Section 7 Lecture 44 – Digital Signatures

In face-to-face communications, we often use a *signature* to verify ourselves – the idea being that a signature is unique to a person and no-one else can replicate it, so it confirms their identity. But how can we do this in electronic communications? Let us look at a way in which we can "sign" messages using public-key ideas. As usual, this document will just contain the basic ideas and steps needed, we will look in much more detail about the ideas in the lecture slides.

Hashing

First of all we need the concept of a *hash function.* A hash function (or just *hash*) is some defined method for turning a message into something else, with the idea being that once it has been hashed, the original message can never be recovered.

Note the important difference between encryption and hashing. In encryption, we want to recover the message (decrypt) at the other end, whereas with hashing, it is a one-way process, so that once hashed, the original message can never be recovered.

There are several well-known hashing algorithms – I will discuss this further on the slides, for now it is enough for you just to know what a hash is and know that it can be applied to a message.

Digital signatures

Suppose we are sending a message from Alice to Bob using some public-key method (say RSA), which means Alice is using Bob's public key to encrypt her message, so that only Bob can decrypt it.

Recall that this gives us security (only Bob can decrypt it) but we have no way of confirming it actually was Alice that sent it, it could have been anyone. What we would like is a *signature* to confirm it really came from Alice. We can create such a "signature" using public key ideas as well.

The steps are summarised here – again, I will talk through it on the slides in the lecture.

- Alice takes her message m and applies some agreed hashing algorithm to it to obtain the hash h
- Alice then encrypts the hash h using her private key to obtain her signature s
- She then encrypts the message m and signature s using Bob's public key and sends these to Bob

Upon receipt, Bob can decrypt the message in the usual way, but he can also check the signature, following these steps:

- Bob decrypts the transmission using his private key to obtain the message m and the signature s
- He then decrypts the signature s using Alice's public key to obtain the hash h

- Finally he applies the agreed hashing algorithm to *m* – if everything is in order then he should obtain exactly the same hash *h* as in the previous step

This will probably take a while to click with you. It might well help you to draw this pictorially – I'll draw some diagrams on the slides, and hopefully the tutorial exercises will give you some practice and help this become clearer to you.

Remember, the encryption and decryption of the message is exactly the same as our previous public-key approach – the new thing here is the hash, which only Alice can encrypt to form the signature *s*, and so it can only be decrypted from Alice's public key if it really was created by Alice.

P.A.I.N. categories

We already had *privacy* through our usual public-key approach.

But now we have *authentication* as well – only Alice can create the signature so if the hashes match the message must actually have come from Alice.

This system also provides *integrity* – if the message was tampered with or corrupted in some way, then the hashes will no longer match and so something has gone wrong.

Non-repudiation is also provided here – the signature is effectively acting as a guarantee of the transaction.

Hence we have seemingly addressed everything we wanted to!

Summary

Remember, the crucial thing is the checking of the hashes, that they match. If they don't match, then something has gone wrong – either the message didn't come from Alice, or it was altered in transmission.

Take some time to think about this principle, and hopefully it will start to sink in!