Cryptography or Smalltalkers 2
Public Key Cryptography

Martin Kobetic
Cincom Smalltalk Development
ESUG 2006
Public Key Algorithms

Encryption (RSA)

Key Establishment (RSA, DH)

Signing
  - Hashes (SHA, MD5)
  - MACs (HMAC, CBC-MAC)
  - Digital Signatures (RSA, DSA)
Public Key Algorithms

- public and private key
  - hard to compute private from the public
  - sparse key space => much longer keys
- based on “hard” problems
  - factoring, discrete logarithm
- much slower
- RSA, DSA, DH, ElGamal
- elliptic curves: ECDSA, ECDH, …
provides:
- confidentiality

symmetric (secret) key ciphers
- same (secret) key => encrypt and decrypt
  - DES, AES, RC4

asymmetric (public) key ciphers
- public key => encrypt
- private key => decrypt
  - RSA, ElGammal
RSA (1977)

- RSA Security, PKCS #1
  modulus \( n \) = product of 2 large primes \( p, q \)
  public: \( e \) = relatively prime to \((p-1)(q-1)\)
  private: \( d = e^{-1} \mod ((p-1)(q-1)) \)
  \( C = P^e \mod n \) [ \( P < n \) ]
  \( P = C^d \mod n \)
- small \( e \) \( \Rightarrow \) faster encryption
keys := RSAKeyGenerator keySize: 512.
alice := RSA new publicKey: keys publicKey.
ctxt := alice encrypt: 'Hello World' asByteArray.
ctxt asHexString

bob := RSA new privateKey: keys privateKey.
(bob decrypt: ctxt) asString
keys := RSAKeyGenerator keySize: 512.
alice := RSA new publicKey: keys publicKey.
msg := 'Hello World' asByteArrayEncoding: #utf8.
msg := alice encrypt: msg.
bob := RSA new privateKey: keys privateKey.
msg := bob decrypt: msg.
msg asStringEncoding: #utf8
Key Establishment

- public key too slow for bulk encryption
  - public key => secure symmetric key
  - symmetric key => bulk encryption
- key exchange (RSA)
  - generate one-time symmetric key
  - public key => encrypt the symmetric key
- key agreement (DH)
  - parties cooperate to generate a shared secret
key := DSSRandom default byteStream next: 40.
msg := 'Hello World!' asByteArray.
msg := (ARC4 key: key) encrypt: msg.
alice := RSA new publicKey: keys publicKey.
key := alice encrypt: key.
bob := RSA new privateKey: keys privateKey.
key := bob decrypt: key
((ARC4 key: key) decrypt: msg) asString.
Diffie-Hellman (1976)

- shared secret over unprotected channel
- \url{http://www.ietf.org/rfc/rfc2631.txt}

  modulus \( p \): large prime \((\geq 512b)\)
  order \( q \): large prime \((\geq 160b)\)
  generator \( g \): order \( q \) mod \( p \)
  private \( x \): random \(1 < x < q - 1\)
  public \( y \): \( g^x \) mod \( p \)
  public \( y' \): other party’s \( y = g^{x'} \) (mod \( p \))
  shared secret: \( y'^x = y^x \) (mod \( p \))
Diffie-Hellman (interactive)

```
gen := DHParameterGenerator m: 160 l: 512.
alice := DH p: gen p q: gen q g: gen g.
ya := alice publicValue.

bob := DH p: alice p q: alice q g: alice g.
yb := bob publicValue.
ss := bob sharedSecretUsing: ya

ss = (alice sharedSecretUsing: yb)
```
bob := DH newFrom: gen.
yb := bob publicValue.

alice := DH newFrom: gen.
ya := alice publicValue.
ss := (alice sharedSecretUsing: yb) asByteArray.
msg := 'Hello World!' asByteArray.
msg := (ARC4 key: ss) encrypt: msg.

ss := (bob sharedSecretUsing: ya) asByteArray.
((ARC4 key: ss) decrypt: msg) asString.
Signing

- Provides:
  - integrity (tamper evidence)
  - authentication
  - non-repudiation
- Hashes (SHA, MD5)
- Digital Signatures (RSA, DSA)
Hash Functions

- provides:
  - data “fingerprinting”

- unlimited input size => fixed output size

- must be:
  - one-way: \( h(m) \Rightarrow m \)
  - collision resistant: \( m1,m2 \Rightarrow h(m1) = h(m2) \)

- MD2, MD4, MD5, SHA, RIPE-MD
Hash Functions

- compression function:
  \[ M = M_1, M_2, \ldots \]
  \[ h_i = f(M_i, h_{i-1}) \]

- MD-strengthening:
  - include message length (in the padding)
  - doesn’t completely prevent “length extension”
MD5 (1992)

  - digest: 128-bits (16B)
  - block: 512-bits (64B)
  - padding: M | 10...0 | length (64bits)

- broken in 2004, avoid MD5!
MD5

(MD5 hash: 'Hello' asByteArray) asHexString

(MD5 hash: #[1 2 3 4 5] from: 2 to: 4) asHexString

input := #[1 2 3 4 5 6 7 8 9] readStream.
(MD5 hashNext: 3 from: input) asHexString

(MD5 hashFrom: input) asHexString
SHA (1993)

- SHS - NIST FIPS PUB 180
  - digest: 160 bits (20B)
  - block: 512 bits (64B)
  - padding: M | 10...0 | length (64bits)


- SHA-1 broken in 2005!
input := 'Hello World!' asByteArray readStream.
sha := SHA new.
sha updateWithNext: 5 from: input.
sha digest asHexString.

sha updateFrom: input.
sha digest asHexString.

input reset.
(SHA256 hashFrom: input) asHexString.
Digital Signatures

- authentic, non-reusable, unalterable
- signing
  - uses the **private** key
  - message, key => signature
- verification
  - uses the **public** key
  - message, key, signature => true/false
RSA

- signing:
  - hash the plaintext
  - encode digest
  - encrypt digest with private key

- verifying:
  - decrypt digest with public key
  - decode digest
  - hash the plaintext
  - compare the digests
alice := RSA new privateKey: keys privateKey.
msg := 'Hello World' asByteArray.
sig := alice sign: msg.
sig asHexString

bob := RSA new publicKey: keys publicKey.
bob verify: sig of: msg
NIST FIPS PUB 186
- $p$ prime (modulus): $(512 + k \times 64 \leq 1024)$
- $q$ prime factor of $p - 1$ (160 bits)
- $g > 1; g^q \mod p = 1$ (g has order $q \mod p$)
- $x < q$ (private key)
- $y = g^x \mod p$ (public key)

FIPS 186-3 (?2006): bigger keys up to 15K bits
keys := DSAKeyGenerator keySize: 512.
alice := DSA new privateKey: keys privateKey.
sig := alice sign: 'Hello World' asByteArray

bob := DSA new publicKey: keys publicKey.
bob verify: sig of: 'Hello World' asByteArray
Books

- [2] Ferguson, Schneier: Practical Cryptography