)^{X_A} =$$

$$= v^{X_B} \; w^{X_A}$$

N^{X_B} can be computed by sending N to Bob and asking for N^{X_B}