

INTRODUCTION TO PUBLIC-KEY CRYPTOGRAPHY

© *Alfred Menezes*

– 202

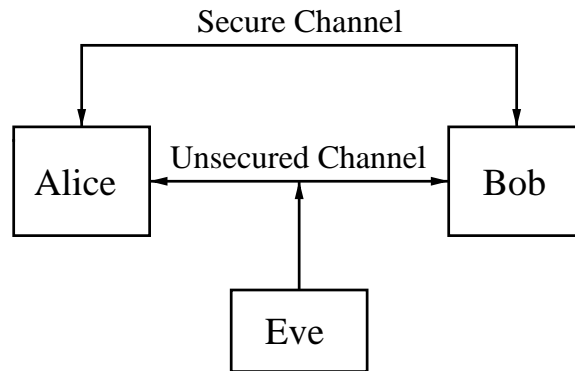
Outline

1. *Drawbacks with Symmetric-Key Cryptography*
 - (a) Key Establishment Problem
 - (b) Key Management Problem
 - (c) Non-Repudiation is Difficult to Achieve
2. *Public-Key Cryptography*
 - (a) Public-Key Encryption
 - (b) Digital Signatures
 - (c) Hybrid Schemes

– 203

Drawbacks with Symmetric-Key Cryptography

Symmetric-key cryptography: Communicating parties a priori share some *secret* keying information.



The shared secret keys can then be used to achieve confidentiality (e.g., using AES) and authentication (e.g., using HMAC).

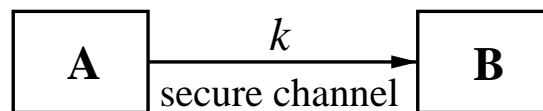
– 204

Key Establishment Problem

How do Alice and Bob establish the secret key k ?

Method 1: *Point-to-point key distribution*.

(Alice selects the key and sends it to Bob over a secure channel)



The secure channel could be:

- ▶ A trusted courier.
- ▶ A face-to-face meeting.
- ▶ Installation of an authentication key in a SIM card.

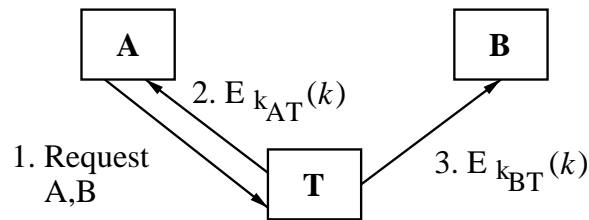
This is generally not practical for large-scale applications.

– 205

Key Establishment Problem (2)

Method 2: Use a Trusted Third Party (TTP) T .

- ▶ Each user A shares a secret key k_{AT} with T for a symmetric-key encryption scheme E .
- ▶ To establish this key, A must visit T once.
- ▶ T serves as a *key distribution centre* (KDC):



1. A sends T a request for a key to share with B .
2. T selects a session key k , and encrypts it for A using k_{AT} .
3. T encrypts k for B using k_{BT} .

– 206

Key Establishment Problem (3)

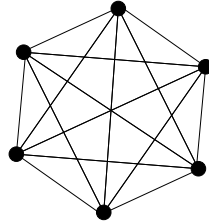
Drawbacks of using a KDC:

1. The TTP must be unconditionally trusted.
 - ▶ Makes it an attractive target.
2. Requirement for an on-line TTP.
 - ▶ Potential bottleneck.
 - ▶ Critical reliability point.

– 207

Key Management Problem

- ▶ In a network of n users, each user has to share a different key with every other user.



- ▶ Each user thus has to store $n - 1$ different secret keys.
- ▶ The total number of secret keys is $\binom{n}{2} \approx n^2/2$.

– 208

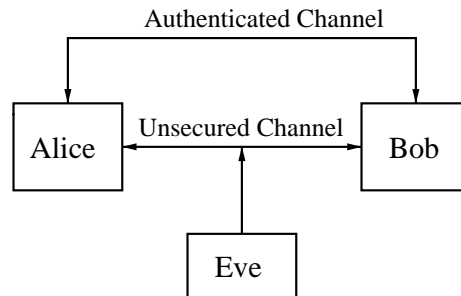
Non-Repudiation is Impractical

- ▶ *Non-repudiation*: Preventing an entity from denying previous actions or commitments.
 - Denying being the source of a message.
- ▶ Strictly speaking, symmetric-key techniques cannot be used to achieve non-repudiation.
 - Why?
- ▶ However, symmetric-key techniques can be used to achieve *some* degree of non-repudiation, but typically requires the services of an *on-line TTP* (e.g., use a MAC algorithm where each user shares a secret key with the TTP).

– 209

Public-Key Cryptography

- Public-key cryptography: Communicating parties a priori share some *authenticated* (but non-secret) information.



- Invented by Ralph Merkle, Whitfield Diffie, Martin Hellman in 1975.

– 210

Merkle-Diffie-Hellman



Ralph Merkle, Whit Diffie, Martin Hellman

– 211

Ralph Merkle (1974)

Excerpts from Merkle's CS 244 project proposal
(Computer Security, UC Berkeley, Fall 1974)

“Secure communications are made possible because of knowledge, known to both people, which is not known to anyone else. Usually, both people know this knowledge because they were able to hold a private conversation with each other before they began to send encrypted messages over an unsecure channel.”

“It might seem intuitively obvious that if two people have never had the opportunity to prearrange an encryption method, then they will be unable to communicate securely over an insecure channel. While this might seem intuitively obvious, I believe it is false. I believe that it is possible for two people to communicate securely without having made any prior arrangements that are not completely public.”

– 212

Merkle Puzzles

Goal: Alice and Bob establish a secret session key by communicating over an authenticated (but non-secret) channel.

1. Alice creates N puzzles P_i , $1 \leq i \leq N$ (e.g., $N = 10^9$). Each puzzle takes t hours to solve (e.g., $t = 5$). The solution to P_i reveals a 128-bit *session key* sk_i and a randomly-selected 128-bit *serial number* n_i .
2. Alice sends P_1, P_2, \dots, P_N to Bob.
3. Bob selects j at random from $[1, N]$ and solves puzzle P_j to obtain sk_j and n_j .
4. Bob sends n_j to Alice.
5. The secret session key is sk_j .

An eavesdropper has to solve 500,000,000 puzzle on average to determine the puzzle index j (and thus sk_j).

– 213

Merkle Puzzles (2)

Example of a Merkle puzzle:

$P_i = \text{AES-CBC}_{k_i}(sk_i, n_i, n_i)$, where $k_i = (r_i \parallel 0^{88})$ and r_i is a randomly selected 40-bit string.

P_i can be solved in 2^{40} steps by exhaustive key search.

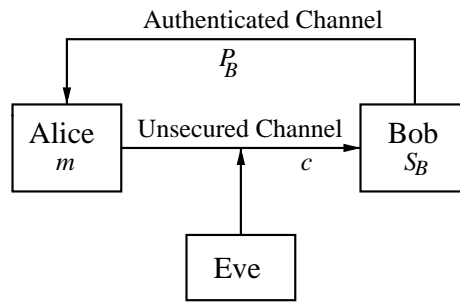
– 214

Key Pair Generation for Public-Key Crypto

- ▶ Each entity A does the following:
 1. Generate a key pair (P_A, S_A) .
 2. S_A is A 's *secret key*.
 3. P_A is A 's *public key*.
- ▶ Security requirement: It should be infeasible for an adversary to recover S_A from P_A .
- ▶ Example: $S_A = (p, q)$ where p and q are randomly-selected prime numbers;
 $P_A = p \cdot q$.

– 215

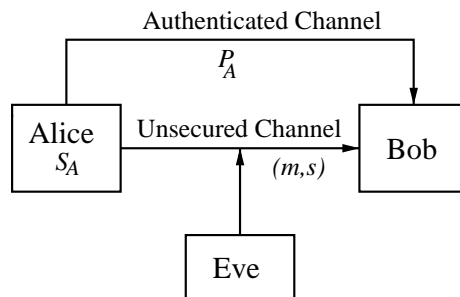
Public-Key Encryption



- To *encrypt* a secret message m for Bob, Alice does:
 1. Obtain an *authentic* copy of Bob's public key P_B .
 2. Compute $c = E(P_B, m)$; E is the encryption function.
 3. Send c to Bob.
- To *decrypt* c , Bob does:
 1. Compute $m = D(S_B, c)$; D is the decryption function.

– 216

Digital Signatures



- To *sign* a message m , Alice does:
 1. Compute $s = \text{Sign}(S_A, m)$.
 2. Send m and s to Bob.
- To *verify* Alice's signature s on m , Bob does:
 1. Obtain an *authentic* copy of Alice's public key P_A .
 2. Accept if $\text{Verify}(P_A, m, s) = \text{Accept}$.

– 217

Digital Signatures (2)

- ▶ Suppose that Alice generates a signed message (m, s) .
- ▶ Then *anyone* who has an authentic copy of Alice's public key P_A can verify the authenticity of the signed message.
 - This authentication property cannot be achieved with a symmetric-key MAC scheme.
- ▶ Digital signatures are widely used to sign software updates which are then broadcast to computers around the world.

– 218

Public-Key Versus Symmetric-Key

Advantages of public-key cryptography:

- ▶ No requirement for a secured channel.
- ▶ Each user has only 1 key pair, which simplifies key management.
- ▶ A signed message can be verified by anyone.
- ▶ Facilitates the provision of non-repudiation services (with digital signatures).

Disadvantages of public-key cryptography:

- ▶ Public keys are typically larger than symmetric keys.
- ▶ Public-key schemes are slower than their symmetric-key counterparts.

– 219

Hybrid Schemes

In practice, symmetric-key and public-key schemes are used together. Here is an example:

To *encrypt a secret signed message* m , Alice does:

1. Compute $s = \text{Sign}(S_A, m)$.
2. Select a secret key k for a symmetric-key encryption scheme such as AES.
3. Obtain an authentic copy of Bob's public key P_B .
4. Send $c_1 = E(P_B, k)$ and $c_2 = \text{AES}_k(m, s)$.

To *recover m and verify its authenticity*, Bob does:

1. Decrypt c_1 : $k = D(S_B, c_1)$.
2. Decrypt c_2 using k to obtain (m, s) .
3. Obtain an authentic copy of Alice's public key P_A .
4. Verify Alice's signature s on m .

– 220

ALGORITHMIC NUMBER THEORY

©Alfred Menezes

– 221

Outline

1. Fundamental Theorem of Arithmetic
2. Basic Concepts from Complexity Theory
3. Basic Integer Operations
4. Basic Modular Operations

– 222

Fundamental Theorem of Arithmetic

Theorem: Every integer $n \geq 2$ has a unique prime factorization (up to ordering of factors).

Interesting questions:

- ▶ Given an integer n , how do we find its prime factorization *efficiently*?
- ▶ How do we *efficiently* verify an alleged prime factorization of an integer n ?

– 223

Basic Concepts from Complexity Theory

- ▶ An *algorithm* is a “well-defined computational procedure” (e.g., a Turing machine) that takes a variable input and eventually halts with some output.
 - For an integer factorization algorithm, the *input* is a positive integer n , and the output is the prime factorization of n .
- ▶ The *efficiency* of an algorithm is measured by the scarce resources it consumes (e.g. time, space, number of processors).
- ▶ The *input size* is the number of bits required to write down the input using a reasonable encoding.
 - The *size* of a positive integer n is $\lfloor \log_2 n \rfloor + 1$ bits.

– 224

Basic Concepts from Complexity Theory (2)

- ▶ The *running time* of an algorithm is an upper bound as a function of the input size, of the worst case number of basic steps the algorithm takes over all inputs of a fixed size.
- ▶ An algorithm is a *polynomial-time* (efficient) algorithm if its (expected) running time is $O(k^c)$, where c is a fixed positive integer, and k is the *input size*.
- ▶ Recall that if $f(n)$ and $g(n)$ are functions from the positive integers to the positive real numbers, then $f(n) = O(g(n))$ means that there exists a positive constant c and a positive integer n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
 - For example, $7.5n^3 + 1000n^2 - 99 = O(n^3)$.

– 225

Basic Integer Operations

Input: Two k -bit positive integers a and b .

Input size: $O(k)$ bits.

Operation	Running time of naive algorithm (in bit operations)
Addition: $a + b$	$O(k)$
Subtraction: $a - b$	$O(k)$
Multiplication: $a \cdot b$	$O(k^2)$
Division: $a = qb + r$	$O(k^2)$
GCD: $\gcd(a, b)$	$O(k^2)$

– 226

Basic Modular Operations

Input: A k -bit integer n , and integers $a, b, m \in [0, n - 1]$.

Input size: $O(k)$ bits.

Operation	Running time of naive algorithm (in bit operations)
Addition: $a + b \bmod n$	$O(k)$
Subtraction: $a - b \bmod n$	$O(k)$
Multiplication: $a \cdot b \bmod n$	$O(k^2)$
Inversion: $a^{-1} \bmod n$	$O(k^2)$
Exponentiation: $a^m \bmod n$	$O(k^3)$

– 227

Modular Exponentiation

Input: A k -bit integer n , and integers $a, m \in [1, n - 1]$.

Output: $a^m \bmod n$.

► Naive algorithm 1:

 Compute $d = a^m$.

 Return($d \bmod n$).

► Naive algorithm 2:

$A \leftarrow a$

 For i from 2 to m do: $A \leftarrow A \times a \bmod n$.

 Return(A).

– 228

Modular Exponentiation

Repeated square-and-multiply algorithm

Let the binary representation of m be $m = \sum_{i=0}^{k-1} m_i 2^i$. Then

$$a^m \equiv a^{\sum_{i=0}^{k-1} m_i 2^i} \equiv \prod_{i=0}^{k-1} a^{m_i 2^i} \equiv \prod_{\substack{0 \leq i \leq k-1 \\ m_i = 1}} a^{2^i} \pmod{n}.$$

This suggests the following algorithm for computing $a^m \bmod n$:

 Write m in binary: $m = \sum_{i=0}^{k-1} m_i 2^i$.

 If $m_0 = 1$ then $B \leftarrow a$; else $B \leftarrow 1$.

$A \leftarrow a$.

 For i from 1 to $k - 1$ do:

$A \leftarrow A^2 \bmod n$.

 If $m_i = 1$ then $B \leftarrow B \times A \bmod n$.

 Return(B).

Analysis: At most k modular squarings and k modular multiplications, so worst-case running time is $O(k^3)$ bit operations.

– 229

RSA

© *Alfred Menezes*

– 218

Outline

1. RSA Public-Key Encryption Scheme
 - (a) Basic RSA Encryption Scheme
 - (b) RSA PKCS #1 v1.5 Encryption Scheme
 - (c) RSA-OAEP
2. Status of Integer Factorization
3. RSA Signature Scheme
 - (a) Basic RSA Signature Scheme
 - (b) RSA-FDH
 - (c) RSA PKCS # v1.5 Signature Scheme

– 219

RSA



Ron Rivest, Adi Shamir, Len Adleman

– 220

RSA



Ron Rivest



Adi Shamir

– 221

RSA Public-Key Encryption



See class notes

– 222

Toy Example: RSA Key Generation

Alice does the following:

1. Selects primes $p = 23$ and $q = 37$.
2. Computes $n = pq = 851$ and $\phi(n) = (p - 1)(q - 1) = 792$.
3. Selects $e = 631$ satisfying $\gcd(631, 792) = 1$.
4. Solves $631d \equiv 1 \pmod{792}$ to get $d \equiv -305 \equiv 487 \pmod{792}$, and selects $d = 487$.
5. Alice's *public key* is $(n = 851, e = 631)$; her *private key* is $d = 487$.

– 223

Toy Example: RSA Encryption

To encrypt a *plaintext* $m = 13$ for Alice, Bob does:

1. Obtains Alice's public key ($n = 851$, $e = 631$).
2. Computes $c = 13^{631} \bmod 851$ using the repeated-square-and-multiply algorithm:
 - (a) Write $e = 631$ in binary:

$$e = 2^9 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0.$$

- (b) Compute successive squarings of $m = 13$ modulo n :

$13 \equiv 13 \pmod{851}$	$13^2 \equiv 169 \pmod{851}$
$13^{2^2} \equiv 478 \pmod{851}$	$13^{2^3} \equiv 416 \pmod{851}$
$13^{2^4} \equiv 303 \pmod{851}$	$13^{2^5} \equiv 752 \pmod{851}$
$13^{2^6} \equiv 440 \pmod{851}$	$13^{2^7} \equiv 423 \pmod{851}$
$13^{2^8} \equiv 219 \pmod{851}$	$13^{2^9} \equiv 305 \pmod{851}.$

– 224

Toy Example: RSA Encryption (2)

- (c) Multiply together the squares 13^{2^i} for which the i th bit (where $0 \leq i \leq 9$) of the binary representation of 631 is 1:

$$\begin{aligned} 13^{631} &= 13^{2^9+2^6+2^5+2^4+2^2+2^1+2^0} \\ &= 13^{2^9} \cdot 13^{2^6} \cdot 13^{2^5} \cdot 13^{2^4} \cdot 13^{2^2} \cdot 13^{2^1} \cdot 13^{2^0} \\ &\equiv 305 \cdot 440 \cdot 752 \cdot 303 \cdot 478 \cdot 169 \cdot 13 \pmod{851} \\ &\equiv 616 \pmod{851}. \end{aligned}$$

3. Bob sends the *ciphertext* $c = 616$ to Alice.

To decrypt $c = 616$, Alice uses her private key $d = 487$ as follows:

1. Compute $m = 616^{487} \bmod 851$ to get $m = 13$.

– 225

Status of Integer Factorization

1. Definitions from Complexity Theory
2. Special-Purpose Factoring Algorithms
3. General-Purpose Factoring Algorithms
4. History of Factoring

– 226

Big-O and Little-o Notation

- ▶ Let $f(n)$ and $g(n)$ be functions from the positive integers to the positive real numbers.
- ▶ (*Big-O notation*) We write $f(n) = O(g(n))$ if there exists a positive constant c and a positive integer n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
 - Example: $3n^3 + 4n^2 + 79 = O(n^3)$.
- ▶ (*Little-o notation*) We write $f(n) = o(g(n))$ if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

- Example: $\frac{1}{n} = o(1)$.

– 227

Measures of Running Time

- ▶ (*Polynomial-time algorithm*) One whose worst-case running time function is of the form $O(n^c)$, where n is the *input size* and c is a constant.
- ▶ (*Exponential-time algorithm*) One whose worst-case running time function is not of the form $O(n^c)$.
 - In this course, fully exponential-time functions are of the form 2^{cn} , where c is a constant.
 - (*Subexponential-time algorithm*) One whose worst-case running time function is of the form $2^{o(n)}$, and not of the form $O(n^c)$ for any constant c .

Roughly speaking, “polynomial-time = efficient”,
“fully exponential-time = terribly inefficient”,
“subexponential-time = inefficient but not terribly so”.

– 228

Example (Trial Division)

- ▶ Consider the following algorithm (trial division) for factoring RSA-moduli n .
- ▶ Trial divide n by the primes $2, 3, 5, 7, \dots, \lfloor \sqrt{n} \rfloor$. If any of these, say ℓ , divides n , then stop and output the factor ℓ of n .
- ▶ The running time of this method is at most \sqrt{n} trial divisions, which is $O(\sqrt{n})$.
- ▶ Is this a polynomial-time algorithm for factoring RSA moduli?

– 229

Subexponential Time

- ▶ Let A be an algorithm whose inputs are elements of the integers modulo n , \mathbb{Z}_n , or an integer n (so the input size is $O(\log n)$).
- ▶ If the expected running time of A is of the form

$$L_n[\alpha, c] = O\left(\exp\left((c + o(1))(\log n)^\alpha (\log \log n)^{1-\alpha}\right)\right),$$

where c is a positive constant, and α is a constant satisfying $0 < \alpha < 1$, then A is a subexponential-time algorithm.

Note ($\alpha = 0$): $L_n[0, c] = O((\log n)^{c+o(1)})$ (polytime).

Note ($\alpha = 1$): $L_n[1, c] = O(n^{c+o(1)})$ (fully exponential time).

– 230

Special-Purpose Factoring Algorithms

- ▶ Examples: Trial division, Pollard's p-1 algorithm, Pollard's ρ algorithm, elliptic curve factoring algorithm, special number field sieve.
- ▶ These are only efficient if the number n being factored has a *special form* (e.g., n has a prime factor p such that $p - 1$ has only small factors; or n has a prime factor p that is relatively small).
- ▶ To maximize resistance to these factoring attacks on RSA moduli, one should select the RSA primes p and q *at random and of the same bitlength*.

– 231

General-Purpose Factoring Algorithms

- ▶ These are factoring algorithms whose running times do not depend of any properties of the number being factored.
- ▶ Two major developments in the history of factoring:
 1. (1982) *Quadratic sieve factoring algorithm (QS)*:
Running time: $L_n[\frac{1}{2}, 1]$.
 2. (1990) *Number field sieve factoring algorithm (NFS)*:
Running time: $L_n[\frac{1}{3}, 1.923]$.

Recall:

$$L_n[\alpha, c] = O\left(\exp\left((c + o(1))(\log n)^\alpha (\log \log n)^{1-\alpha}\right)\right)$$

– 232

History of Factoring

Year	Number	Bits	Method	Notes
1903	$2^{67} - 1$	67	Naive	F. Cole (3 years of Sundays) In 2014: 0.01 secs in Maple
1988	$\approx 10^{100}$	332	QS	Distributed computation by 100's of computers; communication by email
1994	RSA-129	425	QS	1600 computers around the world; 8 months
1999	RSA-155	512	NFS	300 workstations + Cray; 5 months
2002	RSA-158	524	NFS	≈ 30 workstations + Cray; 3 months
2003	RSA-174	576	NFS	
2005	RSA-200	663	NFS	(55 years on a single workstation)
2009	RSA-768	768	NFS	2.5 years using several hundred processors

The RSA Factoring Challenge:

<http://tinyurl.com/RSAfactoring>

– 233

RSA-768

The largest 'hard' number factored to date is RSA-768 (230 decimal digits, 768 bits), which was factored in 2009.

12301866845301177551304949583849627207728535695953
34792197322452151726400507263657518745202199786469
38995647494277406384592519255732630345373154826850
79170261221429134616704292143116022212404792747377
94080665351419597459856902143413

=

33478071698956898786044169848212690817704794983713
76856891243138898288379387800228761471165253174308
7737814467999489

*

36746043666799590428244633799627952632279158164343
08764267603228381573966651127923337341714339681027
0092798736308917

– 234

RSA-1024

RSA-1024 factoring challenge (1024 bits, 309 decimal digits):

13506641086599522334960321627880596993888147560566
70275244851438515265106048595338339402871505719094
41798207282164471551373680419703964191743046496589
27425623934102086438320211037295872576235850964311
05640735015081875106765946292055636855294752135008
52879416377328533906109750544334999811150056977236
890927563

– 235

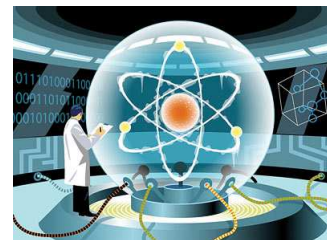
Equivalent Security Levels

Security in bits	Block cipher	Hash function	RSA $\ n\ _2$
80	SKIPJACK	(SHA-1)	1024
112	Triple-DES	SHA-224	2048
128	AES Small	SHA-256	3072
192	AES Medium	SHA-384	7680
256	AES Large	SHA-512	15360

– 236

Summary

- ▶ Factoring is *believed* to be a hard problem. However, we have no *proof* or *theoretical evidence* that factoring is indeed hard.
- ▶ However, factoring is *known* to be *easy* on a quantum computer (Shor's algorithm).
In December 2001, the number 15 was factored on a quantum computer by a team of IBM scientists.
The big open question is whether large-scale quantum computers can ever be built.
- ▶ 512-bit RSA is considered insecure today.
- ▶ 1024-bit RSA is considered secure today & widely deployed.
- ▶ Applications are moving to 2048-bit and 3072-bit RSA.



– 237

The RSA Signature Scheme

1. Basic RSA Signature Scheme
2. Security of the RSA Signature Scheme
 - (a) Hardness of RSAP
 - (b) Security Properties of the Hash Function
 - (c) Objectives of the Adversary
 - (d) Attack Model
 - (e) Security Definition
3. Full Domain Hash (FDH)
4. PKCS #1 v1.5 RSA Signature Scheme

– 238

Basic RSA Signature Scheme

Key generation: Each entity A does the following:

1. Randomly select 2 large distinct primes p and q of the same bitlength.
2. Compute $n = pq$ and $\phi(n) = (p - 1)(q - 1)$.
3. Select arbitrary e , $1 < e < \phi(n)$, such that $\gcd(e, \phi(n)) = 1$.
4. Compute d , $1 < d < \phi(n)$, such that $ed \equiv 1 \pmod{\phi(n)}$.
5. A 's *public key* is (n, e) ; A 's *private key* is d .

– 239

Signature Generation and Verification

Signature generation: To sign a message $m \in \{0, 1\}^*$, A does the following:

1. Compute $M = H(m)$, where H is a hash function.
2. Compute $s = M^d \bmod n$.
3. A 's signature on m is s .

Signature verification: To verify A 's signature s on m , B does the following:

1. Obtain an authentic copy of A 's public key (n, e) .
2. Compute $M = H(m)$.
3. Compute $M' = s^e \bmod n$.
4. Accept (m, s) if and only if $M = M'$.

– 240

Security of the Basic RSA Signature Scheme

Hardness of RSAP

We require that RSAP be intractable, since otherwise E could forge A 's signature as follows:

1. Select arbitrary m .
2. Compute $M = H(m)$.
3. Solve $s^e \equiv M \pmod{n}$ for s .
4. Then s is A 's signature on m .

– 241

Security Properties of the Hash Function

The following are desired security properties of H :

Preimage resistance: If H is not preimage resistant, and the range of H is $[0, n - 1]$, E can forge signatures as follows:

1. Select arbitrary $s \in [0, n - 1]$.
2. Compute $M = s^e \bmod n$.
3. Find m such that $H(m) = M$.
4. Then s is A 's signature on m .

2nd preimage resistance: If H is not 2nd preimage resistant, E could forge signatures as follows:

1. Suppose that (m, s) is a valid signed message.
2. Find an m' , $m \neq m'$, such that $H(m) = H(m')$.
3. Then (m', s) is a valid signed message.

– 242

Security Properties of the Hash Function (2)

Collision resistance: If H is not collision resistant, E could forge signatures as follows:

1. Select m_1, m_2 such that $H(m_1) = H(m_2)$, where m_1 and m_2 are two distinct messages.
2. Induce A to sign m_1 : $s = H(m_1)^d \bmod n$.
3. Then s is also A 's signature on m_2 .

– 243

Goals of the Adversary

1. *Total break*: E recovers A 's private key, or a method for systematically forging A 's signatures (i.e., E can compute A 's signature for arbitrary messages).
2. *Selective forgery*: E forges A 's signature for a selected subset of messages.
3. *Existential forgery*: E forges A 's signature for a single message; E may not have any control over the content or structure of this message.

– 244

Attack Model

Types of attacks E can launch:

1. *Key-only attack*: The only information E has is A 's public key.
2. *Known-message attack*: E knows some message/signature pairs.
3. *Chosen-message attack*: E has access to a signing oracle which it can use to obtain A 's signatures on some messages of its choosing.

– 245

Security Definition

Definition: A signature scheme is said to be *secure* if it is existentially unforgeable by a computationally bounded adversary who launches a chosen-message attack.

Note: The adversary has access to a signing oracle. Its goal is to compute a single valid message/signature pair for any message that was not previously given to the signing oracle.

Question: Is the basic RSA signature scheme secure?

Answer:

NO, if H is SHA-256; [Details not covered in this course.]

YES if H is a ‘full domain’ hash function.

– 246

Full Domain Hash RSA (RSA-FDH)

- ▶ Same as the basic RSA signature scheme, except that the hash function is $H : \{0, 1\}^* \longrightarrow [0, n - 1]$.

- ▶ In practice, one could use:

$$H(m) =$$

$$\text{SHA-256}(0, m) \parallel \text{SHA-256}(1, m) \parallel \dots \parallel \text{SHA-256}(t, m).$$

- ▶ Theorem (Bellare & Rogaway, 1996):

If RSAP is intractable and H is a random function, then RSA-FDH is a secure signature scheme.

– 247

ELECTRONIC CASH

© *Alfred Menezes*

– 248

Outline

1. What is Electronic Cash?
2. Simplified Electronic Cash Protocols
3. RSA Blind Signatures
4. On-Line Electronic Cash Protocol
5. Off-Line Electronic Cash Protocol
6. Miscellaneous Notes

– 249

David Chaum



David Chaum

– 250

What is Electronic Cash?

- ▶ *Electronic cash* is an electronic payment system modeled after our paper cash system.
- ▶ Some features of paper cash:
 - *Recognizable* (as legal tender)
 - *Portable* (easily carried)
 - *Transferable* (without involvement of the financial network)
 - *Divisible* (has the ability to make change)
 - *Unforgeable* (difficult to duplicate)
 - *Untraceable* (difficult to keep a record of where money is spent)
 - *Anonymous* (no record of who spent the money)
- ▶ Note: Many of these features are not available with credit cards.

– 251

Basic Concepts

- ▶ There are three players in an electronic payment:
 - A *payer* or *consumer* (called Alice)
 - A *payee* or *merchant* (called Bob)
 - A *financial network* (called the Bank)
- ▶ The electronic representation of cash is called a *token* or (*electronic*) *coin*.
- ▶ A device that stores the coin is called a *card*.

– 252

Payment System

- ▶ The sequence of events in a payment system is:
 - *Withdrawal*: Alice transfers some funds from her Bank account to her card.
 - *Payment*: Alice transfers money from her card to Bob's.
 - *Deposit*: Bob transfers the money he has received to his Bank account.
- ▶ *On-line payments*: Bob calls the Bank to verify the validity of Alice's coin before accepting her payment.
- ▶ *Off-line payments*: Bob submits Alice's coin for verification and deposit sometime after the payment transaction is completed.

– 253

Desirable Security Properties of E-Cash

- ▶ For the payer:
 - *Payer anonymity* during payment.
 - *Payment untraceability* so that the Bank cannot tell whose money is used in a particular payment.
- ▶ For the payee and Bank:
 - *Unforgeable coins*: creating a valid-looking coin without making a corresponding Bank withdrawal should be infeasible.
 - *No double-spending*: A coin cannot be used more than once for making a payment.

– 254

Other Desirable Properties of E-Cash

- ▶ Off-line payment system is preferable to on-line
- ▶ Payment mechanism should be cheap
- ▶ Payment mechanism should be efficient
- ▶ Cash should be transferable
- ▶ Cash should be divisible

– 255

Simplified Electronic Cash Protocols

Assumptions:

- ▶ The Bank has an RSA public key (n, e) and a corresponding private key d .
- ▶ The Bank's public key is known to Alice and Bob.

Notation:

- ▶ If M is a message, then $\{M\}$ denotes the message together with the Bank's RSA signature on the message. That is, $\{M\} = (M, s)$, where $s = H(M)^d \bmod n$.

– 256

Simplified On-Line Protocol

Withdrawal protocol

1. Alice requests a \$100 withdrawal from the Bank.
2. Bank debits Alice's account by \$100, and gives Alice the coin {This is a \$100 bill, #123456789}. Here, 123456789 is a (randomly selected) serial number.

Payment/Deposit protocol

1. Alice gives the coin to Bob.
2. Bob forwards the coin to the Bank.
3. Bank verifies the signature.
4. Bank verifies that the coin has not already been spent.
5. Bank enters coin in a *spent-coin database*.
6. Bank credits Bob's account with \$100 and informs Bob.
7. Bob completes the transaction with Alice.

– 257

Security Properties

Security properties of the simplified on-line protocol:

- ▶ Coins are unforgeable.
- ▶ Double-spending is *prevented*.
- ▶ Payments are traceable (since the Bank can link the coin to Alice's withdrawal request and the serial number).
- ▶ No payer anonymity (since the Bank can link the coin to Alice's withdrawal request and the serial number).

– 258

Simplified Off-Line Protocol

Withdrawal protocol

1. Alice requests a \$100 withdrawal from the Bank.
2. Bank debits Alice's account by \$100, and gives Alice the coin {This is a \$100 bill, #123456789}.

Payment protocol

1. Alice gives the coin to Bob.
2. Bob verifies the signature.
3. Bob completes the transaction with Alice.

Deposit protocol

1. Bob forwards the coin to the Bank.
2. Bank verifies the signature.
3. Bank verifies that the coin has not already been spent.
4. Bank enters coin in a *spent-coin database*.
5. Bank credits Bob's account with \$100.

– 259

Security Properties

Security properties of the simplified off-line protocol:

- ▶ Coins are unforgeable.
- ▶ Payments are traceable.
- ▶ No payer anonymity.
- ▶ Double-spending is not prevented. However, double-spending is *detected* after it has occurred. The culprit cannot be identified.

– 260

RSA Blind Signatures

Used for payer anonymity and payment untraceability.

Withdrawal protocol

1. Alice prepares the message $M = (\text{This is a \$100 bill, \#123456789})$.
2. Alice selects $r \in_R \mathbb{Z}_n^*$. [$r = \text{blinding factor}$]
3. Alice computes $m' = H(M)r^e \bmod n$.
4. Alice requests a \$100 withdrawal from the Bank, and gives m' to the Bank.
5. Bank debits Alice's account by \$100, and gives Alice the signature $s' = (m')^d \bmod n$.
Note that $s' \equiv H(M)^d r^{ed} \equiv H(M)^d r \equiv sr \pmod{n}$.
6. Alice computes $s = s'r^{-1} \bmod n$.
The coin is $\{M\} = (M, s)$.

– 261

RSA Blind Signatures (2)

Notes:

- ▶ $s = H(M)^d \bmod n$ is the Bank's signature on M .
However, the Bank does not know M .
- ▶ The payment and deposit protocols are the same as those in the simplified schemes.

– 262

Security Properties

- ▶ Payer anonymity and payment untraceability are provided since the Bank cannot link a coin to a particular withdrawal.
- ▶ Two problems remain:
 1. Coins can be forged. For example, the Bank may think it is signing a \$100 message, when in fact it is signing a \$1,000 message.
 2. Double-spending is detected in the off-line scheme, but the culprit cannot be identified.

– 263

Preventing Coin Forgery

There is a simple solution for preventing the forgery of coins.

- ▶ The Bank has one key pair for each denomination (e.g., one key pair for \$20 coins, one key pair for \$100 coins).
- ▶ The public key for each denomination is only valid for generating coins of that denomination.

– 264

On-Line Electronic Cash Protocol

Withdrawal protocol

1. Alice prepares the message $M = (\text{This is a \$100 bill, \#123456789})$.
2. Alice obtains the Bank's public key (n, e) for generating \$100 coins.
3. Alice selects $r \in_R \mathbb{Z}_n^*$.
4. Alice computes $m' = H(M)r^e \bmod n$.
5. Alice requests a \$100 withdrawal from the Bank, and gives m' to the Bank.
6. Bank debits Alice's account by \$100, and gives Alice the signature $s' = (m')^d \bmod n$ (where d is the Bank's private key for generating \$100 coins).
7. Alice computes $s = s'r^{-1} \bmod n$. The coin is (M, s) .

– 265

On-Line Electronic Cash Protocol (2)

Payment/Deposit protocol

1. Alice gives the coin to Bob.
2. Bob forwards the coin to the Bank.
3. Bank verifies the signature (with its public key for generating \$100 coins).
4. Bank verifies that the coin has not already been spent.
5. Bank enters coin in a spent-coin database.
6. Bank credits Bob's account with \$100 and informs Bob.
7. Bob completes the transaction with Alice.

– 266

Security Properties

- ▶ Payer anonymity and payment untraceability are provided.
- ▶ Coins are unforgeable.
- ▶ Double spending is prevented.

– 267

Off-Line Electronic Cash Protocol

- ▶ Double-spending of electronic cash in an off-line protocol cannot be (easily) prevented.
- ▶ Many protocols have been invented for detecting double-spending and identifying the culprit.
- ▶ The protocol described here is somewhat inefficient, but illustrates the basic ideas.
 - Protocols that are more efficient are known (but not described in this course).

– 268

Withdrawal Protocol

1. Alice prepares a string ID_A that contains her identifying information.
2. Alice obtains the Bank's public key (n, e) .
3. For each $1 \leq i \leq 1024$, $1 \leq j \leq 128$, Alice does:
 - Select a random bit string $x_{i,j}$ [*mask*]
 - Let $x'_{i,j} = x_{i,j} \oplus ID_A$ [*masked ID*]
 - Select random bit strings $s_{i,j}, s'_{i,j}$ [*salts*]
 - Compute $y_{i,j} = H(x_{i,j}, s_{i,j})$, $y'_{i,j} = H(x'_{i,j}, s'_{i,j})$ [*commitments*]
4. For each $1 \leq i \leq 1024$, Alice does:
 - Prepare the message:
 $M_i = (\text{This is a \$100 bill, } y_{i,1}, y'_{i,1}, \dots, y_{i,128}, y'_{i,128}).$
 - Select $r_i \in_R \mathbb{Z}_n^*$ and compute $m'_i = H(M_i)r_i^e \bmod n$.
5. Alice requests a \$100 withdrawal from the Bank, and gives $m'_1, m'_2, \dots, m'_{1024}$ to the Bank.

– 269

Withdrawal Protocol (2)

6. The Bank selects $k \in_R [1, 1024]$ and asks Alice to “unblind” all the m'_i except for m'_k .
7. Alice gives the Bank all the $M_i, x_{i,j}, x'_{i,j}, s_{i,j}, s'_{i,j}$ and r_i except for $i = k$.
8. For each $i \in [1, 1024]$, except for $i = k$, the Bank checks:
 M_i indeed is for \$100.
 $m'_i = H(M_i)r_i^e \bmod n$.
 $y_{i,j} = H(x_{i,j}, s_{i,j}), y'_{i,j} = H(x'_{i,j}, s'_{i,j})$ for $1 \leq j \leq 128$.
 $x_{i,j} \oplus x'_{i,j} = ID_A$ for $1 \leq j \leq 128$.
9. Bank debits Alice’s account by \$100, and gives Alice the signature $s' = (m'_k)^d \bmod n$.
10. Alice computes $s = s' r_k^{-1} \bmod n$.
11. The coin is (M_k, s) .

– 270

Withdrawal Protocol (3)

- $s = H(M_k)^d \bmod n$ is the Bank’s signature on M_k .
- The Bank does not know M_k .
- However, the Bank is convinced with probability $\frac{1023}{1024}$ that M_k is indeed for \$100, and that Alice knows $x_{k,j}, x'_{k,j}, s_{k,j}, s'_{k,j}$ such that $y_{k,j} = H(x_{k,j}, s_{k,j}), y'_{k,j} = H(x'_{k,j}, s'_{k,j})$ and $x_{k,j} \oplus x'_{k,j} = ID_A$ for $1 \leq j \leq 128$.

– 271

Payment Protocol

1. Alice gives (M_k, s) to Bob.
2. Bob verifies the signature using the Bank's public key.
3. Bob selects random bits b_1, b_2, \dots, b_{128} and sends them to Alice as a *challenge*.
4. Alice sends the *response* $(z_1, z_2, \dots, z_{128})$ to Bob, where $z_j = (x_{k,j}, s_{k,j})$ if $b_j = 0$ and $z_j = (x'_{k,j}, s'_{k,j})$ if $b_j = 1$.
5. For each $1 \leq j \leq 128$, Bob checks that $y_{k,j} = H(z_j)$ if $b_j = 0$ or $y'_{k,j} = H(z_j)$ if $b_j = 1$.
6. Bob completes the transaction with Alice.

– 272

Deposit Protocol

1. Bob sends (M_k, s) and $z = (z_1, \dots, z_{128})$ to the Bank.
2. Bank verifies the signature.
3. Bank verifies that the coin has not already been spent. If the coin has been spent, the Bank compares the z vectors of the two coins. If they are the same, the Bank concludes that Bob is trying to deposit the coin twice. If they are different, say the first components of the vectors are different, then the Bank XORs these components to reveal Alice's identity and concludes that Alice double-spent the coin.
4. Bank enters the coin and z in the spent-coin database.
5. Bank credits Bob's account with \$100.

– 273

Notes on the Off-Line E-Cash Protocol

- ▶ If Alice tries to double-spend a coin, then with probability $\frac{2^{128}-1}{2^{128}}$ the second challenge vector is different from the first. Hence the z vectors that Alice returns will almost certainly reveal her identity.
- ▶ Except with negligible probability, Bob cannot create a valid z vector different from the one that Alice gave him the first time she spent the coin.
- ▶ Payer anonymity and payment untraceability are provided.
- ▶ Coins are unforgeable.
- ▶ Double-spending is detected, and the culprit identified.

– 274

Miscellaneous Notes

- ▶ Preventing double-spending in off-line electronic cash schemes is possible with tamper-resistant cards.
- ▶ *Divisibility* of electronic cash is relatively easy to achieve.
- ▶ *Transferability* of electronic cash is not easy to achieve without tamper-resistant cards:
 - The coin must contain information about every individual who has spent it so that the Bank maintains the ability to identify multiple spenders.
 - Multiple spending will not be noticed until two copies of the same coin are eventually deposited.
- ▶ *Fair* electronic cash schemes have been proposed
 - Anonymity can be removed given an appropriate court order.

– 275

DISCRETE LOGARITHM CRYPTOGRAPHY

© *Alfred Menezes*

– 276

Outline

1. Basic Notions from Group Theory
2. The Discrete Logarithm Problem (DLP)
3. The Digital Signature Algorithm (DSA)
4. Diffie-Hellman (DH) Key Agreement

– 277

Basic Notions from Group Theory

- ▶ \mathbb{Z}_p , the integers modulo a prime p .
- ▶ The *order* of a nonzero element in \mathbb{Z}_p .
- ▶ Properties of order.

[See class notes]

– 278

The Discrete Logarithm Problem (DLP)

- ▶ Statement of the discrete logarithm problem.
- ▶ Shanks's algorithm.
- ▶ Running time of Pollard's rho algorithm.
- ▶ Running time of the number field sieve (NFS).

[See class notes]

– 279

Example 1

► Let $p = 53$.

► Then $\alpha = 12$ is a generator of \mathbb{Z}_{53}^* :

$12^1 = 12$	$12^2 = 38$	$12^3 = 32$	$12^4 = 13$	$12^5 = 50$
$12^6 = 17$	$12^7 = 45$	$12^8 = 10$	$12^9 = 14$	$12^{10} = 9$
$12^{11} = 2$	$12^{12} = 24$	$12^{13} = 23$	$12^{14} = 11$	$12^{15} = 26$
$12^{16} = 47$	$12^{17} = 34$	$12^{18} = 37$	$12^{19} = 20$	$12^{20} = 28$
$12^{21} = 18$	$12^{22} = 4$	$12^{23} = 48$	$12^{24} = 46$	$12^{25} = 22$
$12^{26} = 52$	$12^{27} = 41$	$12^{28} = 15$	$12^{29} = 21$	$12^{30} = 40$
$12^{31} = 3$	$12^{32} = 36$	$12^{33} = 8$	$12^{34} = 43$	$12^{35} = 39$
$12^{36} = 44$	$12^{37} = 51$	$12^{38} = 29$	$12^{39} = 30$	$12^{40} = 42$
$12^{41} = 27$	$12^{42} = 6$	$12^{43} = 19$	$12^{44} = 16$	$12^{45} = 33$
$12^{46} = 25$	$12^{47} = 35$	$12^{48} = 49$	$12^{49} = 5$	$12^{50} = 7$
$12^{51} = 31$	$12^{52} = 1$			

– 280

Example 1 (cont'd)

► $g = 47$ has order $q = 13$ in \mathbb{Z}_{53}^* :

$47^1 = 47$	$47^2 = 36$	$47^3 = 49$	$47^4 = 24$
$47^5 = 15$	$47^6 = 16$	$47^7 = 10$	$47^8 = 46$
$47^9 = 42$	$47^{10} = 13$	$47^{11} = 28$	$47^{12} = 44$
$47^{13} = 1$			

► The subgroup of \mathbb{Z}_{53}^* generated by $g = 47$ is:

$$\langle 47 \rangle = \{1, 10, 13, 15, 16, 24, 28, 36, 42, 44, 46, 47, 49\}.$$

► DLP instance: Solve $47^\ell = 13$ in \mathbb{Z}_{53}^* .

– 281

The Digital Signature Algorithm (DSA)

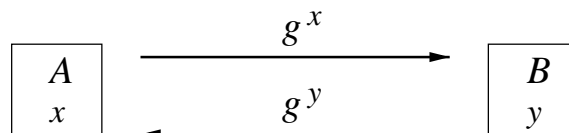
- ▶ Key generation.
- ▶ Signature generation.
- ▶ Signature verification.
- ▶ Security of DSA.
- ▶ Comparisons with the RSA signature scheme.

[See class notes]

– 282

Basic Diffie-Hellman Key Agreement Scheme

- ▶ Inventors: Whit Diffie and Martin Hellman; 1976.
- ▶ *Domain parameters*: p, q, g (same as DSA).

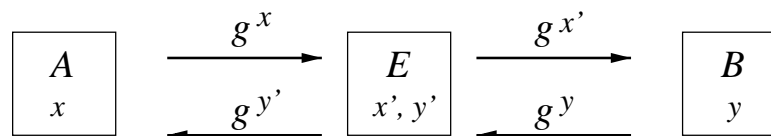


1. A selects $x \in_R [1, q - 1]$ and sends $g^x \bmod p$ to B .
2. B selects $y \in_R [1, q - 1]$ and sends $g^y \bmod p$ to A .
3. A computes $K = (g^y)^x \bmod p$ (the *shared secret*).
4. B computes $K = (g^x)^y \bmod p$.
5. The *shared key* is $k = H(K)$. (H is a hash function)

– 283

Security Notes

- ▶ An eavesdropper is faced with the following problem: given p, q, g, g^x and g^y , find g^{xy} . This is called the *Diffie-Hellman Problem (DHP)*.
- ▶ Assuming that the DHP is hard, the scheme is secure against *passive* adversaries.
- ▶ The scheme is totally insecure against *active* attacks. Classic example: Intruder-in-the-middle attack:



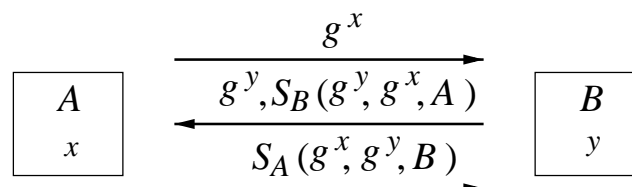
A computes $k = H(g^{xy'})$; B computes $k' = H(g^{x'y})$.
 E can compute both these keys.

- ▶ Problem: Exchanged numbers are not authenticated.

– 284

Station-to-Station Key Agreement Protocol

- ▶ Assumption: A and B a priori know each others public keys for a signature scheme such as RSA-FDH.
- ▶ Notation: $S_A(m)$ is A 's signature on m .



The shared key is $k = H(g^{xy})$.

- ▶ Note: The intruder-in-the-middle attack fails.
- ▶ STS is believed to be a secure key agreement protocol provided that the DHP is hard and the signature scheme is secure.

– 285

KEY MANAGEMENT

© *Alfred Menezes*

– 286

Outline

1. Public Key Management
2. PGP
3. Public-Key Infrastructures (PKI)
4. SSL/TLS
5. The Ideal Certification Process
6. Certificate Revocation

– 287

Key Management

- ▶ *Key management*: A set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties.
- ▶ We will consider management of public keys that are used for public-key encryption (such as RSA-OAEP), and for verification of digital signatures (such as RSA-FDH, DSA, ECDSA).

– 288

Two Scenarios

1. Suppose that A wishes to use public-key encryption (or hybrid encryption) to encrypt a message for B . To do this, A needs an authentic copy of B 's public key.
 - ▶ Example: A wishes to send B a confidential email.
 - ▶ Example: A wishes to send her credit number to B while making an online purchase.
2. Suppose that A received a message purportedly signed by B . To verify the signature on the message, A needs an authentic copy of B 's public key.
 - ▶ Example: A person A wishes to verify the authenticity of a software patch that was purportedly signed by Microsoft (B).
 - ▶ Example: The manager B of a bank branch can authorize financial transactions worth up to \$20,000.

– 289

Public Key Management

Some questions and concerns:

- ▶ Where does A get B 's public key from?
- ▶ How does A know she really has B 's public key?
- ▶ How can a bank limit use of B 's public/private key pair?
- ▶ What happens if B 's private key is compromised? Lost? Who is liable?
- ▶ How can a bank revoke B 's public key?
- ▶ How can B 's public keys be updated?
- ▶ How can non-repudiation services be provided?

– 290

Techniques for Distributing Public Keys

1. Point-to-point delivery over a trusted channel.
 - ▶ Trusted courier.
 - ▶ One-time user registration.
 - ▶ Voice.
2. Direct access to a trusted public file.
 - ▶ Digitally signed file.
 - ▶ Authentication trees.
3. Use of an on-line trusted server.
4. Off-line certification authority (CA).
5. Identity-based cryptography (IBE).

We will mainly discuss certification authorities.

– 291

Pretty Good Privacy (PGP)



- ▶ Introduced by Phil Zimmermann in 1991.
- ▶ Aims to provide low-cost email security for all.
- ▶ Free version: GNU Privacy Guard
(<http://www.gnupg.org/>).
- ▶ Uses of PGP:
 - Authenticating email: Helps to stop problems of email masquerading.
 - Confidentiality: Prevents others from reading the content of an email message.

– 292

PGP Crypto

- ▶ Symmetric-key algorithms supported:
 - AES, Triple-DES, ...
- ▶ Public-key algorithms supported:
 - RSA encryption and signatures, DSA, ...
- ▶ Public key management is not specified.
 - Obtain public keys directly — fingerprints (hashes of public keys) make this easier.
 - *Web of trust*: “If Alice trusts Bob, and Bob trusts Chris, then Alice trusts Chris”

– 293

Web of Trust

- ▶ A user Alice can have her public key validated and signed by other users.
- ▶ Alice has a *key ring* which stores others people's public keys.
- ▶ Also stored is an indication of how much Alice trusts the key's owner to validate and sign other keys:
 - Unknown, none, marginal, full.
- ▶ Alice can determine the authenticity of Bob's public key by verifying the signature of a user who signed Bob's public key. This chaining of trust can be carried out to any depth.
- ▶ There are several registries of public keys:
 - <http://pgp.mit.edu/> (MIT PGP public key server).
 - <http://keyserver.pgp.com/>.

– 294

Certification Authorities (CAs)

- ▶ A CA issues certificates which bind an entity's identity A and its public key.
- ▶ A 's *certificate* $Cert_A$ consists of:
 - *Data part* D_A : A 's identity, her public key, and other information such as validity period.
 - *Signature part* S_T : The CA's signature on the data part.
- ▶ B obtains an authentic copy of A 's public key as follows:
 - Obtain an authentic copy of the CA's public key (e.g., shipped in browsers or in an operating system).
 - Obtain $Cert_A$ (over an unsecured channel).
 - Verify the CA's signature S_T on D_A .
- ▶ Note: The CA does not have to be trusted with users' private keys.
- ▶ Note: The CA has to be trusted to not create false certificates.

– 295

Public-Key Infrastructure (PKI)

PKI: The supporting services (technological, legal, business, etc.) that are needed if public-key crypto is to be used on a wide scale.

Some components of a PKI:

- ▶ Certificate format.
- ▶ The certification process.
- ▶ Certificate revocation.
- ▶ Trust models.
- ▶ Certificate distribution.
- ▶ *Certificate policy*: Details of intended use and scope of a particular certificate.
- ▶ *Certification practice statement* (CPS): Practices and policies followed by a CA.

– 296

Public-Key Infrastructure (PKI)

Although conceptually very simple, there are many practical problems that are encountered when deploying PKI on a large scale. Many of these problems arises from business, legal, and useability considerations.

Problems include:

- ▶ Interoperability (alleviated by having standards and certificate formats).
- ▶ Certificate revocation.
- ▶ Trust models.
- ▶ Liability.


– 297

Secure Sockets Layer (SSL)

- ▶ SSL was designed by Netscape.
- ▶ TLS (Transport Layer Security) is an IETF version of SSL.
- ▶ SSL/TLS is used by web browsers such as Internet Explorer and Safari to protect web transactions.
- ▶ Main components of SSL/TLS
 - *Handshake protocol*: Allows server and client to authenticate each other and to negotiate cryptographic keys.
 - *Record protocol*: Used to encrypt and authenticate transmitted data.

– 298

Public Key Management in SSL/TLS

- ▶ Root CA keys are pre-installed in browsers.
 - Click on “Security” and then “Signers” to see a list of root CA keys in Safari.
- ▶ Web servers get their public keys certified by one of the Root CA’s (for a fee, of course).
 - *Verisign’s* web server certification business:
www.verisign.com/server/index.html

(Verisign is owned by Symantec)
- ▶ Clients (users) can obtain their own certificates. However, most users do not currently have their own certificates.
 - If clients do not have certificates, then authentication is only one-way (server authenticates itself to the client).

– 299

Example of an X.509 Certificate

Go to <https://www.amazon.ca> and click on the padlock.

- ▶ Subject name: Amazon.com, Inc.
`www.amazon.ca`
- ▶ Issuer name: Symantec Corporation
- ▶ Serial number: 19 74 91 ... 35 74
- ▶ Signature algorithm: SHA-256 with RSA Encryption.
- ▶ Valid from: Oct 23, 2016
- ▶ Valid to: Dec 30, 2017
- ▶ Subject public key info: RSA encryption, *2048-bit* n , $e = 65537$
- ▶ Key usage: Encrypt, Verify, Wrap, Derive
- ▶ Verisign's *2048-bit* RSA signature

– 300

SSL/TLS Handshake Protocol

- ▶ Main key establishment schemes:
 1. *RSA key transport*: The shared secret is selected by the client and encrypted with the server's RSA public key (using RSA PKCS #1 v1.5).
 2. *Fixed Diffie-Hellman*: Server's Diffie-Hellman public key g^x is in its certificate. Client may have g^y in its certificate, or generates a 1-time value g^y .
 3. *Ephemeral Diffie-Hellman*: Server selects a 1-time Diffie-Hellman public key g^x and signs it with its RSA or DSA signature key. Client selects 1-time g^y and signs it only if it has a certificate.
- ▶ MAC: HMAC-SHA-1, HMAC-MD5.
- ▶ *Symmetric-key encryption*: DES, Triple DES, RC4, AES, ...

– 301

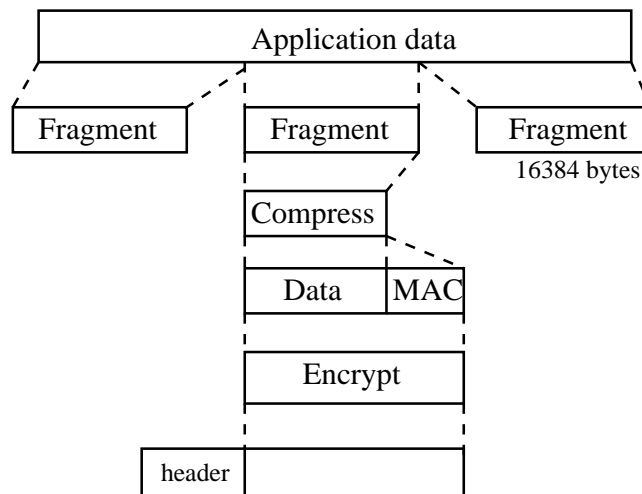
SSL/TLS Handshake Protocol (2)

1. *Phase 1*: Establish security capabilities.
 - ▶ Protocol version, compression method, cryptographic algorithms,...
2. *Phase 2*: Server authentication and key exchange.
 - ▶ Server sends its certificate, and key exchange parameters (if any).
3. *Phase 3*: Client authentication and key exchange.
 - ▶ Client sends its certificate (if available) and key exchange parameters.
4. *Phase 4*: Finish.

– 302

SSL/TLS Record Protocol

- ▶ Suppose that client and server share a MAC secret key and a session encryption key.



– 303

The Ideal Certification Process

1. The signature key pair of the CA is generated.
 - ▶ Security of the CA's private key is paramount.
 - ▶ Preferably done in tamper-resistant hardware.
 - ▶ Distribute shares of private key to several modules so that certificates cannot be created by any one device.
2. The key pair of A is generated.
 - ▶ Either by A , or by the CA.
3. A certificate for A is requested.
 - ▶ Authenticity of the request is required.
 - ▶ It may be necessary for the CA to maintain a record of the request.

– 304

The Ideal Certification Process (2)

4. A 's identity is verified.
 - ▶ May be tedious or expensive in practice.
 - ▶ Off-load the work to a *Registration Authority (RA)*; e.g., the post office or bank.
 - ▶ RA generates registration certificates and passes on to the CA for certificate issuance.
5. A 's key pair is validated.
 - ▶ CA verifies that the public key is valid, i.e., a private key logically exists.
 - ▶ A proves possession of the private key.
6. The CA produces A 's certificate.
7. A checks that the certificate is correct.
 - ▶ CA should require notification that A has accepted.

– 305

Certificate Revocation

- ▶ If a user's public key expires or is compromised, the CA must somehow inform all users that the certificates containing this public key are no longer valid. This is called *certificate revocation*.
- ▶ Reasons for certificate revocation include:
 1. Key compromise (rare).
 2. Owner leaves an organization.
 3. Owner changes role in an organization.
- ▶ A user should check validity of a certificate prior to using it.
- ▶ Revocation is easy to manage with an on-line CA.

– 306

Certificate Revocation Lists (CRLs)

A *CRL* is a list of revoked certificates that is signed and periodically issued by a CA. A user checks the CRL before using a certificate.

Problems with CRLs:

1. CRL time granularity
 - ▶ Time between revocation and CRL update.
2. The CRL may become too large. There are many ways to mitigate this problem including:
 - ▶ Delta CRLs: include only recently revoked certificates.
 - ▶ Group by revocation reason.
 - ▶ Distribution points: revocation data is split into buckets; each certificate contains data that determines the bucket it should be placed in.

– 307

ELLIPTIC CURVE CRYPTOGRAPHY

© *Alfred Menezes*

– 308

Outline

1. Introduction to Elliptic Curves
2. Elliptic Curve Discrete Logarithm Problem (ECDLP)
3. Elliptic Curve Diffie-Hellman (ECDH)
4. Elliptic Curve Digital Signature Algorithm (ECDSA)
5. Google and ECC

– 309

Elliptic Curves

- ▶ Definition of an elliptic curve.
- ▶ Basic properties of elliptic curves.
- ▶ Adding two points on an elliptic curve.
- ▶ The group of points on an elliptic curve.

[See class notes]

– 310

Example 2

- ▶ Let $p = 23$, and consider $E/\mathbb{Z}_{23} : Y^2 = X^3 + X + 1$.
- ▶ $\#E(\mathbb{Z}_{23}) = 28$.
- ▶ The 28 points in $E(\mathbb{Z}_{23})$ are:

(0,1)	(5,4)	(9,16)	(17,3)
(0,22)	(5,19)	(11,3)	(17,20)
(1,7)	(6,4)	(11,20)	(18,3)
(1,16)	(6,19)	(12,4)	(18,20)
(3,10)	(7,11)	(12,19)	(19,5)
(3,13)	(7,12)	(13,7)	(19,18)
(4,0)	(9,7)	(13,16)	∞
- ▶ Example of addition: $(3, 10) + (9, 7) = (17, 20)$.
- ▶ Example of doubling: $2(3, 10) = (7, 12)$.

– 311

Example 2 (cont'd)

- $P = (0, 1)$ is a generator of $E(\mathbb{Z}_{23})$, as the following shows:

$P=(0,1)$	$8P=(5,19)$	$15P=(9,7)$	$22P=(7, 12)$
$2P=(6,19)$	$9P=(19,18)$	$16P=(17,3)$	$23P=(18,20)$
$3P=(3,13)$	$10P=(12,4)$	$17P=(1,7)$	$24P=(13,7)$
$4P=(13,16)$	$11P=(1,16)$	$18P=(12,19)$	$25P=(3,10)$
$5P=(18,3)$	$12P=(17,20)$	$19P=(19,5)$	$26P=(6,4)$
$6P=(7,11)$	$13P=(9,16)$	$20P=(5,4)$	$27P=(0,22)$
$7P=(11,3)$	$14P=(4,0)$	$21P=(11,20)$	$28P=\infty$

– 312

Example 3

- Let $p = 23$.
- Consider $E/\mathbb{Z}_{23} : Y^2 = X^3 + X + 4$.
- $\#E(\mathbb{Z}_{23}) = 29$.
- The 29 points in $E(\mathbb{Z}_{23})$ are:

$(0, 2)$	$(9, 11)$	$(15, 6)$
$(0, 21)$	$(9, 12)$	$(15, 17)$
$(1, 11)$	$(10, 5)$	$(17, 9)$
$(1, 12)$	$(10, 18)$	$(17, 14)$
$(4, 7)$	$(11, 9)$	$(18, 9)$
$(4, 16)$	$(11, 14)$	$(18, 14)$
$(7, 3)$	$(13, 11)$	$(22, 5)$
$(7, 20)$	$(13, 12)$	$(22, 18)$
$(8, 8)$	$(14, 5)$	∞
$(8, 15)$	$(14, 18)$	

– 313

Example 3 (cont'd)

- $P = (0, 2)$ is a generator of $E(\mathbb{Z}_{23})$ as the following shows:

$1P = (0, 2)$	$11P = (10, 5)$	$21P = (14, 18)$
$2P = (13, 12)$	$12P = (17, 9)$	$22P = (15, 17)$
$3P = (11, 9)$	$13P = (8, 15)$	$23P = (9, 12)$
$4P = (1, 12)$	$14P = (18, 9)$	$24P = (7, 3)$
$5P = (7, 20)$	$15P = (18, 14)$	$25P = (1, 11)$
$6P = (9, 11)$	$16P = (8, 8)$	$26P = (11, 14)$
$7P = (15, 6)$	$17P = (17, 14)$	$27P = (13, 11)$
$8P = (14, 5)$	$18P = (10, 18)$	$28P = (0, 21)$
$9P = (4, 7)$	$19P = (22, 18)$	$29P = \infty$
$10P = (22, 5)$	$20P = (4, 16)$	

- ECDLP instance: Given $P = (0, 2)$ and $Q = (10, 18)$ find $\ell \in [0, 28]$ with $Q = \ell P$.

– 314

Elliptic Curve Cryptography



Neal Koblitz, Victor Miller

- The elliptic curve discrete logarithm problem (ECDLP).
- Elliptic Curve Diffie-Hellman (ECDH).
- Elliptic Curve Digital Signature Algorithm (ECDSA).
- Key size comparisons: RSA, DSA, ECDSA.

[See class notes]

– 315

P-256 Elliptic Curve

- ▶ P-256 is an elliptic curve chosen by the National Security Agency for U.S. federal government use.
- ▶ P-256 should be used for applications that require the *128-bit security level*.
- ▶ $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ is prime.
- ▶ P-256 is the elliptic curve $E : Y^2 = X^3 - 3X + b$ over \mathbb{Z}_p , where $b =$

410583637251521421293261297800472684091
14441015993725554835256314039467401291.

- ▶ We have $q = \#E(\mathbb{Z}_p)$ is prime, where $q =$

115792089210356248762697446949407573529
996955224135760342422259061068512044369.

– 316

Google and ECC

- ▶ In November 2011, Google started using ECDH as its default key establishment mechanism in SSL-secured applications including *Gmail* (<http://gmail.com>) and encrypted *search* (<http://google.com>).
- ▶ ECDH is supported in Chrome, Firefox, ...
- ▶ ECDH was chosen because it provides *forward secrecy* and because it is significantly faster than ordinary DH.
- ▶ The elliptic curve used is *P-256*.

– 317

SSL (as commonly implemented)

When a web browser (Alice) visits a secured web page (Bob):

1. Bob sends its *certificate* to Alice.
2. Alice verifies the signature [using RSA PKCS #1 v1.5] in the certificate using the Certificate Authority's RSA public key (which is embedded in the browser). If the certificate verifies, then Alice is assured that she has an authentic copy of Bob's RSA public key.
3. Alice selects a random *session key* k , and encrypts k with Bob's RSA public key [using RSA PKCS #1 v1.5]. Alice sends the resulting ciphertext c to Bob.
4. Bob decrypts c using his RSA private key and obtains k .
5. The session key is used to authenticate (with HMAC) and encrypt (with AES) all data exchanged for the remainder of the session.

– 318

Forward Secrecy

- ▶ SSL, as described on the previous slide, does not provide forward secrecy.
- ▶ That is, suppose that an eavesdropper saves a copy of c and the encrypted data. If, at a future point in time, the eavesdropper is able to break into Bob's machine and learn his RSA private key, then the eavesdropper is able to decrypt c and thus recover k and the data that was encrypted with k .
- ▶ Using ECDH to establish a session key k provides forward secrecy.

– 319

SSL as implemented by Google

When a web browser (Alice) visits a secured web page (Bob):

1. Bob sends its *certificate* to Alice.
2. Alice verifies the signature in the certificate using the Certificate Authority's RSA public key (which is embedded in the browser). If the certificate verifies, then Alice is assured that she has an authentic copy of Bob's RSA public key.
3. Alice selects a random integer $a \in [1, q - 1]$ and sends $A = aP$ to Bob. (P is a fixed point on P-256.)
4. Bob selects a random integer $b \in [1, q - 1]$, signs $B = bP$ with its RSA signing key, and sends B and the signature to Alice.
5. Alice verifies the signature using Bob's RSA public key, and is thus assured that B was sent by Bob.

– 320

SSL as implemented by Google (2)

6. Both Alice and Bob compute the *shared secret* $K = aB = bA = abP$, and derive a *session key* k from K .
7. Alice deletes a . Bob deletes b .
8. The session key is used to authenticate (with HMAC-SHA1) and encrypt (with AES) all data exchanged for the remainder of the session.
9. At the end of the session, Alice and Bob delete k .

Note that forward secrecy is provided.

For further information, see

<http://tinyurl.com/GoogleECDH>

– 321

Example of an X.509 Certificate

Go to <https://www.google.ca> and click on the padlock.

- ▶ Subject name: Google Inc.
*.google.com
- ▶ Issuer name: Google Internet Authority G2
- ▶ Serial number: 3128375222928547080
- ▶ Signature algorithm: SHA-256 with RSA Encryption
- ▶ Valid from: Dec 15, 2016
- ▶ Valid to: Mar 9, 2017
- ▶ Subject public key info: *Elliptic Curve Public Key, secp256r1*
- ▶ Key usage: Encrypt, Verify, Derive
- ▶ Issue's *2048-bit* RSA signature

– 322

RSA vs DL vs ECC Usage in SSL

Percentage of https connections that use RSA, DL or ECC as of November 2016:

Key exchange:

RSA: 39%

DH: 10%

ECDH: 51%

Signatures:

RSA: 99%

DSA: >0%

ECDSA: 1%

– 323

RSA vs DL vs ECC Usage in SSH

Percentage of ssh connections that use RSA, DL or ECC as of November 2016:

Key exchange:

RSA: $\approx 0\%$

DH: 52%

ECDH: 48%

Signatures:

RSA: 93%

DSA: 7%

ECDSA: 0.3%