COMPUTER & NETWORK SECURITY

Lecture 11:
Digital Signatures

SIGNATURES

Used to bind an author to a document

Desirable properties for a signature:

- **Authentic:** sufficient belief that the signer deliberately signed the document.
- **Unforgeable:** proof that the signer and no-one else signed the document.
- **Non-reusable:** the signature is intrinsically bound to the document and cannot be moved to another (i.e. be reused).
- **Unalterable:** the signature cannot be altered after signing.
- **Non-repudiation:** the signer cannot later deny that they did not sign it (most important of all).

As with all things, these properties can be attacked and subverted.

In designing systems involving signatures we must consider the effort of such attacks

DIGITAL SIGNATURES

If **m** is the message to be signed and **k** is the secret key known only to the signer, then **S = F(m, k)** binds the signature **S** to the message m for some signature scheme **F**

Given **(m, S)** anyone can verify the signature without the secret **k**.

Non-repudiation is achieved through the secrecy of **k**.

DIGITAL SIGNATURES USING PUBLIC KEY CRYPTO

Say Alice wishes to sign a message and send it to Bob

Generation of a key:

Alice generates public (verifying) and private (signing) keys

$\mathbf{A_S}$ is kept secret and $\mathbf{A_V}$ is published in a public directory

Signature Generation:

Alice chooses a random $\mathbf{r} = \mathbf{\{0, 1\}^n}$

Alice hashes the message **d = h(m)** using a collision resistant hash function (CRHF)

Alice generates $\mathbf{S = signature(d, r, A_S)}$

Alice sends **(m, S)** to Bob

Signature Verification:

Bob obtains $\mathbf{A_V}$ from the public directory

Bob computers **d = h(m)**

Bob runs $\mathbf{verify(d, A_V, S)}$

EXAMPLE OF DIGITAL SIGNATURES USING PUBLIC KEY CRYPTO

Attack Models

Total Break

Attacker can recover $\mathbf{A_S}$ from $\mathbf{A_V}$ and **(m, S)**

Selective Forgery

Attacker can forge signatures for a particular message or class of message

Existential Forgery

Possible only in theory (based on currently available resources)

REPLAY

Why might it include **r** = **{0, 1}**^{n} in the signature?

Consider the following scenario:

= Alice sends Bob a digital cheque for $100.

= Bob takes the cheque to the bank.

= The bank verifies the signature is valid and credits Bob's account.

What is stopping Bob from cashing the same cheque twice?
(i.e. perform a *replay attack*)

The random value **r** is known as a *nonce* and is used to avoid replay
(in other words it assures "freshness")

The bank keeps track of all nonces it has seen so far from Alice.

SIGNATURE BASED ON RSA

A naïve protocol based on RSA might be the following:

Key Generation:

$n = p \cdot q$ # p, q are large primes

$d \cdot e \equiv 1 \bmod \Phi(n)$

verify key = (n, e) # public key

signature key = (n, d) # private key

Signature Generation:

Assume $m \in Z_n^*$

$S = (m^d \bmod n)$ # RSA decryption

Signature Verification:

$S^e = m \bmod n$ # RSA encryption

PROBLEMS WITH THE NAÏVE RSA SCHEME

Eve can trick Alice into signing any message **m**

Based on RSA's homomorphic property:
If $\mathbf{s_1 = m_1^d \pmod n}$ and $\mathbf{s_2 = m_2^d \pmod n}$ then $\mathbf{s_1 s_2 = (m_1 m_2)^d \bmod n}$

Attack on naïve RSA scheme:
Eve wants Alice to sign hidden message **m**
Eve picks random $\mathbf{r} \in \mathbf{Z_n^*}$
Eve computes $\mathbf{m' = m \cdot r^e \pmod n}$
Eve asks Alice to sign **m'**
Alice returns $\mathbf{s' = (m')^d \pmod n}$
Eve computes $\mathbf{s = (s' / r) \pmod n}$

The pair (m, s) is a valid message signature pair!
Eve tricked Alice into signing hidden message **m**
Note that this trick also works with RSA decryption
(Eve can get Alice to decrypt messages if Alice is not careful)

PKCS#1 SIGNATURE SCHEME (RFC2313)

Public Key Cryptography Standards #1

Note the RSA naive signature scheme has message recovery (the verification function returns the message)

PKCS#1 processes a hash instead (far faster)

Signature Generation:

n = p . q # 1024-bit modulus

Alice calculates d = h(m) # 160-bit hash

Define EB (encryption block) = [00 | BT | PS | 00 | D]

= The header is essentially padding (PS)

= Block type (BT) dictates padding style

= EB is 864 bits + 160 bits = 1024 bits

Alice calculates $S = EB^d \pmod n$

Alice sends (S, m)

PKCS#1 SIGNATURE SCHEME (RFC2313)

Signature Verification:

$S = EB^d \pmod n$

Alice calculates $S^e \bmod n = EB \pmod n$

Alice tests the 864 most significant bits are valid, then

Alice tests the 160 least significant bits are valid [=h(m)]

EL GAMAL SIGNATURE SCHEME (DISCRETE LOG)

Key Generation:

Alice picks 1024 bit prime and generator $g \in Z_p^*$

Alice picks secret key $a \in Z_{p-1}^*$

Alice publishes public key $y = g^a \bmod p$

Signature Generation:

Alice picks random $r \in Z_{p-1}^*$

Alice hashes the message $d = h(m)$

Alice calculates $k = g^r \pmod p$ $0 \leq k \leq p-1$

Alice calculates $s = r^{-1(d - ak)} \pmod{p-1}$

Alice sends signature (k, s) and message m to Bob

EL GAMAL SIGNATURE SCHEME (DISCRETE LOG)

Signature Verification

Bob verifies that $0 \leq k \leq p-1$

Bob verifies that $g^d = y^k k^s \pmod p$

Note:

$y^k k^s = (g^a)^k (g^r)^{r^{-1}(d - ak)}$

$= (g^a)^k g^{(d - ak)}$

$= g^d$

NOTES ON EL GAMAL SIGNATURE SCHEME

It is unknown why this scheme is secure, but the obvious attacks don't work. Analysis shows:

= Attacker can't recover a from the public key data since this requires computing discrete log (which we know is hard).

= Picking **k** at random and trying to find **s** can't be done since it also requires discrete log.

= Picking **s** at random needs to solve $\mathbf{c = a^k k^s}$ (is this hard??)

Recent attacks have shown that:

= If weak generators are chosen, selective forgery can be done

= **r** must be random for each signature. If **r** is used twice, then an adversary can retrieve the private key **a**.

DIGITAL SIGNATURE STANDARD (DSS)

NIST 1991: Hash function is SHA-1

Key Generation:

Pick prime q (160 bits)

Pick prime p (1024 bits) such that $q \mid p - 1$

Pick $g \in Z_p^*$ of order q ($g^q \equiv 1 \bmod p$), Pick random $h \in Z_p^*$

Set $g = h^{(p-1)/q} \pmod p$

Iterate until $g \neq 1$

Pick random $a < q$

The public key is $\mathbf{y = g^a \pmod p}$, **p**, **q**, and **g**

Secret key is **a**

DIGITAL SIGNATURE STANDARD (DSS)

Signature Generation:

Pick random $r \in Z_q^*$ $(1 < r < q)$

Set $k = [g^r \bmod p] \bmod q$

$s = r^{-1} [h(m) + ka] \bmod q$

Signature Verification:

Obtain the public key

Test $1 \leq k < q$, $1 \leq s < q$

Set $w = s^{-1} \bmod q$

Test $[g^{wh(m)} y^{kw} \bmod p] \bmod q = k$

Note that

$$g^{wh(m)} y^{kw} = g^{s^{-1}h(m)} (g^a)^{ks^{-1}}$$
$$= g^{(h(m)r/h(m)+ka)} g^{(akr/h(m)+ka)}$$
$$= g^r$$
$$= k \pmod q$$

The main point is that the signature is only 320 bits

NOTES ABOUT DSS

The security analysis of El Gamal applies to DSS as well.

DSS is the standard for signatures for a number of reasons:

= DSS cannot be used for encryption

= Signatures are short

= Patent issues

DSS is based on the security of subgroups <g>

It is not known whether a sub-exponential algorithm exists in the size of the subgroup exists for discrete log.

DSS signature verification can be sped up by using simultaneous exponentiation (speed up of 2).

SIGNATURES BASED ON ONE-WAY FUNCTIONS

Lamport-Diffie Signature

Let $h: \{0,1\}^n \rightarrow \{0,1\}^k$ be a one way hash function.

Key Generation:

If the message is n bits long, we generate 2n x m bit numbers :

$$\{x_1^{(0)}, \ldots, x_n^{(0)}\}, \{x_1^{(1)}, \ldots, x_n^{(1)}\} \in \{0,1\}^m$$

The public key is $v_i^{(j)} = h(x_i^{(j)})$ for all i, j

The private key are all the $x_i^{(j)}$

Signature Generation:

For message $M = m_1 \ldots m_n$

The signature is $s = (x_1^{(m1)} \ldots x_n^{(mn)})$

i.e. we select block $x_1^{(0)}$ if bit 1 of m is 0, otherwise $x_1^{(1)}$

Signature Verification:

Bob tests that for all i, $h(x_i^{(mi)}) = v_i^{(mi)}$

SIGNATURES BASED ON ONE-WAY FUNCTIONS

Notes:

Only the sender knows the values of the **x**'s that produce the signature.

The main problem with this technique is that it involves a very long public key, which must be changed for every message sent.

Additionally, the message itself expands by a factor of **m** (each bit expands to a m-bit block). Since **m** must be sufficiently large to obviate an attack by exhaustively testing one-way function input numbers, the message expansion is considerable.

REFERENCES

Handbook of Applied Cryptography

– read § 1, §11 - 11.3.3, 11.3.6, 11.5