

# Digital Signatures and Authentication

# Outline

- What is a digital signature ?
- General model
- Foundations of security
- RSA, DSA, ECDSA signatures
- Zero knowledge (Guillo-Quisquater)
- One-time signature
- Special signatures
- Message Authentication Codes
- Conclusion

# What is a digital signature ?

- Cryptographic message enhancement that
  - identifies signer
  - authenticates message - every bit
  - anyone can verify, but only signer can apply
- Stronger than authentication, which may involve two parties (e.g., Kerberos)
  - nonrepudiation

# General Model

- Extension of trapdoor public-key cryptography model
- Signature with private key
  - message, private key  $\rightarrow$  signature
  - may be randomized
  - hard without private key
- Verification with public key
  - message, public key, signature  $\rightarrow$  “valid” or “invalid”

# General Model (cont'd)

- Message recovery
  - the message can be recovered from the signature during verification
  - signature, public key  $\rightarrow$  message, “valid” or “invalid”
- Reversibility
  - the signature capability can be “reversed” to provide encryption
- These two properties are independent

# Foundations of Security

- Finding private key vs. forgery
- Can forgery be proved as hard as finding private key ?
- RSA :
  - finding private key as hard as factoring
  - forgery as hard as root extraction
  - forgery may or may not be as hard as factoring

# Practical issues

- Hybrid cryptography
  - digital signatures and one-way hash functions
  - message “digested” under hash function for speed
  - digest signed with digital signature for convenience

# Example 1: RSA

- R. Rivest, A. Shamir, L. Adleman (1977, pub 1978)
- Based on factoring / root extraction
- Moderate speed, high security
  - verification high speed
- Finding private key as hard as factoring
- Forgery may or may not be as hard

# RSA (cont'd)

- Public key :  $n, e$
- Private key :  $d$

where

- $n$  is a composite integer (*modulus*)
- $e$  is an integer (*public exponent*)
- $d$  is an integer (*private exponent*)

such that

$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$

where  $p, q$  are prime factors of  $n$

# RSA (cont'd)

- Signature :

$$s = m^d \bmod n$$

where

- $m$  is message
  - $s$  is signature
  - $d$  is private key
- Verification:

$$m = s^e \bmod n$$

# Multiplicative property of RSA

- If

$$s_1 = m_1^d \pmod n \quad \text{and}$$
$$s_2 = m_2^d \pmod n \quad \text{then}$$

$s = s_1 \cdot s_2 \pmod n$  is a valid signature for message  $m = m_1 \cdot m_2$  since

$$s = (m_1 \cdot m_2)^d \pmod n$$

- This allows a forgery

# Example 2: DSA

- NIST (1991) : Digital Signature algorithm
  - part of Digital Signature Standard (FIPS 186, 1994)
- Based on discrete logarithms; variants of ElGamal, Schnorr schemes
- Moderate speed
  - signature high speed with some precomputation
- Finding private key as hard as discrete logarithms
- Forgery may or may not be as hard

# DSA (cont'd)

- System parameters:  $p, q, g$ 
  - $p$  is prime
  - $q$  is a prime dividing  $p - 1$
  - $g$  generates a set of  $q$  elements  $g^x \bmod p$
- Public key:  $y$
- Private key:  $x$

where

- $x$  is integer
- $y$  is integer defined as  $y = g^x \bmod p$

# DSA (cont'd)

- Signature:

$$r = (g^k \bmod p) \bmod q$$

$$s = (m + x \cdot r)k^{-1} \bmod q$$

where

- $m$  is message
- $(r,s)$  is signature
- $k$  is a random integer
- $x$  is private key

# DSA (cont'd)

- Verification

$$r \stackrel{?}{=} (g^{m \cdot w} \cdot y^{r \cdot w} \bmod p) \bmod q$$

where

- $w = s^{-1} \pmod{q}$
- $y$  is public key

# Example 3: ECDSA

- The Elliptic Curve Digital Signature Algorithm (ECDSA) is being proposed as an ANSI X9.62 standard
- Like DSA based on ElGamal signature scheme
- Better than DSA
- With much smaller key length it provides same level of security as those of RSA and DSA
- Speed can be optimized

# ECDSA (cont'd)

- Public keys:  $(E, P, n, Q)$
- Private keys:  $d$

where

- $E$  is an Elliptic Curve
- $P$  is a point on the curve whose order is  $n$
- $d$  is an integer randomly selected in the interval  $[1, n-1]$
- $Q$  is another point on the curve such that

$$Q = d \cdot P$$

# ECDSA (cont'd)

- Signature:

$$k \cdot P = (x_1, y_1) \text{ and } r = x_1 \bmod n$$

$$s = k^{-1} \cdot [h(m) + d \cdot r] \bmod n$$

where

- $h(m)$  is Secure Hash of the message  $m$  (SHA-1)
- $k$  is a random integer in the interval  $[1, n-1]$
- $(r, s)$  is signature
- $(x_1, y_1)$  is components of an EC point (integers)

# ECDSA (cont'd)

- Verification:

$$u_1 = h(m) \cdot w \bmod n \quad \text{and} \quad u_2 = r \cdot w \bmod n$$

$$u_1 \cdot P + u_2 \cdot Q = (x_0, y_0) \quad \text{and} \quad v = x_0 \bmod n$$

?

$$v = r$$

- where

$$- w = s^{-1} \pmod{n}$$

# Comparison

- Security
  - They provide same security level with different key lengths.
  - DSA and ECDSA are less examined than RSA
- Implementation
  - Signature speeds are comparable, DSA is faster with precomputation
  - An elliptic curve with a point whose order 160 offers approximately the same level of security as DSA with a 1024-bit modulus  $p$  and RSA with a 1024-bit modulus  $n$

# Comparison (cont'd)

- Implementation (cont'd)
  - Underlying field and a representation for its elements can be selected so that the implementation speed can be optimized
  - ECDSA offers low cost implementations in restricted computing environments such as smart cards and wireless devices.

# Zero-knowledge

- Based on interactive proofs
  - Alice proves she knows something
  - Bob verifies
  - challenge-response protocol
- No transferable knowledge in transcript
  - Bob learns nothing about what Alice knows
  - he cannot convince anyone else
- For signatures, replace Bob with one-way hash function

# Guillou-Quisquater scheme

- L. Guillou, J.-J. Quisquater (1988)
- Based on factoring, zero knowledge; improvement on Fiat-Schamir scheme
- Moderate speed (faster than RSA) *provable* security
- Finding private key as hard as root extraction
- Forgery provably as hard, assuming good hash function

# Guillou-Quisquater (cont'd)

- Public key:  $n, e, I$
- Private key:  $S$

where

- $n$  is a composite modulus
- $e$  is an integer (*exponent*)
- $I, S$  are integers such that

$$I = S^e \bmod n$$

# Guillou-Quisquater (cont'd)

- Signature:

$$x = r^e \bmod n$$

$$c = h(m, x)$$

$$y = r \cdot S^c \bmod n$$

where

- $m$  is message,  $(x, y)$  is signature
- $r$  is a random integer
- $S$  is private key

# Guillou-Quisquater (cont'd)

- Verification:

$$c = h(m, x)$$
$$y^e = x \cdot I^c \pmod n$$

where  $(n, e, I)$  is public key

- Alternative signature:  $(c, y)$
- Hash function  $h$  simulates the verifiers's challenges in the zero-knowledge interactive proof that the signer knows the private key  $S$

# Towards higher speeds

- Are there faster schemes?
- Alternatives:
  - tree signatures (Merkle (1987))
  - on-line/off-line signatures  
(Even-Goldreich-Micali (1989))
- Faster schemes often have longer signatures

# One-time signature schemes

- A mechanism which can be used to sign, at most, one message; otherwise, signatures can be forged
- A new public key is required for each message
- Public information (*validation parameters*) is necessary for verification
- Signature generation and verification are very efficient
- Useful in applications such as smart cards, where low computational complexity is required

# The Rabin one-time signature scheme

- One-time public key:
  - $(k_1, k_2, \dots, k_{2n})$ , each of bitlength  $l$
- One-time private key:
  - $(y_1, y_2, \dots, y_{2n})$ , each of bitlength  $l$

such that  $y_i = E_{k_i}(M_0(i))$ ,  $1 \leq i \leq 2 \cdot n$

where

- $E$  is a symmetric-key encryption scheme (e.g. DES)
- $M_0(i) = 0^{l-e} \parallel b_{e-1} \dots b_1 b_0$ ,  $b_{e-1} \dots b_1 b_0$  is the binary representation of  $i$ .

# The Rabin scheme (cont'd)

- Signature:

$$s_i = E_{k_i}(h(m)) \quad 1 \leq i \leq 2 \cdot n$$

where

- $m$  is message
- $(s_1, s_2, \dots, s_{2n})$  is signature
- $h$  is hash function
- $E$  is a symmetric-key encryption scheme (e.g. DES)

# The Rabin scheme (cont'd)

- Verification:
  - Select  $n$  distinct random numbers  $r_j$  such that  $1 \leq r_j \leq 2n$
  - Request the private keys  $k_{r_j}$ ,  $1 \leq j \leq n$
  - Verify the authenticity of key by checking  $z_j = y_{r_j}$  where  $z_j = E_{k_{r_j}}(M_0(r_j))$
  - Verify that  $s_{r_j} = E_{k_{r_j}}(h(m))$ ,  $1 \leq j \leq n$

# Special Signatures

- *Blind Signatures*
  - users sign or verify messages without learning the contents
  - blinded verification, blinded message or fully blind
  - verification of a *weak* blind signature requires the use of some third party or trusted center
  - In a banking application, a message  $m$  might represent a monetary value which a customer can spend. Bank signs the message without seeing the contents. This scheme help customers prevent their spending patterns from being monitored

# Special Signatures (cont'd)

- *Blind* Signatures (cont'd)
  - Chaum's blind signature protocol uses RSA
- *Undeniable* signatures
  - no public verification function
  - signature verification protocol requires the cooperation of the signer
  - convertible to ordinary signatures with signer's help
  - confirmation/disavowal protocols

# Special Signatures (cont'd)

- *Undeniable* signatures (cont'd)
  - Chaum-van Antwerpen undeniable signature scheme; based on discrete logarithm problem
- *Designated confirmer* signatures
  - designated confirmer acts independently of the signer
  - avoids problems when a user refuses to participate in verification

# Special Signatures (cont'd)

- *Unconditionally secure* signatures
  - impossible to forge, except by guessing
  - usually involve multiple verification functions

# Fail-Stop signatures

- E. van Heyst, T. Pedersen (1992)
- Based on discrete logarithms
- Moderate speed, high security, large signatures or secret key
- Finding private key unconditionally hard
- Forgery as hard as discrete logarithm
- Signer can prove forgery

# Fail-Stop signatures (cont'd)

- How it works:
  - many private keys yield a given signature, hard for forger to find right one
  - given new signature forged with different private key, signer can compute a certain discrete logarithm
  - assuming it is hard for signer to compute discrete logarithms, this is *proof* of forgery
- One time construction, can be extended

# Fail-Stop signatures (cont'd)

- Public key:  $p_1, p_2$
- Private key:  $x_1, x_2, y_1, y_2$ 
  - $p$  is a prime number
  - $q$  is a prime dividing  $p - 1$
  - $g$  and  $h$  are unrelated elements of order  $q \bmod p$
  - $x_1, x_2, y_1, y_2$  are integers
  - $p_1, p_2$  are integers defined as

$$p_1 = g^{x_1} \cdot h^{x_2} \bmod p \qquad p_2 = g^{y_1} \cdot h^{y_2} \bmod p$$

# Fail-Stop signatures (cont'd)

- Signature:

$$\mathbf{s}_1 = x_1 + m \cdot y_1 \bmod q$$

$$\mathbf{s}_2 = x_2 + m \cdot y_2 \bmod q$$

where

–  $m$  is message and  $(\mathbf{s}_1, \mathbf{s}_2)$  is signature

- Verification

$$p_1 \cdot p_2^m \stackrel{?}{=} g^{\mathbf{s}_1} \cdot h^{\mathbf{s}_2} \bmod p$$

# Fail-Stop signatures (cont'd)

- $q^2$  private keys for each public key
- $q$  private keys for each signature
- Given signature with different private key, signer can compute  $\log_g h$  :

proof of forgery

- $(\mathbf{s}_1, \mathbf{s}_2)$  and  $(\mathbf{s}'_1, \mathbf{s}'_2)$  both signatures on  $m$ :

$$g^{\mathbf{s}_1} \cdot h^{\mathbf{s}_2} = g^{\mathbf{s}'_1} \cdot h^{\mathbf{s}'_2} \pmod{p}$$

# Fail-Stop signatures (cont'd)

- Solve for  $\log_g h$

$$g^{s_1 - s'_1} = h^{s'_2 - s_2} \pmod p$$

$$g^{s_1 - s'_1 / s'_2 - s_2} = h \pmod p$$

# Message Authentication

- *Authenticator* is a value to be used to authenticate the integrity of a message
- Functions that produce an authenticator:
  - MAC (*Message Authentication Code* or *Cryptographic Checksum*):  
A public function of the message and a secret key that produces a fixed-length value
  - *Hash* function: A public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator

# MAC

- Shared secret key:  $k$
- Generation of MAC:  $MAC_k(m)$
- Sender transmits  $(m, MAC_k(m))$
- Message integrity check:

$$MAC_k(m') \stackrel{?}{=} MAC_k(m)$$

- where  $m'$  is received message
- No public verification method

# Examples of MACs

- CBC-MAC
  - block cipher based: DES, IDEA, etc.
  - compute a CBC encryption of message and output only the last block

$$MAC_k(m_1, m_2, m_3) = DES_k(DES_k(DES_k(m_1) \oplus m_2) \oplus m_3)$$

- Keyed-hash
  - one-way hash based: MD5, SHA1, etc.
  - $MAC_k(m) = MD5(k_1, m, k_2)$  (*envelope method*)

# Hash functions

- A hash function  $h$  maps bit strings of arbitrary finite length to strings of fixed length, say  $n$  bits
- One way functions
- Used in almost every digital signature scheme to generate a value of finite length so that it can be signed
- SHA-1 was proposed by NIST for certain US federal government applications; (160-bit)
- MD5 is used in commercial applications; (128-bit)

# Conclusion

- Many digital signature schemes
  - speed
  - security foundations
  - special properties
- Research improving each area