31/10/2017

DIGITAL SIGNATURES

According to the *Wikipedia* definition we have that: a **digital signature** is a mathematical scheme for demonstrating the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity). In few words, a digital signature provides *authentication, integrity* and *non-repudiation*. We already studied an approach for *data integrity*, it was based on MACs. But MACs relies on a shared secret key. In the case Alice and Bob share a secret key then (K, MAC_K(M)) convinces Bob that the sender is indeed Alice, but in case of a dispute, how they can show it to a third party (who has not the key)? In particular, since Alice and Bob share the private key, Bob could generate that message himself, so how can a third party check whether the message was originated by Alice or Bob? Impossible. This is just a **limit** of MAC tag based on shared private key.
We need some extra ingredient, some extra effort.
Then we discussed **RSA** and in general about the usage of ***public and private keys***:

- Encryption with other party's public key → you get *confidentiality* (he decrypts with his private key)
- Encryption with my private key → *no confidentiality, but authentication* (everyone is enabled to decrypt my ciphertext, it is just sufficient to use my public key in order to do it, but in this way other parties can be sure about the fact that it was me who encrypted the message)

Now I'm going to show an approach that does not work (just for teaching purposes) in order to show you why we need an extra ingredient. Lets' just assume the digital signature is made in the following way: assume this is the message M and Bob is the owner of a private key K_s and a public key K_p. The signature is by definition the encryption made using the private key K_p of Bob that gives as outcome a ciphertext C. This is not the real definition, pay attention! Bob can send the pair (M,C) to Alice. Alice takes the signature (C) and then she can check if the decryption is the same as M (she just decrypts C with the K_p of Bob, as the Public Key Cryptography offers by construction). Also in this case we are introducing a model in where we are sending a pair of files and the receiver runs an algorithm that answers whether rejecting or accepting. If the attacker changes even only a bit of C, Alice can never decrypt in order to get the original message. Here is the wrong definition of digital signature:

$$Sig(M) = E_{Ks}(M) = C$$
$$Verif: D_{Kp}(C) = M \ ? \ accept : reject$$

FORGERIES: I want to show you an existential forgery. Remember: forgery is the ability of the adversary to find a pair (Message,Signature) such that who is verifying the signature will obtain *accept* (the message of the attacker has not been decided and sent by someone in advance, the attacker invented it). There are 3 types of forgery:

- existential forgery is the ability of the attacker to generate a valid pair (Message,Signature) where the message is a random message for which the attacker does not have the control (he cannot decide whatever message).
- selective forgery is the ability of the attacker to create a valid pair (Message,Signature) where the message is decided in advance before running the attack (this does not means the message can be whatever he wants, maybe it has to follow some constraints, some matemathical rules, but not whatever message). This forgery implies the existential forgery so it is a **stronger** attack.
- universal forgery is the the ability of the attacker to create a valid pair (Message,Signature) such that the attacker has a complete control on the messsage, he can design whatever message he wants and create for it a valid signature. This is the strongest kind of forgery and it means that Universal

forgery => Selective Forgery => Existential Forgery. This does not mean that preventing Universal Forgery we prevent even Selective Forgery.

U => S => E therefore !E => !S => !U
Therefore we need a method to prevent the **Existential Forgery**.

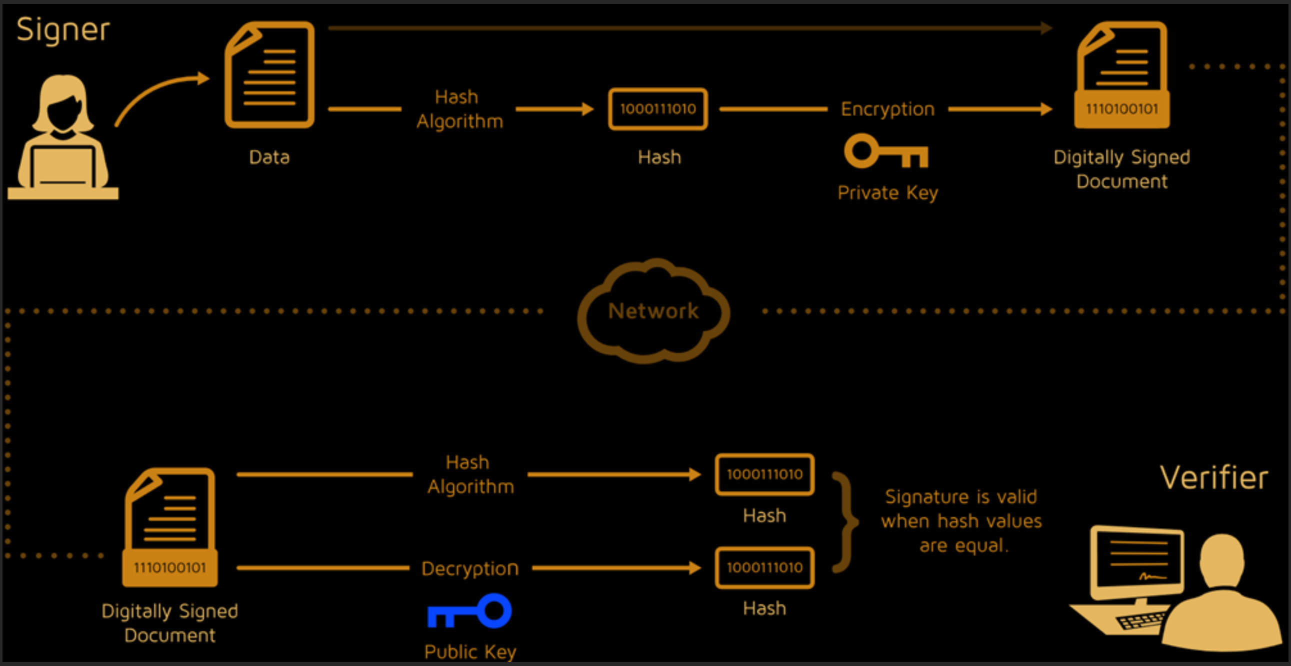
Example of forgery attack: the attacker generates a random file T and encrypts it with the K_p of Bob. Let's call the file T and the cipher R, so $R=E_{Kp}(T)$, then the attacker sends R as plaintext and T as digital signature. Alice then will follow the definition we gave, so she takes the digital signature T and so she encrypts T with k_p of Bob and she gets $E_{Kp}(T) = R$, but since the attacker sent R as plaintext, Alice sees that the decryption of the digital signature matches with the plaintext sent, so she accepts the document. That's why all this is not good.

Correct definition of DIGITAL SIGNATURE

We have to introduce another step, that requires hashing. The correct scheme that it has to be followed when creating a Digital Signature is the following:

$$Sig(M) = E_{Ks}(H(M))$$
$$Verif: H(M) == D_{Kp}(Sig(M)) \ ? \ accept : reject$$

That is, the signature of the document M is the *encryption* by means of my own private key K_s made on the HASH of the document. What is the verification step? Bob is sending(M,Sig(M)), then Alice takes M, computes H(M), then she takes Sig(M) and decrypts it with K_p of Bob. She should get that: $D_{Kp}(Sig(M)) = H(M)$. If she gets it she accepts, otherwise she rejects!



With hashing we get two benefits:
1. The previous attack cannot be performed anymore (we will see why).
2. The process of signing is much faster: we are using asymmetric encryption, which is slow, but here it is performed on the *hash* of the document which is much smaller then the plain document.

Note that the hashing function is a public hashing function and it is not keyed.

According to the new definition, if the attacker wants to perform a forgery attack he has to get a string and he has to encrypt it with the public key of Bob K_p, according to the new definition this will be the Hash that the victim expects to find decrypting the ciphertext with the K_p of Bob. So what the attacker need to to now is, starting from this hash (that is just E_{Kp}(string)), get the pre-image, that is the document that has as result of the hashing that hash. If he finds a colliding document of course the attack succeedes, but it is hard and it would not be an *universal forgery attack*. Obviously the attacker gains the complete power when he discovers the private key K_s of Bob, with which he can sign whatever document he likes, but without it the attacker can just perform attacks finding two document with same hash.
So we see that the previous attack needs an extra step here, computing a document that has that digest. So the security of this approach relies completely on the security of the hash function.

If we remember the definition of RSA, it had a multiplicative property such that given 2 messages M1="I owe bob $20" and M2="100" the outcome of the encryption of their product will be the cuncatenation of the two corresponding ciphers. So the previous definition of Digital Signature suffered for this weakness. With the new definition this is not possible anymore (no possibility to exploit the multiplicative property of RSA, the hash of the product is not the cuncatenation of the hashes).

Standards for Digital Signatures

RSA + PKCS#1

We already saw the PKCS#1 standard. With RSA we generate the pair of keys, then we construct the *digital signature* by encrypting with our private key the following message:

0 || 1 || at least 8 byte FF || 0 || specs on hash function || hash(M)

Where:

- The first byte 0 ensures that the message is less then n
- Second byte 1 denotes *signature* (if 2 it means that message is encrypted for *confidentiality* purposes)
- The bytes 111.. imply that the encoded message is large
- M is the original message

"As a conclusion for RSA I want to stress the concept that when you are **using RSA** you want to sign, the other typical usage of RSA is send an encrypted key. You don't use RSA for confidentiality purpose, for encrypt a document, you encrypt just a key. Because of its complexity, computational effort, much bigger than Rijndael."

El-Gamal Signature Scheme

This scheme uses two parts. The first one is used to generate the pair of keys, the second for making the gidital signature.
Generation of the key:

- Choose a prime p of many bits (1024) such that Discrete Logarithm in Z_p* is hard.
- Find a generator g of Z_p*.
- Pick a random number x in [2, p-2].
- Compute y= g^x mod p.
- Then K_p = (p,g,y) and K_s = x.

Generation of the signature:
If you want to sign a message M we define a new pair of keys, (r,k). Why need an extra pair? Because the proposal was "let's make a standard usage of our keys but also generate temporary keys for running the encryptions". If we do that everytime we sign the document we will get a different signature every time. Let's call r and k these new keys. These keys are temporary. Start by considering the original message and compute the digest m = H(M). Now we can pick a random k in [1,p-2] relatively prima to p-1. Compute r=g^k mod p.

Compute the digest s = (m-rx)k^{-1} mod (p-1) (if it is 0, restart). Output signature will be (r,s). Then verification step is shown on slide 16.

DSS (Digital Signature Standard)

This is important because is the choice made by NIST, it is inspired on El-Gamal approach and according to the original formulation DSS is using SHA, just because the hashing function chosen by the NIST is SHA. There are two acronimous: DSS (digital signature standard) and DSA (digital signature algorithm). The second is just an algorithm in order to get the digital signature. We will see DSA now. The idea is working with 2 prime numbers p and q such that:

- *p* is a L bit prime number such that the Discrete Log mod p is intractable
- *q* is a 160 bit number that divides *p-1*, i.e. $p = jq+1$

Then let α be a q-th root of 1 mod p, i.e. $\alpha^q=1 \bmod p$. It easy to compute α.

"In three times i asked at the exam the question: compute the 7th root of 1. The problem is, given p and q, to find α s.t. $\alpha^q = 1 \bmod p$. In our case $\alpha^7 = 1 \bmod p$."

In order to compute α we take a random number *h* s.t. $1<h<p-1$. Then we compute $g=h^{(p-1)/q} \bmod p$. If $g=1$ we have to try another *h* for security reasons. Then it holds that $g^q=h^{(p-1)/q\ *\ q} \bmod p = h^{p-1} \bmod p$, which is equal to 1 mod p by the *Fermat Theorem* (because p is prime, so every number less than p, raised to *p-1* is equal to 1 mod p). Finally choose $\alpha=g$.
Now let's go to the algorithm: choose p and q both prime, such that *p-1 = 0 mod q (p-1 is a multiple of q)* and consider α as the *q-th* root of 1 mod p (as we just showed). Then the private key is *s* (a random between 1 and q-1) and the public key is ($p,q,\alpha,y= \alpha^s \bmod p$). Is it hard to find s? It is the problem of computing the discrete log, and choose t so that the problem of computing the discrete log mod p of the multiplicative group Zp* is hard. So it is hard by hypotesis. What is the signature of M? The computation is again inspired by ElGamal. Choose a random *k* between 1 and q-1, then the signature will be compowed by 2 parts:

1. Part I = ($\alpha^k \bmod p) \bmod q$
2. Part II = $(SHA(M) + s(\text{Part I}))\, k^{-1} \bmod q$

The process of verification is shown on slide 21. The proof is on slide 22.

"DSA != ElGamal. Many people state that if you are using DSA you are using ElGamal. Actually this is just philosophy, but in some sense it is true (DSA is an optimization of ElGamal)."

Note: verification of DSS is more demanding then RSA (it requires two exponentiations, then two mod operations).

DSS versus RSA

Imagine a company asks you to design a software that should be authenticating itself by using some digital signature, what are you choosing? RSA or DSS? If key is long enough both are good. The secondo point is: there is an important difference between the usage of … standard (RSA) and NIST standard (DSS). If the company is the american company it prefers NIST standard. RSA is preferred by Open Source community because it is smoother and there are lots of implementations.

DSS is used for signature, RSA is used both for signature and for key management. Diffie Hellman is used just for key managment.

Timestamping a document

When a document needs to be signed, in many cases it is important for a formal point of view that also the date, the time is certified. This is timestamping. Timestamping a document means associate a timestamp to a document. In many cases there is a third party, an authority, who signs the document with the timestamp. So what happens is that we send to the authority the document togheter with an hash function, the authority

will add the timestamp to the document and will has it, then he will sign the hash and will give back to us the timestamped document, the hash and the signature of the hash.
Weak part: imagine I want to perform an attack such that a document appears to be signed by another part P, I generate my pair of keys K_s and K_p, then i perform the signature by means of K_s, then i go to the victim and I say “this is the Public Key of P, trust me!”, the victim will check the signature by means of the wrong key (wrong means not honest, but it is the key that verifies the signing), then the verification says ‘good’. What is the problem in here? It is associated to the identity of the one who gave the public key. Here is where attackers are strong. So there are mechanisms to verify the ownership of the public key, this is the concept of Digital Certificate, we will study it.