Asymmetric Key Encryption

Asymmetric Key Encryption

- Different keys are used to encrypt and decrypt the plaintext.
- Keys now come in pairs.
- Less computationally efficient than symmetric key encryption.

Public and Private Keys

- One key is kept private.
- The other key is made public.
- Can be used for confidentiality (but typically is not because it is not as efficient as a symmetric key).
- Is used mainly for authentication.

Asymmetric Key Encryption

- Confidentiality
  - Amazon can receive data encrypted with their public key (as only they can decrypt it)
- Authentication
  - Amazon can use their private key to encrypt something.
  - If we decrypt it with Amazon's public key and it makes sense, then it must be from Amazon, as only they counld have encrypted it.

Public-Key Encryption Algorithms

- RSA (Rivest, Shamir and Adleman)
  - 1978
  - 1K to 4K key.
  - A 768 bit key has been broken
- DSA (Digital Signature Algorithm)
  - Adopted in 1993

RSA

- Published in 1978 by Rivest, Shamir and Adleman.
- Clifford Clocks working in GCHQ described an equivalent system in 1973.
- Not revealed due to top secret classification.

RSA

- 1K to 4K key.
- [A 768 bit key has been broken]
- Three phases
  - Key Generation
  - Encryption
  - Decryption

DSA

- The Digital Signature Algorithm (DSA)
- 1991
- The hash function used
  - originally SHA1
  - now SHA2 used.

DSA

- Key Generation
  - Choosing parameters for algorithm
  - Generate public/private key pairs.
- Signing
- Verifying

Digital Signatures and Digital Certificates

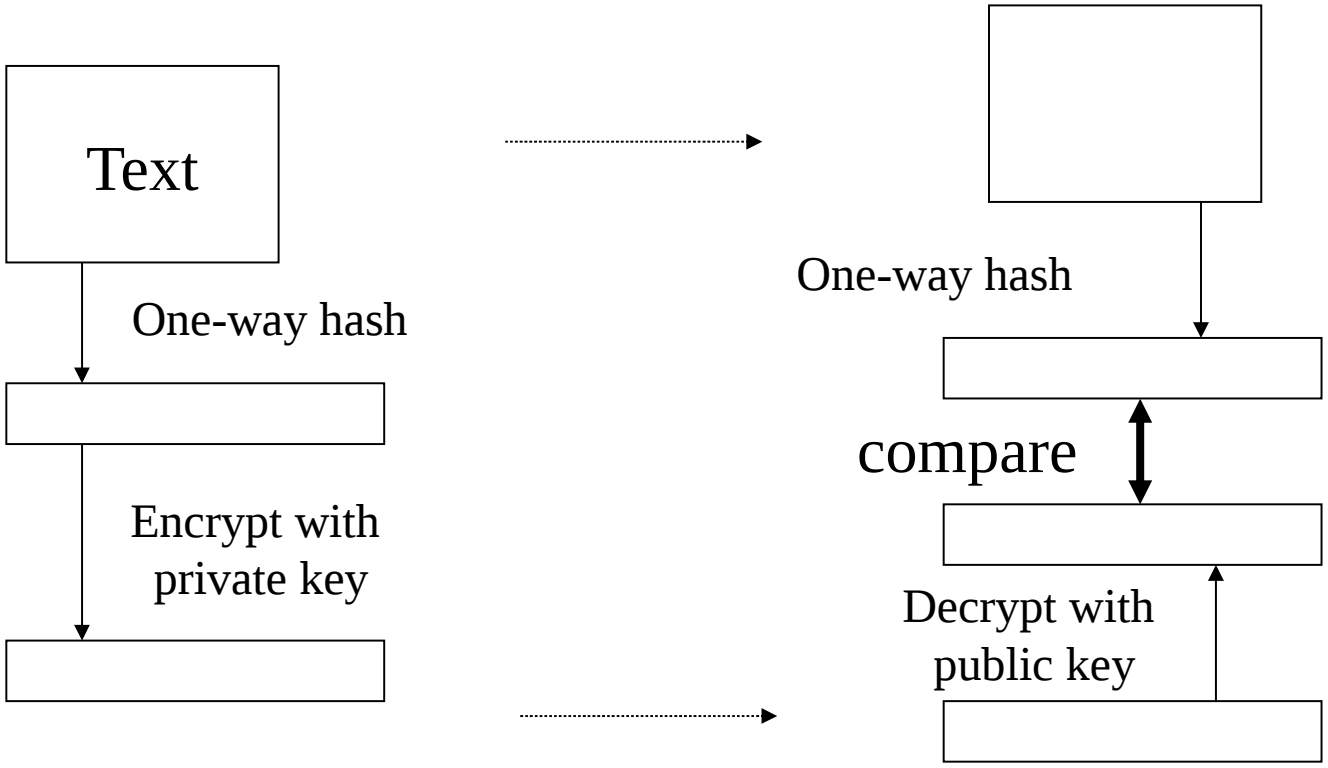
Authentication: Digital Signatures

- Authentication of a peer can be achieved, [if we are sure that we have the correct public key of the proposed peer], by using digital signatures.
- More about management of keys later (Digital Certificates).
- Digital signatures are not be forgeable.
- Therefore they also provide non-repudiation. You cannot claim later that it wasn’t you

Digital Signature - Authentication

- A digital signature is essentially
  - Some data
  - A one-way hash of the data encrypted with a private key.
- The receiving peer
  - decrypts the digital signature
  - hashes the data, using the same algorithm
  - compares the two
- Nobody but the person with access to the private key could have signed the data.

Digital Signature



Digital Signature - Integrity

- Notice that we also get integrity of data (not confidentiality)
- If the data has been changed, then its hash value won’t match the decrypted Digital Signature.

Authentication using Certificates and Digital Signatures

- Authentication is actually a two step process.

- First obtain and be sure that you have the right public key of the proposed peer (Digital Certificate)
- Secondly ensure that the proposed peer has the associated private key (using Digital Signatures).

Digital Certificates

Digital Certificates & Certificate Authorities

- Certificates are issued by Certificate Authorities (CA), who we trust (Verisign/Thawte).
- The Certificate Authority has a (well published) procedure for verifying applicants for their digital certificates.
- In effect the Digital Certificate is a “letter of introduction” by the CA.
- Verisign provide a range of Certificates:-

Digital Certificates

- Thawte Personal Basic CA
- Thawte Personal Freemail CA
- Thawte Personal Premium CA
- Thawte Premium Server CA
- Thawte Server CA
- Verisign Server CA
- Verisign Class 1 CA …….
- Verisign Class 4 CA
- Verisign Server CA

X.509 Digital Certificates

- Digital certificates are defined in ISO standard X.509
- An X.509 certificate binds a name to a public key.
- An X.509 certificate contains
  - name of the CA
  - the digital signature of the CA
  - serial number
  - validity period of the certificate
  - name of the owner (subject)
  - subject’s public key

X.509 Digital Certificates - Distinguished Names

- To be more precise, an X.509 certificate binds a Distinguished Name (DN) to a public key.
- A DN is a sequence of name-value pairs, e.g.

  uid=pjacob,e=pjacob@ait.ie, cn=Paul Jacob, o=Athlone Institute of Technology,c=IE

Digital Certificates

- If we trust the authority, then we can trust name/public key mapping.
- [Note that Digital Certificates on their own can not be used to authenticate a peer, as these digital certificates are freely passed around.
- Hence the need to determine that the proposed peer has the private key corresponding to the public key on the certificate and the use of digital signatures.]

Summary: Digital Certificates & Digital Signatures

- Send
  - Digital Certificate
  - Random piece of text
  - digitally signed (encrypted one-way hash of text)

Summary: Digital Certificates & Digital Signatures

- The receiver
  - Verifies the Digital Certificate, i.e. checks the Digital Signature on the Certificate (using the CA's public key).
  - Reads the Public Key of the sender from the Certificate.
  - Uses this to check the Digital Signature of the sender.

Public/Private Key Encryption - Java

Java Example

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
SecureRandom random = new SecureRandom();
random.setSeed(2345);
keyGen.initialize(1024, random);
KeyPair pair = keyGen.generateKeyPair();
PrivateKey privateKey = pair.getPrivate();
PublicKey  publicKey = pair.getPublic();
```

Java Example (cont)

```
// sending the data
Signature dsa = Signature.getInstance("SHA1withDSA");
dsa.initSign(privateKey);
byte[] sendText = "Sending Data".getBytes();
dsa.update(sendText);
byte[] sig = dsa.sign();

// receiving the data and verifying

dsa.initVerify(publicKey);
dsa.update(sendText);
boolean verifies = dsa.verify(sig);
System.out.println("signature verifies: " + verifies);
```

java.security API

java.security.KeyPairGenerator

- static KeyPairGenerator getInstance(String algorithm)
  - Returns a KeyPairGenerator object that generates public/private key pairs for the specified algorithm.
- public void initialize(int keysize, SecureRandom random)
  - initializes the key pair generator for a certain keysize with the given source of randomness

java.security.KeyPairGenerator

- public final KeyPair genKeyPair()
  - generates a key pair.

java.security.SecureRandom

- Provides a cryptographically strong random number generator (RNG).
- public SecureRandom()
  - constructor
- public void setSeed(byte[] seed)
  - Reseeds this random object. The given seed supplements, rather than replaces, the existing seed.

java.security.KeyPair

- PrivateKey getPrivate()
  - Returns a reference to the private key component of this key pair.
- PublicKey getPublic()
  - Returns a reference to the public key component of this key pair.

java.security.Signature

- Provide the functionality of a digital signature algorithm.
- Can sign text and verify signed text.
- To sign text you will need a private key.
- To verify a signature, you need the public key.

java.security.Signature

- static Signature getInstance(String algorithm)
  - Returns a Signature object that implements the specified signature algorithm.
- Algorithms
  - MD5withRSA, SHA1withRSA, MD5withDSA etc. etc.

java.security.Signature

- public final void initSign(PrivateKey privateKey)
  - initializes the object for signing (or verifying)
- public final void update(byte[] data)
  - updates the data to be signed (adds to it)
- public final byte[] sign()
  - returns the signature bytes of all the data updated
  - resets this signature object to its state before any updates.

java.security.Signature

- public final void initVerify(PublicKey publicKey)
  - initializes the object for verifying
- public final void update(byte[] data)
  - updates the data to be verified (adds to it)
- boolean verify(byte[] signature)
  - verifies the signature of the bytes updated
  - resets this signature object to its state before any updates.