DIGITAL SIGNATURES

Whitfield Diffie once stated that his and Martin Hellman's invention of public-key cryptography was the result of "two problems and a misunderstanding." The misunderstanding was this: Diffie and Hellman assumed that users of the cryptographic system would not want to have to trust any third party in order to complete their connection. The first problem, which we have already dealt with, was: "How can two people, who have never met before, carry on a secure conversation?" Now, the second problem was the problem of authentication: "How can the recipient of a digital message be sure of who the sender was?" Here is where digital signatures come into play.

A digital signature is an electronic, encrypted, stamp of authentication on digital information such as email messages, macros, or electronic documents. It should be difficult to forge, and it should be difficult to remove from one document and attach it to some other document. It is interesting that the tools used to construct digital signatures are very similar to the tools used to construct asymmetric ciphers.

Here is the exact problem that a digital signature is supposed to solve. Let's introduce Samantha, who has a (digital) document D (for example, a computer file). Suppose she wants to create some additional piece of information D^{Sam} that can be used to prove conclusively that Samantha herself approves of the document. Samantha's digital signature D^{Sam} may be viewed as analogous to her actual signature on an ordinary paper document.

COMPONENTS OF A DIGITAL VERIFICATION SCHEME

Despite their different purposes, digital signature schemes are similar to asymmetric cryptosystems in the sense that they involve public and private keys and invoke algorithms that use these keys. Here is an abstract description of the pieces that make up a digital signature scheme:

K^{Pri}: A private signing key

K^{Pub}: A public verification key

Sign: A signing algorithm that takes as input a digital document *D* and a private key K^{Pri} and returns a signature D^{sig} for *D*

Verify: A verification algorithm that takes as input a digital document *D*, a signature D^{sig}, and a public key K^{Pub}. The algorithm returns *True* if D^{sig} is a signature for *D* associated to the private key K^{Pri}, and otherwise it returns *False*.

Importantly, the verification algorithm does not know the private key K^{Pri} when it determines whether D signed by K^{Pri} is equal to D^{sig}. The verification algorithm has access only to the public key K^{Pub}.

GENERAL CONDITIONS FOR A SECURE DIGITAL SIGNATURE SCHEME

Necessary general conditions for a secure digital signature scheme include the following:

- Given K^{Pub}, an attacker cannot feasibly determine K^{Pri}, nor can they determine any other private key that produces the same signatures as K^{Pri}
- Given K^{Pub} and a list of signed documents $D_1, \ldots, D_n$ with their signatures D_1^{sig}, ..., D_n^{sig} , an attacker cannot feasibly determine a valid signature on any document D that is not in the list $D_1, \ldots, D_n$.

The second condition is rather different from the situation for encryption schemes. In public key encryption, an attacker can create as many ciphertext/plaintext pairs as she wants, since she can create them using the known public key. However, each time a digital signature scheme is used to sign a new document, it is revealing a new document/signature pair, which provides new information to an attacker. The second condition says that the attacker gains nothing beyond knowledge of that new pair. **An attack on a digital signature scheme that makes use of a large number of known signatures is called a transcript attack.**

RSA DIGITAL SIGNATURE

The original RSA paper described both the RSA encryption scheme and an RSA digital signature scheme. The setup is the same as for RSA encryption; our signer Samantha chooses two large secret primes p and q and she publishes their product $N = pq$ and a public verification exponent v. Samantha uses her knowledge of the factorization of N to solve the congruence

$$sv \equiv 1 \bmod (p-1)(q-1)$$

In the present setup s is her signing exponent and v is her verification exponent. In order to sign a digital document D, which we assume to be an integer in the range 1 < D < N, Samantha computes

$$S \equiv D^s \pmod{N}$$

Our verifier Victor verifies the validity of the signature S on D by computing $S^v \bmod N$ and checking that it is equal to D.

This process works because Euler's formula tells us that

$$S^v \equiv D^{sv} \equiv D \pmod{N}$$

If an attacker Eve can factor N, then she can solve the first congruence for Samantha's secret signing key s. However, just as with RSA encryption, the hard problem underlying RSA digital signatures is not directly the problem of factorization. In order to forge a signature on a document D, Eve needs to find a v^{th} root of D mod N. This is exactly analogous to the RSA decryption hard problem, in which the plaintext is the e^{th} root of the ciphertext.

INEFFICIENCY OF DIGITAL SIGNATURES FOR LARGE DOCUMENTS

The natural capability of most digital signature schemes is to sign only a small amount of data, say b bits, where b is between 80 and 1000. It is thus quite inefficient to sign a large digital document D, both because it takes a lot of time to sign each b bits of D and because the resulting digital signature is likely to be as large as the original document.

The standard solution to this problem is to use a **hash function**, which is an easily computable function

$$Hash : (arbitrary - size - documents) \rightarrow \{0, 1\}^k$$

that is very hard to invert. Then, rather than signing her document *D*, Samantha computes and signs the hash *Hash(D)*. For verification, we have Victor who computes and verifies the signature on *Hash(D)*.

Let us now look at Hash functions.