Digital signatures- DSA

Signatures vs. MACs

Suppose parties A and B share the secret key K. Then M, $MAC_K(M)$ convinces A that indeed M originated with B. But in case of dispute A cannot convince a judge that M, MAC_K (M) was sent by B, since A could generate it herself.

Problems with "Pure" DH Paradigm

- Easy to forge signatures of random messages even without holding D_A: Bob picks R arbitrarily, computes $S=E_A(R)$.
- Then the pair (S, R) is a valid signature of Alice on the "message" S.
- Therefore the scheme is subject to existential forgery.

forgery

ability to create a pair consisting of a message m and a signature (or MAC) σ that is valid for m, where m has not been signed in the past by the legitimate signer

Existential forgery

- adversary creates any message/signature pair (m,σ), where σ was not produced by the legitimate signer
- adversary need not have any control over m; m need not have any particular meaning
- existential forgery is essentially the weakest adversarial goal, therefore the strongest schemes are those which are "existentially unforgeable"

Selective forgery

- adversary creates a message/signature pair (m,σ) where m has been chosen by the adversary prior to the attack
- m may be chosen to have interesting mathematical properties with respect to the signature algorithm; however, in selective forgery, m must be fixed before the start of the attack
- the ability to successfully conduct a selective forgery attack implies the ability to successfully conduct an existential forgery attack

Universal forgery

- adversary creates a valid signature σ for any given message m
- it is the strongest ability in forging and it implies the other types of forgery

Problems with "Pure" DH Paradigm

- Consider specifically RSA. Being multiplicative, we have (products mod N)

$$D_A(M_1M_2) = D_A(M_1)D_A(M_2)$$

- If M_1="I OWE BOB \$20" and M_2="100" then under certain encoding of letters we could get M_1M_2 ="I OWE BOB \$20100"

Standard Solution: Hash First

- Let E_A be Alice's public encryption key, and let D_A be Alice's private decryption key.
- To sign the message M, Alice first computes the strings $y = H(M)$ and $z = D_A(y)$. Sends (M, z) to Bob
- To verify this is indeed Alice's signature, Bob computes the string $y = E_A(z)$ and checks $y = H(M)$
- The function H should be collision resistant, so that cannot find another M' with $H(M) = H(M')$

General Structure: Signature Schemes

- Generation of private and public keys (randomized).
- Signing (either deterministic or randomized)
- Verification (accept/reject) - usually deterministic.

Schemes Used in Practice

- RSA
- El-Gamal Signature Scheme (85)
- The DSS (digital signature standard, adopted by NIST in 94 is based on a modification of El-Gamal signature)

RSA

- Signature: code hash of message using private key
- Only the person who knows the secret key can sign
- Everybody can verify the signature using the public key

Instead of RSA we can use other Public Key cryptographic protocols

RSA: Public-Key Crypto. Standard (PKCS)

- Signature: code hash of message ("digest") using private key
- PKCS#1: standard encrypt using secret key
- 0||1||at least 8 byte FF base 16|| 0|| specification of used hash function || hash(M)
- (M message to be signed)
  - first byte 0 implies encoded message is less than n
  - second byte (=1) denotes signature (=2 encoding)
  - bytes 11111111 imply encoded message is large
  - specification of used hash function increases security

PKCS#1 2.2 for dig. sig.

- RFC 8017, 2016
- defining a Signature Scheme with Appendix (SSA)
  - the appendix is the signature, added to the message
  - not considering *signature schemes with message recovery* (message is embedded in the signature)

signing schemes

two approaches that differ in how the encoded message is obtained

- RSASSA-PSS (probabilistic signature scheme, new)
  - EMSA-PSS: encoding method for signature appendix, probabilistic signature scheme
    - inspired to OAEP
    - makes use of random salt added per signature
- RSASSA-PKCS1-v1_5 (old, for compatibility)
  - deterministic, uses EMSA-PKCS1-v1_5 (see previous slides)

EMSA-PSS

```
                             +-----------+
                             |     M     |
                             +-----------+
                                   |
                                   V
                                 Hash
                                   |
                                   V
                     +--------+----------+----------+
                M' = |Padding1|  mHash   |   salt   |
                     +--------+----------+----------+
                                    |
           +--------+----------+    V
     DB =  |Padding2|   salt   |   Hash
           +--------+----------+    |
                    |               |
                    V               |
                   xor <--- MGF <---|
                    |               |
                    |               |
                    V               V
           +-------------------+----------+--+
     EM =  |      maskedDB     |     H    |bc|
           +-------------------+----------+--+
```

Figure 2: EMSA-PSS Encoding Operation

El-Gamal Signature Scheme [KPS § 6.4.4]

Generation

- Pick a prime p of length 1024 bits such that DL in Z_p^* is hard
- Let g be a generator of Z_p^*
- Pick x in $[2, p-2]$ at random
- Compute $y = g^x \bmod p$
- Public key: (p, g, y)
- Private key: x

El-Gamal Signature Scheme

Signing M [a per-message public/private key pair (r, k) is also generated]

- Hash: Let m = H(M)
- Pick k in [1, p-2] relatively prime to p-1 at random
- Compute $r = g^k \bmod p$
- Compute $s = (m-rx)k^{-1} \bmod (p-1)$ (***)
  - if s is zero, restart
- Output signature (r, s)

El-Gamal Signature Scheme

Verify M, r, s, p, k

- Compute $m = H(M)$
- Accept if $(0 < r < p) \wedge (0 < s < p-1) \wedge (y^r r^s = g^m) \bmod p$, else reject
- What's going on?
- By (***) $s = (m-rx)k^{-1} \bmod p-1$, so $sk + rx = m$. Now $r = g^k$ so $r^s = g^{ks}$, and $y = g^x$ so $y^r = g^{rx}$, implying $y^r r^s = g^m$

Digital Signature Standard (DSS)

- NIST, FIPS PUB 186
- DSS uses SHA as hash function and DSA as signature
- DSA inspired by El Gamal

see [KPS § 6.5]

The Digital Signature Algorithm (DSA)

- Let p be an L bit prime such that the discrete log problem mod p is intractable
- Let q be a 160 bit prime that divides $p - 1$: $p = j \cdot q + 1$
- Let α be a q-th root of 1 modulo p: $\alpha = 1^{1/q} \bmod p$, or $\alpha^q = 1 \bmod p$

How do we compute α?

computing α

- take a random number h
  s.t. $1 < h < p - 1$ and compute
  $g = h^{(p-1)/q} \bmod p = h^j \bmod p$
- if $g = 1$ try a different h
  - things would be insecure
- it holds $g^q = h^{p-1}$
- by Fermat's theorem $h^{p-1} = 1 \bmod p$
  - p is prime
- choose $\alpha = g$

The Digital Signature Algorithm (DSA)

p prime, q prime, $p - 1 = 0 \bmod q$, $\alpha = 1^{(1/q)} \bmod p$

Private key: secret s, random $1 \leq s \leq q$-1.

Public key: $(p, q, \alpha, y = \alpha^s \bmod p)$

Signature on message M:

Choose a random $1 \leq k \leq q-1$, secret!!

Part I: $P_I = (\alpha^k \bmod p) \bmod q$

Part II: $P_{II} = (\mathrm{SHA}(M) + s\,(P_I))\, k^{-1} \bmod q$

Signature (P_I, P_{II})

*Note that P_I **does not** depend on M (preprocessing)*

P_{II} is fast to compute

The Digital Signature Algorithm (DSA)

p prime, q prime, $p - 1 = 0 \bmod q$, $\alpha = 1^{(1/q)} \bmod p$, Private key: random $1 \leq s \leq q$-1. Public key: (p, q, α, $y = \alpha^s \bmod p$). Signature on message M:

Choose a random $1 \leq k \leq q$-1, secret!!

P_I: $(\alpha^k \bmod p) \bmod q$

P_{II}: $(\mathrm{SHA}(M) + s\,P_I)\,k^{-1} \bmod q$

Verification:

$e_1 = \mathrm{SHA}(M)\,(P_{II})^{-1} \bmod q$

$e_2 = P_I\,(P_{II})^{-1} \bmod q$

ACCEPT Signature if

$(\alpha^{e1}\,y^{e2} \bmod p) \bmod q = P_I$

Digital Signature-correctness

Accept if $(\alpha^{e1} y^{e2} \bmod p) \bmod q = P_I$

$e1 = SHA(M) / P_{II} \bmod q$

$e2 = P_I / P_{II} \bmod q$

Proof : 1. definition of P_I and P_{II} implies

$SHA(M) = (-s P_I + k P_{II}) \bmod q$ hence

$SHA(M)/P_{II} + s P_I/P_{II} = k \bmod q$

2. *Definit. of* $y = \alpha^s \bmod p$ implies $\alpha^{e1} y^{e2} \bmod p = \alpha^{e1} \alpha^{(s\,e2)} \bmod p$

$= \alpha^{SHA(M)/P_{II} + s P_I / P_{II} \bmod q} \bmod p = \alpha^{k + cq} \bmod p$

$= \alpha^k \bmod p$ (since $\alpha^q = 1$).

3. Execution of mod q implies

$(\alpha^{e1} y^{e2} \bmod p) \bmod q = (\alpha^k \bmod p) \bmod q = P_I$

DSS: security [KPS § 6.5.5]

Secret key s is not revealed and it cannot be forged without knowing it

Use of a random number for signing- not revealed (k)

- There are no duplicates of the same signature (even if same messages)
- If k is known then you can compute $s \bmod q = s$ (s is chosen < q)
  - make s explicit from PART II
- Two messages signed with same k can reveal the value k *and therefore* $s \bmod q$
  - 2 equations (Part II and Part II'), 2 unknowns (s and k)

There exist other sophisticated attacks depending on implementation

if adversary knows k...

$P_{II} = (\mathrm{SHA}(M) + s\,P_I)\;k^{-1} \bmod q$

$P_{II}\,k = (\mathrm{SHA}(M) + s\,P_I) \bmod q$

$(P_{II}\,k - \mathrm{SHA}(M))\;P_I^{-1} = s \bmod q = s$ (since $s < q$)

then adv knows s

now adv. wants to sign M'

- $P_I = (\alpha^k \bmod p) \bmod q$ (independent on M')
- $P_{II} = ((\mathrm{SHA}(M') + s\,P_I)\;k^{-1}) \bmod q$

DSS: efficiency

- Finding two primes p and q such that $p - 1 = 0 \bmod q$ is not easy and takes time
- p and q are public: they can be used by many persons

- DSS slower than RSA in signature verification
- DSS and RSA same speed for signing (DSS faster if you use preprocessing)
- DSS requires random numbers: not always easy to generate

DSS versus RSA

DSS: (+) faster than RSA for signing (preprocessing- suitable for smart card)

(+?) uses random numbers to sign (+)

Implementation problems:

- **To generate random numbers you need special hardware (no smartcard);**
- **pseudo random generator requires memory (no smart card)**
- **Random number depending by messages does not allow preprocessing and slow the process**

(+) standard RSA: (+) known since many years and studied - no attacks

(+) faster in signature verification

DSA vs RSA

DSA: signature only

RSA: signature + key management

DH: Key management

DSA: patent free (RSA patented until 2000)

DSA: short signatures (RSA 5 times longer: 40 vs 200 bytes)

DSA faster

Timestamping a document

TimeStamping Authority (TSA): guarantees timestamp of a document

Alice (A) wants to timestamp a document

1. A compute hash of document and sends to TSA
2. TSA adds timestamp, computes new hash (of timestamp and received hash) and SIGNS the obtained hash; sends back to A
3. A keeps TSA's signature as a proof

- Everybody can check the signature
- TSA does not know Alice's document