Class 7: Digital signatures, v1

Note Title

(1) RSA digital signatures (signing whole message)

- Recall from last time: RSA uses
  - public exponent and modulus
  - private exponent

to encrypt: ciphertext = plaintext$^{\text{public exponent}}$ (mod public modulus)

to decrypt: plaintext = ciphertext$^{\text{private exponent}}$ (mod public modulus)

the recipient's exponent & modulus

- Very similar scheme to sign messages:

to sign: signature = plaintext$^{\text{private exponent}}$ (mod public modulus)

to verify signature: signed text = signature$^{\text{public exponent}}$ (mod public modulus)

the sender's exponent & modulus

i.e. to sign instead of encrypt, use sender's private key, not recipient's public key.
to verify instead of decrypt, use sender's public key, not recipient's private key.

Example, using keys on handout:

(a) A wants to sign the (unencrypted) message 13, and send to B.

- signature = $13^{7} \bmod 22 = 7 = 07$
- complete unencrypted message is 1307

(b) Same, but A wants to also encrypt.
- first sign unencrypted message, obtaining 1307.
- now encrypt the result (block by block):
  1st block ciphertext = $13^{5} \bmod 34 = 13$
  2nd block ciphertext = $7^{5} \bmod 34 = 11$
  → transmit 1311.

(c) After receiving the unencrypted, signed message 1307 from A, B wants to verify the signature.
- signedtext = $7^{3} \bmod 22 = 13$
- signedtext matches plaintext, so signature is valid.

(d) After receiving the unencrypted, signed message 1705 from A, B wants to verify the signature.
- signedtext = $5^{-3} \bmod 22 = 15$
- signedtext doesn't match plaintext, so signature is invalid.

(2) Digital signature for long message using hash function

So far our signatures are the same length as our messages — not good for signing, say, a 20GB software package.

Solution: Sign a cryptographic hash of the message instead of the whole message

Note: Our examples will use the "sum of digits" hash function h described on the handout.
(It's <u>not</u> a cryptographic hash; we <u>pretend</u> it is.

Examples:

a) B wants to sign message 3126.
- first compute $h(3126) =$ _____
- now sign: signature = _____ .
- transmit _____

b) B receives signed, unencrypted message 1314 15, purportedly from A. Was it really from A?
- compute $h(1314) = 9$
- unsign 15, obtaining $15^3 \bmod 22 = 9$
- the two values match, so A sent the message.

(3) Message integrity via digital signature

Note that a digital signature verifies both the identity of the sender and the integrity of the message

↑ i.e. the message wasn't altered

Why? Because the signature depends on the message!

Example: (a) Same as example (b) on previous page.
Has the message from A been tampered with?
Answer: No. The matching signature verifies A's identity and the integrity of the message.

(4) Real-life use of cryptography

(a) Implementation details and applications

RSA, and other public key schemes, are much more computationally expensive than symmetric key schemes.

Typically, therefore, an encrypted connection first establishes a shared secret (via Diffie-Hellman or RSA, for example), then uses that secret as the key for symmetric encryption (using chained-block AES, for example).

Some typical settings:

symmetric : 128-bit key

Diffie-Hellman : 1024-bit base and modulus

RSA : 1024-bit exponents and modulus

Some typical applications:

symmetric : any encrypted session (remote login, remote desktop, web page with https in the address)

Diffie-Hellman : establish key for symmetric session

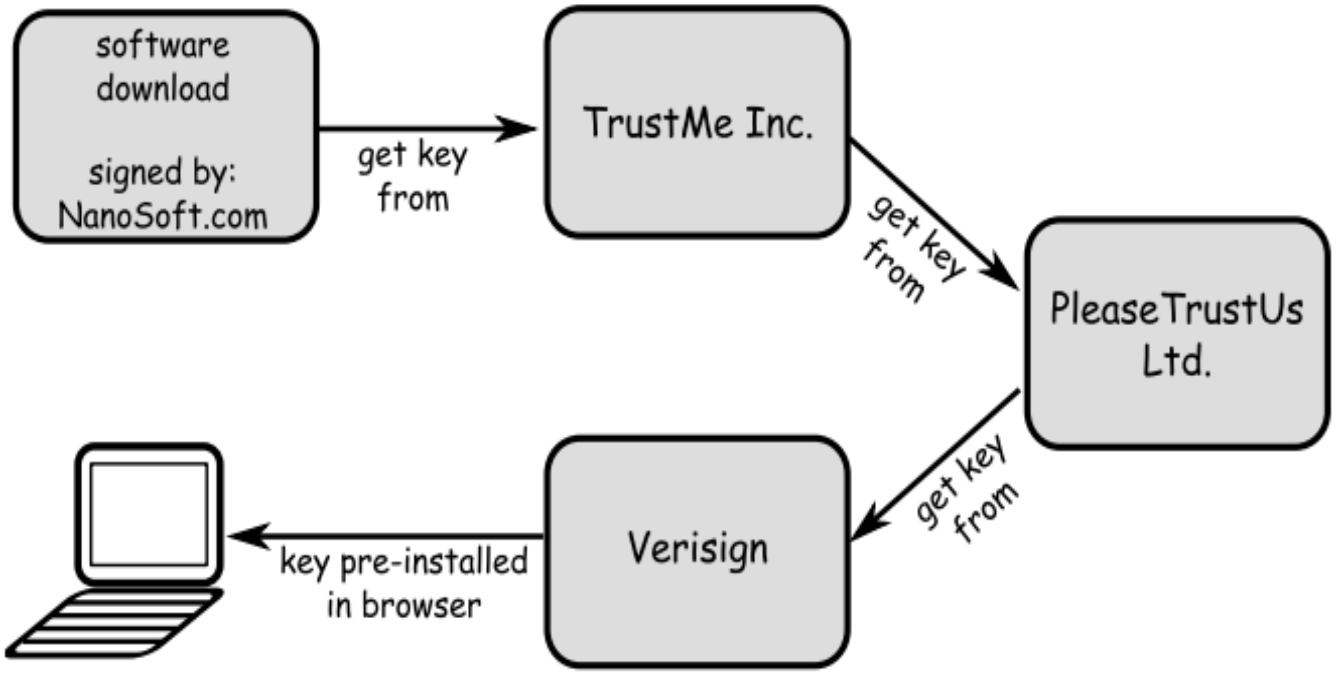
RSA : sign software, verify identity of a server (e.g. Amazon)

(b) How can we trust a public key?

Digital signatures rely on us knowing that a given entity (e.g. Amazon.com) has a given public key. What stops a criminal from publishing a false public key for Amazon, and using it to steal your credit card info?

Answer: A chain of trust via certification authorities (CAs). The keys of several popular root CAs come installed on your computer. See figure below for an example:

e.g. Verisign, GeoTrust



In recent years, there have been many problems with CAs mistakenly issuing keys to criminals, so this system is far from perfect.

see, for example:

http://googleonlinesecurity.blogspot.com/2013/01/enhancing-digital-certificate-security.html