**Cryptography Outline**

*Introduction*: terminology, cryptanalysis, security  
*Primitives*: one-way functions, trapdoors, ...  
*Protocols*: digital signatures, key exchange, ...  
*Number Theory*: groups, fields, ...  
*Private-Key Algorithms*: Rijndael, DES  
*Public-Key Algorithms*:  
  - Diffie-Hellman Key Exchange  
  - RSA, El-Gamal, Blum-Goldwasser  
  - Quantum Cryptography  
*Case Studies*: Kerberos, Digital Cash

---

**Public Key Cryptosystems**

Introduced by Diffie and Hellman in 1976.

\[
\begin{array}{c}
\text{Plaintext} \\
\downarrow \\
K_1 \xrightarrow{\text{Encryption}} E_k(M) = C \\
\downarrow \\
\text{Cyphertext} \\
\downarrow \\
K_2 \xrightarrow{\text{Decryption}} D_k(C) = M \\
\downarrow \\
\text{Original Plaintext}
\end{array}
\]

Typically used as part of a more complicated protocol.

---

**One-way trapdoor functions**

Both Public-Key and Digital signatures make use of one-way trapdoor functions.

**Public Key:**
- Encode: \( c = f(m) \)
- Decode: \( m = f^{-1}(c) \) using trapdoor

**Digital Signatures:**
- Sign: \( c = f^{-1}(m) \) using trapdoor
- Verify: \( m = f(c) \)
Example of SSL (3.0)

SSL (Secure Socket Layer) is the standard for the web (https).

Protocol (somewhat simplified): Bob -> amazon.com

B->A: client hello: protocol version, acceptable ciphers
A->B: server hello: cipher, session ID, [amazon.com] verifying
B->A: key exchange: [masterkey],amazon's public key
A->B: server finished: [amazon,prev-messages,masterkey]key1
B->A: client finished: [bob,prev-messages,masterkey]key2
A->B: server message: (message1,[message1])key1
B->A: client message: (message2,[message2])key2

| issuer          | = Certificate = Issuer, <h,h's public key, time stamp>, issuer's private key
<...private key | = Digital signature = [public key] = Public-key encryption
| key1 and key2   | are derived from masterkey and session ID

Public Key History

Some algorithms
- Diffie-Hellman, 1976, key-exchange based on discrete logs
- Merkle-Hellman, 1978, based on "knapsack problem"
- McEliece, 1978, based on algebraic coding theory
- RSA, 1978, based on factoring
- Rabin, 1979, security can be reduced to factoring
- ElGamal, 1985, based on discrete logs
- Blum-Goldwasser, 1985, based on quadratic residues
- Elliptic curves, 1985, discrete logs over Elliptic curves
- Chor-Rivest, 1988, based on knapsack problem
- NTRU, 1996, based on Lattices
- XTR, 2000, based on discrete logs of a particular field

Diffie-Hellman Key Exchange

A group (G,*) and a primitive element (generator) g is made public.
- Alice picks a, and sends ga to Bob
- Bob picks b and sends gb to Alice
- The shared key is gab

Note this is easy for Alice or Bob to compute, but assuming discrete logs are hard is hard for anyone else to compute.

Can someone see a problem with this protocol?

Person-in-the-middle attack

Can someone see a problem with this protocol?

Mallory gets to listen to everything.
Merkle-Hellman

Gets "security" from the Subset Sum (also called knapsack) which is NP-hard to solve in general.

**Subset Sum (Knapsack):** Given a sequence \( W = \{w_0, w_1, \ldots, w_{n-1}\} \), \( w_i \in \mathbb{Z} \) of weights and a sum \( S \), calculate a boolean vector \( B \), such that:

\[
\sum_{i=0}^{n} B_i W_i = S
\]

Even deciding if there is a solution is NP-hard.

Merkle-Hellman

**W is superincreasing if:** \( w_j \geq \sum_{i=0}^{j-1} w_i \)

It is easy to solve the subset-sum problem for superincreasing \( W \) in \( O(n) \) time.

**Main idea of Merkle-Hellman:**
- Hide the easy case by multiplying each \( w_i \) by a constant \( a \) modulo a prime \( p \)
  \[
  w'_i = a \cdot w_i \mod p
  \]
- Knowing \( a \) and \( p \) allows you to retrieve the superincreasing sequence

Merkle-Hellman: Problem

Was broken by Shamir in 1984.
Shamir showed how to use integer programming to solve the particular class of Subset Sum problems in polynomial time.

Lesson: don’t leave your trapdoor loose.
RSA

Invented by Rivest, Shamir and Adleman in 1978
Based on difficulty of factoring.
Used to hide the size of a group $\mathbb{Z}_n^*$ since:

$|\mathbb{Z}_n^*| = \phi(n) = n \prod_{p | n}(1-1/p)$

Factoring has not been reduced to RSA
- an algorithm that generates $m$ from $c$ does not give an efficient algorithm for factoring
On the other hand, factoring has been reduced to finding the private-key.
- there is an efficient algorithm for factoring given one that can find the private key.

RSA Public-key Cryptosystem

What we need:
- $p$ and $q$, primes of approximately the same size
- $n = pq$
- $\phi(n) = (p-1)(q-1)$
- $e \in \mathbb{Z}_{\phi(n)}^*$
- $d = e^{-1} \mod \phi(n)$

Public Key: $(e,n)$
Private Key: $d$

Encode:
- $m \in \mathbb{Z}_n$
- $E(m) = m^e \mod n$

Decode:
- $D(c) = c^d \mod n$

RSA computations

To generate the keys, we need to
- Find two primes $p$ and $q$. Generate candidates and use primality testing to filter them.
- Find $e^{-1} \mod (p-1)(q-1)$. Use Euclid’s algorithm. Takes time $\log^3(n)$

To encode and decode
- Take $m^e$ or $c^d$. Use the power method.
  Takes time $\log(e) \log^2(n)$ and $\log(d) \log^2(n)$.

In practice $e$ is selected to be small so that encoding is fast.

RSA continued

Why it works:

$D(c) = c^d \mod n$

$= m^{ed} \mod n$

$= m^{(e^{(p-1)(q-1)})} \mod n$

$= m^{\phi(n)k} \mod n$

$= m$

Why is this argument not quite sound?
What if $m \not\in \mathbb{Z}_n^*$ then $m^{\phi(n)} \not\equiv 1 \mod n$

Answer 1: Not hard to show that it still works.
Answer 2: jackpot – you’ve factored $n$
Security of RSA

**Warning:**
- Do not use this or any other algorithm naively!

**Possible security holes:**
- Need to use "safe" primes p and q. In particular p-1 and q-1 should have large prime factors.
- p and q should not have the same number of digits. Can use a middle attack starting at sqrt(n).
- e cannot be too small
- Don’t use same n for different e’s.
- You should always "pad"

Algorithm to factor given d and e

If an attacker has an algorithm that generates d from e, then he/she can factor n in PPT. Variant of the Rabin-Miller primality test.

Function TryFactor(e, d, n)
1. write $ed - 1$ as $2^r \cdot r$ odd
2. choose $w$ at random $< n$
3. $v = w^r \mod n$
4. if $v = 1$ then return (fail)
5. while $v \neq 1 \mod n$
6. $v_0 = v$
7. $v = v^2 \mod n$
8. if $v_0 = n - 1$ then return (fail)
9. return (pass, gcd($v_0 + 1, n$))

Las Vegas algorithm
Probability of pass is > 0.5.
Will return p or q if it passes.
Try until you pass.

RSA Performance

Performance: (600Mhz PIII) (from: ssh toolkit):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bits/key</th>
<th>Mbits/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA Keygen</td>
<td>1024</td>
<td>3.5 sec/key</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>2.83 sec/key</td>
</tr>
<tr>
<td>RSA Encrypt</td>
<td>1024</td>
<td>1786/sec</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>672/sec</td>
</tr>
<tr>
<td>RSA Decrypt</td>
<td>1024</td>
<td>74/sec</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>12/sec</td>
</tr>
<tr>
<td>ElGamal Enc.</td>
<td>1024</td>
<td>31/sec</td>
</tr>
<tr>
<td>ElGamal Dec.</td>
<td>1024</td>
<td>61/sec</td>
</tr>
<tr>
<td>DES-cbc</td>
<td>56</td>
<td>95</td>
</tr>
<tr>
<td>twofish-cbc</td>
<td>128</td>
<td>140</td>
</tr>
<tr>
<td>Rijndael</td>
<td>128</td>
<td>180</td>
</tr>
</tbody>
</table>

RSA in the “Real World”

Part of many standards: PKCS, ITU X.509, ANSI X9.31, IEEE P1363

Used by: SSL, PEM, PGP, Entrust, ...

The standards specify many details on the implementation, e.g.
- e should be selected to be small, but not too small
- "multi prime" versions make use of n = pqr... this makes it cheaper to decode especially in parallel (uses Chinese remainder theorem).
Factoring in the Real World

**Quadratic Sieve (QS):**

\[ T(n) = e^{(1+o(n))(\ln n)^{1/2} \ln(\ln n)^{1/2}} \]

- Used in 1994 to factor a 129 digit (428-bit) number. 1600 Machines, 8 months.

**Number field Sieve (NFS):**

\[ T(n) = e^{(1.923+o(1))(\ln n)^{1/3} \ln(\ln n)^{2/3}} \]

- Used in 1999 to factor 155 digit (512-bit) number. 35 CPU years. At least 4x faster than QS
- Used in 2003-2005 to factor 200 digits (663 bits) 75 CPU years ($20K prize)

ElGamal

Based on the difficulty of the discrete log problem.

Invented in 1985

Digital signature and Key-exchange variants
- Digital signature is AES standard
- Public Key used by TRW (avoided RSA patent)

Works over various groups
- \( \mathbb{Z}_p \)
- Multiplicative group GF(p^n)
- Elliptic Curves

ElGamal Public-key Cryptosystem

\( (G, *) \) is a group

- \( \alpha \) a generator for \( G \)
- \( a \in \mathbb{Z}_{|G|} \)
- \( \beta = \alpha^{a} \)

\( G \) is selected so that it is hard to solve the discrete log problem.

**Public Key:** \((\alpha, \beta)\) and some description of \( G \)

**Private Key:** \( a \)

**Encode:**

Pick random \( k \in \mathbb{Z}_{|G|} \)

\[ E(m) = (y_1, y_2) = (\alpha^k, m * \beta^k) \]

**Decode:**

\[ D(y) = y_2 * (y_1^{-1}) = (m * \beta^k) * (\alpha^{-k})^{-1} = m \]

You need to know \( a \) to easily decode \( y! \)

ElGamal: Example

\( G = \mathbb{Z}_{11}^* \)

- \( \alpha = 2 \)
- \( a = 8 \)
- \( \beta = 2^a \) (mod 11) = 3

**Encode:** 7

Pick random \( k = 4 \)

\[ E(m) = (2^4, 7 * 3^4) = (5, 6) \]

**Decode:** (5, 6)

\[ D(y) = 6 * (5^8)^{-1} = 6 * 4^{-1} = 6 * 3 \) (mod 11) = 7

**Public Key:** (2, 3), \( \mathbb{Z}_2^* \)

**Private Key:** \( a = 8 \)
Probabilistic Encryption

For RSA one message goes to one cipher word. This means we might gain information by running $E_{public}(M)$.

Probabilistic encryption maps every $M$ to many $C$ randomly. Cryptanalysts can't tell whether $C = E_{public}(M)$.

ElGamal is an example (based on the random k), but it doubles the size of message.

BBS “secure” random bits

BBS (Blum, Blum and Shub, 1984)

- Based on difficulty of factoring, or finding square roots modulo $n = pq$.

Fixed
- $p$ and $q$ are primes such that $p = q = 3 \pmod{4}$
- $n = pq$ (is called a Blum integer)

For a particular bit seq.
- Seed: random $x$ relatively prime to $n$.
- Initial state: $x_0 = x^2$
- $i^{th}$ state: $x_i = (x_{i-1})^2$
- $i^{th}$ bit: lsb of $x_i$

Note that: $x_0 = x_{i-2} \pmod{\phi(n)}(\pmod{n})$
Therefore knowing $p$ and $q$ allows us to find $x_0$ from $x_i$.

Blum-Goldwasser: A stream cypher

Public key: $n (= pq)$  Private key: $p$ or $q$

Encrypt: $m_i \ (0 \leq i < l)$ xor $b_i \rightarrow c_i \ (0 \leq i < l)$

Decrypt:
Using $p$ and $q$, find $x_0 = x_i^{-2} \pmod{p-1}(q-1)(\pmod{n})$
Use this to regenerate the $b_i$ and hence $m_i$

Quantum Cryptography

In quantum mechanics, there is no way to take a measurement without potentially changing the state. E.g.
- Measuring position, spreads out the momentum
- Measuring spin horizontally, “spreads out” the spin probability vertically
Related to Heisenberg’s uncertainty principal
Using photon polarization

\[
\begin{align*}
\downarrow &= \downarrow \text{ or } \uparrow \downarrow \text{ (equal probability)} \\
\rightarrow &= \uparrow \text{ or } \rightarrow \text{ (equal probability)}
\end{align*}
\]

measure diagonal measure square

destroys state

Quantum Key Exchange

1. Alice sends bob photon stream randomly polarized in one of 4 polarizations:

\[
\begin{array}{c}
\downarrow \rightarrow \downarrow \uparrow \\
\end{array}
\]

2. Bob measures photons in random orientations
e.g.: \(X + + X X X + X\) (orientations used)
\(\downarrow \rightarrow \downarrow \rightarrow \downarrow \rightarrow \rightarrow\) (measured polarizations)
and tells Alice in the open what orientations he used, but not what he measured.
3. Alice tells Bob in the open which are correct
4. Bob and Alice keep the correct values
Susceptible to a man-in-the-middle attack

In the “real world”

Not yet used in practice, but experiments have verified that it works.
IBM has working system over 30cm at 10bits/sec.
More recently, up to 10km of fiber.

Cryptography Outline

Introduction: terminology, cryptanalysis, security
Primitives: one-way functions, trapdoors, ...
Protocols: digital signatures, key exchange, ...
Number Theory: groups, fields, ...
Private-Key Algorithms: Rijndael, DES
Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...
Case Studies:
- Kerberos
- Digital Cash
Kerberos

A key-serving system based on Private-Keys (DES).

Assumptions
- Built on top of TCP/IP networks
- Many "clients" (typically users, but perhaps software)
- Many "servers" (e.g. file servers, compute servers, print servers, ...)
- User machines and servers are potentially insecure without compromising the whole system
- A kerberos server must be secure.

Kerberos V Message Formats

C = client  S = server  K = key
T = timestamp  V = time range
TGS = Ticket Granting Service  A = Net Address

Ticket Granting Ticket: \[ T_{C,TGS} = TGS, (C, A, V, K_{C,TGS})K_{TGS} \]
Server Ticket: \[ T_{C,S} = S, \{C, A, V, K_{C,S}\}K_{S} \]
Authenticator: \[ A_{C,S} = \{C, T, [K]\}K_{C,S} \]

1. Client to Kerberos: \( (C, TGS)K_{C} \)
2. Kerberos to Client: \( (K_{C,TGS})K_{C}, T_{C,TGS} \)
3. Client to TGS: \( A_{C,TGS}, T_{C,TGS} \)
4. TGS to Client: \( (K_{C})K_{T,TGS}, T_{C,S} \) Possibly repeat
5. Client to Server: \( A_{C,S}, T_{C,S} \)

Kerberos Notes

All machines have to have synchronized clocks
- Must not be able to reuse authenticators
- Servers should store all previous and valid tickets
  - Help prevent replays

Client keys are typically a one-way hash of the password. Clients do not keep these keys.
Kerberos 5 uses CBC mode for encryption Kerberos 4 was insecure because it used a nonstandard mode.
Electronic Payments

Privacy
- Identified
- Anonymous

Involvement
- Offline (just buyer and seller)
  more practical for "micropayments"
- Online
  - Notational fund transfer (e.g. Visa, CyberCash)
  - Trusted 3rd party (e.g. FirstVirtual)

Today: "Digital Cash" (anonymous and possibly offline)

Some more protocols
1. Secret splitting (and sharing)
2. Bit commitment
3. Blind signatures

Secret Splitting
Take a secret (e.g. a bit-string B) and split it among multiple parties such that all parties have to cooperate to regenerate any part of the secret.

An implementation:
- Trent picks a random bit-string R of same length as B
- Sends Alice R
- Sends Bob R xor B
Generalizes to k parties by picking k-1 random values.

Secret Sharing
m out of n (m < n) parties can recreate the secret. Also called an (m,n)-threshold scheme

An implementation (Shamir):
- Write secret as coefficients of a polynomial GF(p)[x] of order m-1 (n ≤ p).
  p(x) = c_{m-1}x^{m-1} + ... + c_1 x + c_0
- Evaluate p(x) at n distinct points in GF(p)
- Give each party one of the results
- Any m results can be used to reconstruct the polynomial.
**Bit Commitment**

Alice commits a bit to Bob without revealing the bit (until Bob asks her to prove it later)

*An implementation:*
- **Commit**
  - Alice picks random $r$, and uses a one-way hash function to generate $y = f(r, b)$
  - $f(r, b)$ must be "unbiased" on $b$ ($y$ by itself tells you nothing about $b$).
  - Alice sends Bob $y$.
- **Open** (expose bit and prove it was committed)
  - Alice sends Bob $b$ and $r$.

*Example:* $y = \text{Rijndael}_r(000...b)$

---

**Blind Signatures**

Sign a message $m$ without knowing anything about $m$

*An implementation: based on RSA*

- Trent blindly signs a message $m$ from Alice
  - Trent has public key $(e, n)$ and private key $d$
  - Alice selects random $r < n$ and generates $m' = m r^e \mod n$
  - Trent signs it: $s(m') = (m r^e)^d \mod n$
  - Trent sends it to Trent. This is called **blinding** $m$.
- Alice calculates: $s(m) = s(m') r^{-1} = m^d r^{-e} r^{-1} = m^d \mod n$

Patented by Chaum in 1990.

---

**An anonymous online scheme**

1. Blinded Unique Random large ID (no collisions). $\text{Sig}_{\text{alice}}$(*request for $100*)
2. $\text{Sig}_{\text{bank} \_ \$100}$(*blinded*($\text{ID}$)): signed by bank
3. $\text{Sig}_{\text{bank} \_ \$100}$(*ID*)
4. $\text{Sig}_{\text{bank} \_ \$100}$(*ID*)
5. OK from bank
6. OK from merchant

*Minting:* 1. and 2. *Spending:* 3.-6. *Left out encryption*
**eCash**

Uses the protocol
Bought assets and patents from Digicash
Founded by Chaum, went into Chapter 11 in 1998
Has not picked up as fast as hoped
- Credit card companies are putting up fight and transactions are becoming more efficient
- Government is afraid of abuse
Currently mostly used for Gift Certificates, but also used by Deutsche Bank in Europe.

**The Perfect Crime**

- Kidnapper takes hostage
- Ransom demand is a series of blinded coins
- Banks signs the coins to pay ransom
- Kidnapper tells bank to publish the coins in the newspaper (they're just strings)
- Only the kidnapper can unblind the coins (only he knows the blinding factor)
- Kidnapper can now use the coins and is completely anonymous

**Chaum’s protocol for offline anonymous cash**

How do we prevent double payment without bank intervention?

**Idea:**
- If used properly, Alice stays anonymous
- If Alice spends a coin twice, she is revealed
- If Merchant remits twice, this is detected and Alice remains anonymous
- Must be secure against Alice and Merchant colluding
- Must be secure against one framing the other.

An amazing protocol

**Chaum’s protocol: money orders**

- $u = Alice's account number (identifies her)$
- $r_0, r_1, ..., r_{n-1} = n$ random numbers
- $(u_l, u_r) = a$ secret split of $u$ using $r_i \ (0 \leq i < n)$
  - e.g. using $(r_l, r_l \ xor \ u)$
- $v_l_i = a$ bit commitment of all bits of $u_l_i$
- $v_r_i = a$ bit commitment of all bits of $u_r_i$

**Money order:**
- Amount
- Unique ID
- $(v_l_0, v_r_0), (v_l_1, v_r_1), ..., (v_l_{n-1}, v_r_{n-1})$
**Chaum’s protocol: Minting**

1. Two blinded money orders and Alice's account #
2. A request to unblind and prove all bit commitments for one of the two orders (chosen at random)
3. The blinding factor and proof of commitment for that order
4. Assuming step 3 passes, the other blinded order signed

**Chaum’s protocol: Spending**

1. The signed money order C (unblinded)
2. A random bit vector B of length n
3. For each i if B_i = 0 return bit values for u_l else return bit values for u_r
   Include all "proofs" that the u_l or u_r match v_l or v_r
   Now the merchant checks that the money order is properly signed by the bank, and that the u_l or u_r match the v_l or v_r

**Chaum’s protocol: Returning**

1. The signed money order
   The vector B along with the values of u_l or u_r that it received from Alice.
2. An OK, or fail
   If fail, i.e., already returned:
   1. If B matches previous order, the Merchant is guilty
   2. Otherwise Alice is guilty and can be identified since
      for some i (where Bs don't match) the bank will have (u_l, u_r), which reveals her secret u (her identity).