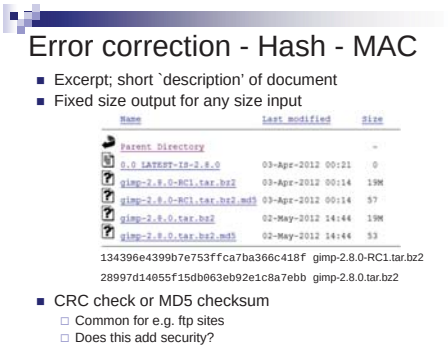
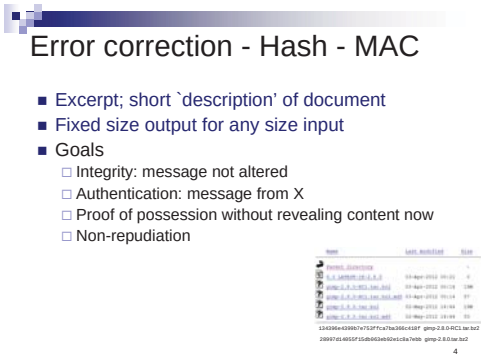
4 Certificates and Trust

4.1 Introduction

As mentioned before, asymmetric cryptography can be used to digitally sign messages; Alice can use her private key to create a signature that everyone can validate using her public key. The sign-validate structure is very similar to decrypt-encrypt and for some public key systems (e.g. RSA) signing can actually be implemented as decryption. For large messages decryption the entire message requires significant effort and we do not need to recover the message from the signature (it may even be better if we cannot). We only need to validate that the signature is actually a signature for this message. Thus it would be sufficient, and more efficient, to make a 'secure' digest of the message and sign (decrypt) that instead. This is where hash functions come in. Below we first discuss hash functions and related notions before moving to their use in digital signatures and certificates which in turn are used in Trust Management.

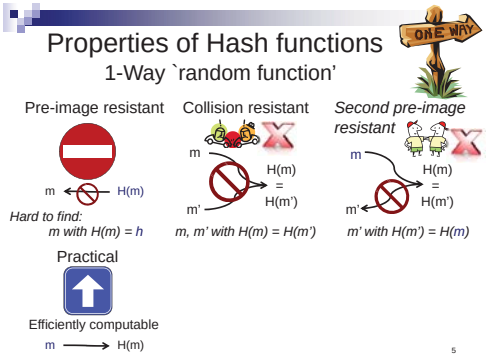
4.2 Hash functions

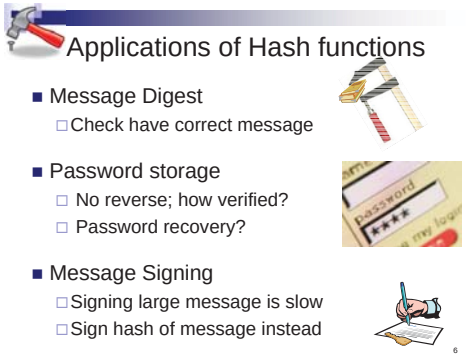




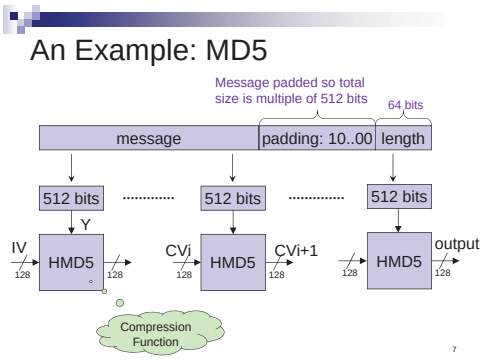
Hash functions are used to create 'secure' fixed size digests of messages; for any length input they create a fixed size hash value. The digest is secure in that it is clearly a digest of the message but does not reveal information about the message. Thus with the message we can check that the hash belong to that message but without the message we learn nothing from the hash.

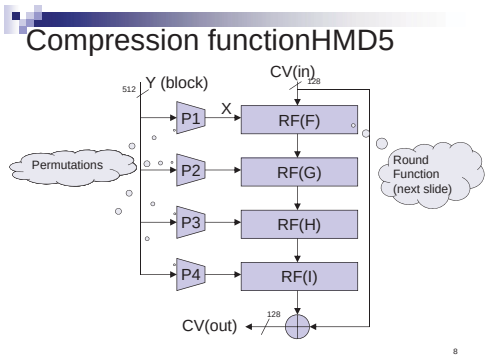
Error correction codes, such as a Cyclic Redundancy Check (CRC), are also digests of a message. These allow checking for changes and finding the correct original value after changes. However, the CRC reveals information about the (structure of the) message to which it belongs. One of the goals of the hash is to be *one-way*; the hash value of a message should not reveal information about that message. A good hash functions is thus a 'random function'; every different input gives a completely different (random) output. Message authentication codes (MACs) in turn aim to authenticate a message; thus not only detect changes but also validate the source of the original message. A keyed-hash where the hash value can only be computed using a key is one way to create MACs.





Several applications include password storage and, as already mentioned message signing. The problem with passwords is not that they are too long but that you would like to be able to store them securely on a system that needs to check them. If, instead of the password, we store the hash of the password it is difficult to recover the original password. Still, we can check if an entered password is correct by hashing the entered value and comparing it with the stored hash value of the correct password. As the hash is *collision resistant* entering a wrong password will not lead to the same hash. Collision resistance also ensures that when we sign a hash it can only be used for the correct message. (Note that collision resistance means it should be hard to find collisions, that collisions exist is a given; think about why.) Sometimes it is sufficient the has is only *second pre-image resistant*; for a specific message there is no way of finding a second message with the same hash. If a single collision has been found, but it is between two messages we are not interested in anyways, collision resistance is broken but we may not care. We only have a real problem when the hash of a message that we would sign is the same as the hash of one we would not sign.

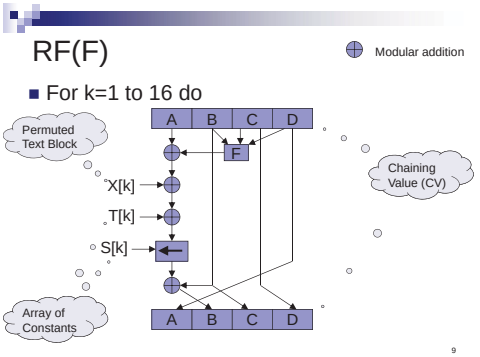


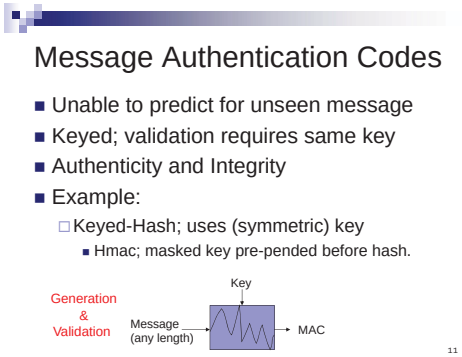


As an example of how a hash function may operate we have a short look at the well known MD5 hash function. It used to be widely used but this is not advisable any more; MD5 has been broken both in theory and in practice. Yet it still serves well as an example of a typical hash function construction. The function has to work on arbitrary size inputs. To achieve a compression function (HDM5) is iteratively applied to (512 bit) blocks of input. The compression function takes two inputs; the current intermediate hash value 'CV' (initially equal to initialization vector IV) and the block of input. The input is extended to a multiple of 512 bits by appending the length of the input preceded by padding as needed. The HDM5 uses rounds of permutations, shifting and addition (XOR) and non-linear functions to mix the input with main structure of MD5; this confusion-diffusion strategy it shares with block ciphers such as DES. Small changes in the input have an avalanche effect which causes the output to be completely different.

Hashes often share some constructions with (symmetric) ciphers; they both try to create a 'random output' but hash function have the advantage that they do not have to be reversible at all. Though obviously dependent on the actual algorithms used, hashing a message typically takes

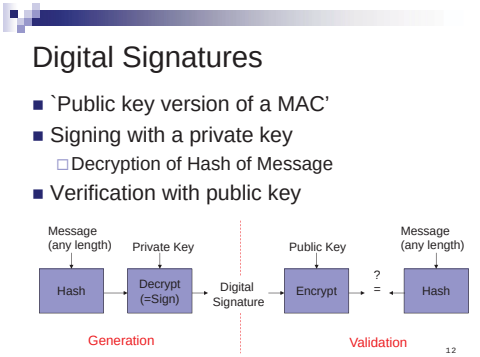
less effort than symmetrically encrypting the same message which in turn takes less effort than asymmetrically encrypting the message.

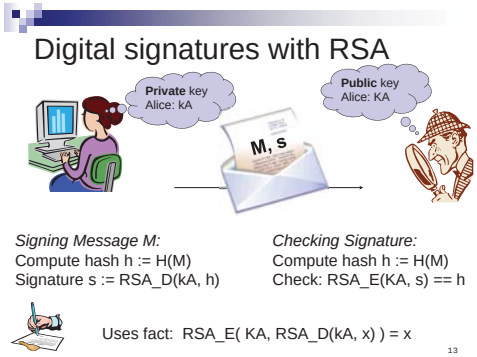




4.3 MACs and Digital signatures

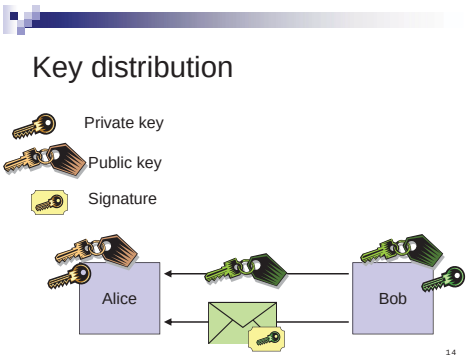
MACs were already discussed shortly above. The can be seen as a combination of hashing and symmetric cryptography; a MAC ensures the authenticity of a message as it is linked to the message and only someone with the key can create the MAC. It takes the same key to regenerate and thus check the validity of the MAC. Digital Signature are the public key version of MACs; it combines hashing and asymmetric cryptography. A private key is used to authenticate the message by hashing it and signing(decrypting) it and every one can validate the signature by encrypting the signature with the public key to recover what the hash of the message should be.

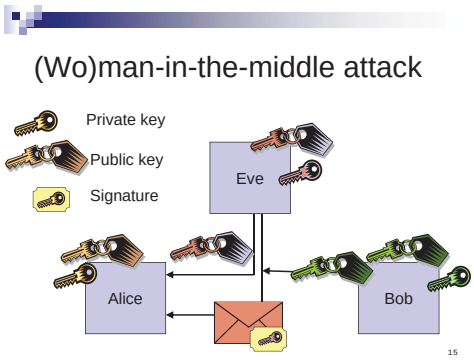




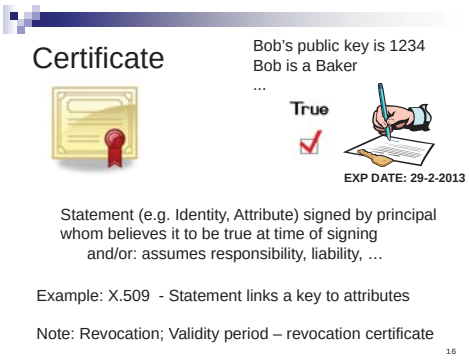
Digital signatures can be done with RSA by using decryption with the public key (of the hash of the message). Recall that encryption and decryption in RSA are the same operation; raising to the power of the key. The order in which they are performed does not matter.

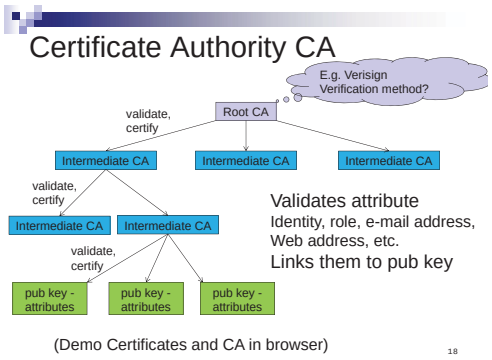
A main problem with validating digital signatures is getting the right public key. (Recall the first lab session.) If Eve can convince Alice that her public key is actually the public key of Bob then she can sign messages that Alice will believe come from Bob. To use digital signature we have to ensure we have the correct public key. In other words, the level of trust you can place in the signature on a message is only as high as your trust in correctness of the public key. Below we will see different techniques to ensure Alice can trust information that she gets, including public keys.





4.4 Certificates and Trust management





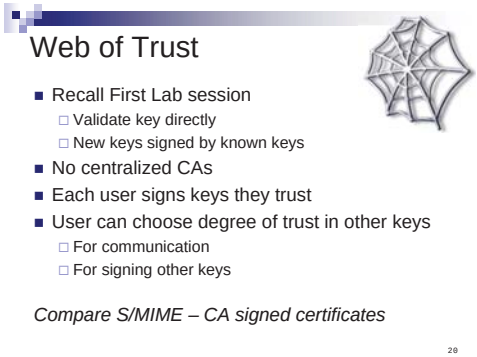
If Alice trusts Bob (and has his public key) than Alice can also trust statements that Bob signs. The level of trust in the signed statement by Bob, i.e. his certificate, can depend on how sure we are that we have Bobs key, how strong the signature scheme is, the level of trust in Bob and the statement that Bob is making. For example, Bob may be an expert on baking so we will trust a bread recipe but not a medical prescription.

When Alice is not sure that the public key p is really the public key of Bob, she cannot trust statements signed with p. When Alice and Bob meet in person they can share a public key (or at least a hash of that key-we used this method in the first lab session). If Alice and Bob can only communicate over an insecure channel (e.g. the internet) then they have to consider the man-in-the-middle attack given above where Eve substitutes her key for that of Bob. However, if Alice trusts Charlie who already knows the key of Bob then Charlie could issue a certificate stating 'the public key of Bob is 1234'. As the statement is signed by Charlie, Eve can no longer change the key without Alice noticing.

In the example above Alice grants Charlie the role of Certificate Authority (CA); she trusts that Charlie is an authority on the key of Bob. How does she know Charlie is an authority? There could be another authority Daisy that says so. How does she know Daisy is an authority on authorities? This chain can be continued for a while but will need to end as some point at a *root CA* Rob that Alice already trusts without the need for other authorities vouching for Rob.

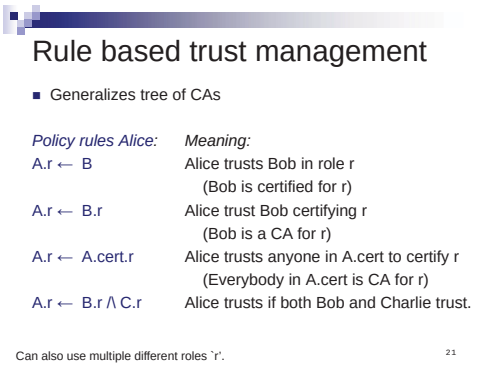
In HTTPS this approach is used to check certificates which authenticate the website you are connecting to. E.g. an (intermediate) *Certificate Authority* Terena CA checks the identity of the tue and its key and signs the statement the public key for `www.tue.nl` is *9d 79 10 57 e9 38 db 1a a2 94 bd aa 6f e4 b4 82 31 0e cd c7 41 13 dc f6 f4 f8 fc ed 9d ab ce 14 0a ea e9 86 60 68 ab 2e 6d 2e 2b 10 b1 c1 26*

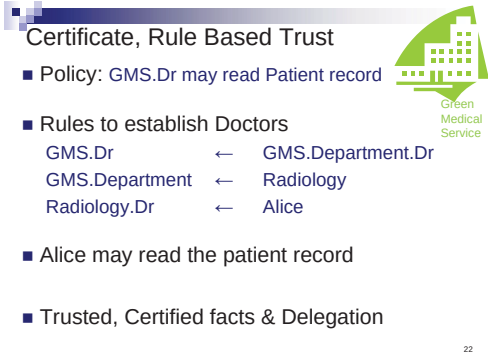
18 93 6e 85 8c cb 09 9a 26 4f 77 be d0 54 92 f4 8e b4 b0 93 20 13 6c c2 4e 79 c4 20 85 9c a8 ff 34 23 9c d7 c0 09 ca 67 0b b2 b7 55 b2 dd 88 e9 20 bd 19 e9 d0 1f c8 a3 11 a6 02 ba 3d fc 81 a5 48 15 84 6a 28 9a a0 e0 0a 7a 61 4c 1e c9 40 60 1c dd 26 a5 b9 3a e8 50 50 b5 f6 07 43 d0 55 b8 ed b3 f5 48 ea 8a a1 1a 6f 78 9f 65 ff 93 22 3d f7 1c fe dd eb 62 ab 73 02 66 b8 3b 76 59 60 90 1f 8f 49 3e 89 c2 2a e4 45 e0 9f fc 38 ad 4d 1e 27 7f a7 a2 5c 32 d1 01 51 17 a2 94 3b af 2d 90 09 33 ed 64 61 9a ec 6a fe d3 96 b9 c4 9f f1 83 8d e2 82 5b 5a 3d bc 4a 9b 73 49 7f c6 10 6b b1 93 85 b3 c5 6f ab 16 db 4d 5b ca 5a fe 55 d6 dc 4b. However, as I do not know Terena and its keys, I need to find out whether I can trust this key and trust Terena CA to issue such statements. For this the Terena has a certificate from another CA, UTN, that validates Terena's key and states that Terena is trusted to issue certificates for websites. This chain of certificates finally ends at a root CA; a well known root of trust that I know and have the public key of (typically built into my browser and/or operating system).



A potential problem with this approach is that trust is full and fully transitive; the root and intermediate CAs are trusted to only certify fully trustworthy intermediate CAs and correctly verify the identities of all websites they issues certificates for. However, if one step in the chain fails the whole system can break down. Several incidents show that this can indeed happen. Hacking into the systems of the CA is one way to obtain fake certificates. For example the recent Turktrust incident (e.g `http://technet.microsoft.com/en-us/security/advisory/2798897`,`http://www.theregister.co.uk/2013/01/04/turkish_fake_google_site_certificate/`) the Comodo incident (`http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`) and the well known Diginotar incident (also 2011). Use of outdated cryptography (such as the MD5 hash) also creates a risk; it has been demonstrated that a fake CA certificate can be created by using MD5 hash collisions (`http://www.win.tue.nl/hashclash/rogue-ca/` with video at e.g. `http://dewy.fem.tu-ilmenau.de/CCC/25C3/video_h264_720x576/25c3-3023-en-making_the_theoretical_possible.mp4`).

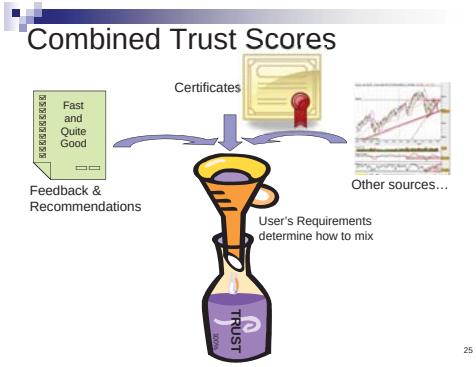
An alternative to the hierarchical trust model with a 'super' trusted entity at the top decides who is trusted for what is to use the 'grass roots', bottom up, model of web of trust (again: recall the first lab session). In this model there are no centralized sources of trust; instead everybody establishes trust locally and tries to expend this through overlapping trust networks. For example; in the lab session you may have checked the public keys of your neighbours yourself so you trust them. The neighbours of your neighbours will have gotten a certificate from your neighbours which you can check with the keys you know and trust. If you trust the checks that your neighbours did then you can now also trust the keys of your neighbours neighbours. Of course, if you know that your neighbour is a bit careless and may sign keys without checking them thoroughly, you may decide not to trust the keys until you have more certificates that confirm this key. You can thus trust that the statement came from your neighbour (because the key is trusted to be correct) but still not trust the statement itself.





Adding the purpose for which we trust parties can be done with certificate based trust management languages such as RT [**?**]. For example, a hospital 'GMS' can state that it trusts its departments to issue doctor credentials. Combined with a certificate by GMS that Radiology is a department and from Radiology that Alice is a doctor this gives that GMS trusts Alice in the role of doctor (and thus will e.g. allow her to read patient records, see also Chapter **??** on Access Control).

Reputation systems try to quantify a level of trust in a party by using past experience. This can include our own experience but also the experience of other that interacted with that party. Their feedback on the behaviour of the party in those interaction helps establish the reputation of the party. (The reputation of the party providing the feedback may influence the degree in which we consider this feedback.)



Finally, one does not need to rely on only one source of trust information, scores provided by different systems can be combined and mixed (certified and good reputation, only accept certificates from reputable entities, only take into account feedback from certified doctors when computing reputations, etc.)

4.5 Conclusions and where to go from here

This chapter provides some basics of Trust Management. It discusses certificates, how they can be created and used to build trust from chains of certificates. Trust scores can be relayed to the user (e.g. through a lock indicating a correctly authenticated connection in a browser, an 'average feedback' score on a retailers website, etc.) or used in some automated way (e.g. as part of access control as touched upon in examples, for installing software/updates/..., etc.)

4.5.1 Literature

Suggested reading (check the course page [2] for the most up to date list of suggested reading materials):

- Security Engineering Introduction [3, Sec 5.6] on hash functions.
- Handbook of applied cryptography [8, Ch 9], Section 1 through 3 on hash functions.
- MD5 considered harmful today. Creating a rogue CA certificate. `http://www.win.tue.nl/hashclash/rogue-ca/`
- RT: A Role-based Trust-management Framework [**?**] (`www.stanford.edu/~jcm/papers/rt_discex03.pdf`)

4.6 Exercises

1. A FTP site publishes the hash of a file along with the file itself. Explain why this hardly has any *security* benefits. Given this, why is this done?
2. Suppose a hash function is used for Digital Signatures. Which properties of a hash are needed and which are not?
3. Argue why RSA signing is correct and complete; we can always create and check a correct signature and only Alice can sign her messages.
4. See the video on creating a fake credential authority on `http://www.win.tue.nl/hashclash/rogue-ca/` Suppose you have created a fake intermediate-CA certificate.
   (a) How could you use this certificate to steal someoneŠs login data?
   (b) What can 1) de (root) CA 2) the Browser maker 3) Microsoft 4) An honest user, etc. do to
      - prevent new fake certificates from being issued?
      - solve the problem of existing fake certificates?

      (Do you already see risks that these countermeasures are circumvented?)
   (c) (*) For El-gamal a reduction proof showing that breaking the it is as hard as solving a 'hard' problem. Why does such a proof not exists for MD5, SHA-1, etc.? Would it be possible to create a hashing schema for which such proves are possible?