15-853: Algorithms in the Real World

Cryptography 3 and 4
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
  - El-Gamal, RSA, Blum-Goldwasser
  - Quantum Cryptography
Case Studies: Kerberos, Digital Cash
**Public Key Cryptosystems**

Introduced by Diffie and Hellman in 1976.

- **Encryption**
  - Plaintext \( E_k(M) = C \)
  - Cyphertext
  - Original Plaintext

- **Decryption**
  - Public Key systems
    - \( K_1 = \) public key
    - \( K_2 = \) private key
  - Digital signatures
    - \( K_1 = \) private key
    - \( K_2 = \) public key

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 TLS (previously SSL)

TLS (Transport Layer Security) is the standard for the web (https), and voice over IP.

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

B->A: **client hello**: protocol version, acceptable ciphers
A->B: **server hello**: cipher, session ID, |amazon.com|_verisign_

B->A: **key exchange**, {masterkey}amazon’s public key
A->B: **server finish**: ([amazon,prev-messages,masterkey])_key1_

B->A: **client finish**: ([bob,prev-messages,masterkey])_key2_

A->B: **server message**: (message1,[message1])_key1_
B->A: **client message**: (message2,[message2])_key2_

|h|_issuer_ = Certificate

= Issuer, ⟨h,h’ s public key, time stamp⟩_issuer’ s private key_

⟨…⟩_private key_ = Digital signature

{…}public key = Public-key encryption

[...] = Secure Hash

(...)key = Private-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 \((G,\ast)\) and a primitive element (generator) \(g\) is made public.

- Alice picks \(a\), and sends \(g^a\) to Bob
- Bob picks \(b\) and sends \(g^b\) to Alice
- The shared key is \(g^{ab}\)

Note the shared key 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

\[ Key_1 = g^{ad} \quad \text{Key}_1 = g^{cb} \]

Mallory gets to listen to everything.
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 $\text{GF}(p^n)$,
  - Elliptic Curves
### ElGamal Public-key Cryptosystem

**Encode:**
Pick random \( k \in \mathbb{Z}_{|G|} \)
\[ E(m) = (y_1, y_2) = (\alpha^k, m \cdot \beta^k) \]

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

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

\((G, \ast)\) 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:** \( a \)
**ElGamal: Example**

\[ G = \mathbb{Z}_{11}^* \]
- \( \alpha = 2 \)
- \( \alpha = 8 \)
- \( \beta = 2^8 \pmod{11} = 3 \)

**Public Key:** \((2, 3), \mathbb{Z}_{11}^*\)
**Private Key:** \(\alpha = 8\)

**Encode:** 7
Pick random \(k = 4\)
\[ E(m) = (2^4, 7 \times 3^4) \]
\[ = (5, 6) \]

**Decode:** \((5, 6)\)
\[ D(y) = 6 \times (5^8)^{-1} \]
\[ = 6 \times 4^{-1} \]
\[ = 6 \times 3 \pmod{11} \]
\[ = 7 \]
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:

$$\sum_{i=0}^{i<n} B_i w_i = S$$

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

W is superincreasing if: \[ w_i \geq \sum_{j=0}^{i-1} w_j \]

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 \( p \)
  \[ w'_i = a \times w_i \mod p \]
- Knowing \( a \) and \( p \) allows you to retrieve the superincreasing sequence
**Merkle-Hellman**

**What we need**
- $w_1, \ldots, w_n$
  - superincreasing integers
- $p > \sum_{i=1}^n w_i$ and prime
- $a, \ 1 \leq a \leq n$
- $w'_i = a \cdot w_i \mod p$

**Public Key:** $w'_i$

**Private Key:** $w_i, p, a,$

**Encode:**
$$y = E(m) = \sum_{i=1}^n m_i \cdot w'_i$$

**Decode:**
$$z = a^{-1} \cdot y \mod p$$

Solve subset sum prob:
$$(w_1, \ldots, w_n, z)$$

obtaining $m_1, \ldots, m_n$
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


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 from the public 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 continued

Why it works:
\[ D(c) = c^d \mod n \]
\[ = m^{ed} \mod n \]
\[ = m^1 + k(p-1)(q-1) \mod n \]
\[ = m^1 + k \phi(n) \mod n \]
\[ = m(m \phi(n))^k \mod n \]
\[ = m \]

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

Answer 1: Not hard to show that it still works.
Answer 2: jackpot – you’ve factored 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 $\log^2(n)$

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.
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 $2^s r$, 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))

LasVegas algorithm
Probability of pass is > .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>.35sec/key</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>2.83sec/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)
- In 2009 the RSA 768 bits number was factored
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: $x_0 = x_i^{-2^i \mod \phi(n)} \mod n$ (mod $n$)
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:**
\[
m_i \ (0 \leq i < l) \quad \xrightarrow{\text{xor}} \quad c_i \ (0 \leq i < l)
\]

**Decrypt:**
Using \( p \) and \( q \), find \( x_0 = x_i^{-2^j \mod (p-1)(q-1)} \mod n \)

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

\[ \uparrow \downarrow = \uparrow \downarrow \quad \text{or} \quad \downarrow \uparrow \quad ? \quad (\text{equal probability}) \]

\[ \uparrow \downarrow = \uparrow \downarrow \quad \text{or} \quad \downarrow \uparrow \quad ? \quad (\text{equal probability}) \]

measure diagonal

measure square

destroys state
Quantum Key Exchange

1. Alice sends bob photon stream randomly polarized in one of 4 polarizations:
   \[ \uparrow \quad \longleftrightarrow \quad \wedge \wedge \]

2. Bob measures photons in random orientations
e.g.: \[ \times + + \times \times \times + \times \] (orientations used)
     \[ \\hat{\uparrow} \quad \hat{-} \quad \hat{-} \hat{-} \hat{-} \hat{-} \hat{-} \] (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:
- Bitcoin
Bitcoin

Developed by “Satoshi Nakamoto” in 2009.
Total value: $175 Billion
About 15 Million “coins” in circulation.
**Bitcoin**

**Blockchain**: Maintains a digital ledger of transactions as a linked list (chain) of blocks.

**Blocks**: Each block can contain multiple transactions.

**Distributed**: Maintained in a distributed fashion.

**“Coinbase transactions”**: Get “paid” in new cash (and possibly transaction fees) for adding a new block of transactions to the chain.

**Proof of work**: To add a block one needs to do some “hard” work and prove they have done it.

**Public**: The blockchain is publicly readable by all, and needs to be.
Bitcoin

Digital Ledger of transactions

1. New Cash -> Joe, $2
2. Joe -> Alice, $2 (1)
3. Alice -> Peter, $1 (2)
4. Alice -> Jane, $1 (2,3)
5. New Cash -> Peter, $2
6. Peter -> Jane $3 (3,5)

At end Jane has $4, everyone else is broke.

Goes back in history to the beginning of time.

Currently 150Gbytes
Uses of Cryptography

1. Digital signatures to sign a transaction
   - Public key is your account name
   - Private key used to sign a transfer from the account

2. Secure hash to link the blocks

3. Merkle hash tree for efficiency (allows updating the hash based on small changes)

4. Secure hash for proof of work
Digital Signatures

Public Key: account name
Private Key: to sign transactions out of an account
For privacy, can use each “account” just once (to receive and later transmit funds).

E.g.
- [Alice -> Jane, $1]_{Alice}
  - Alice and Jane are public keys
  - Transaction (hash of) is signed by Alice

Uses Elliptic Curve Digital Signature Algorithm (ECDSA)
- Similar to Diffie-Hellman/ElGamal
- Uses elliptic curves as the “group”
  • Points over finite field satisfying: $y^2 = x^3 + ax + b$
Elliptic curve

Addition: (other special cases, e.g. if $Q$ is on tangent)
Linking blocks

Each block is limited to 1Mbyte
Every block has a “coinbase” transactions” that claims the “reward”
Merkle Tree

Can add leaf on right or modify any leaf with cost $O(\log n)$ for updating the root.
Orphan Blocks

![Diagram of blockchain with orphan blocks and transactions]

- Genesis block
- Prev. block: Transactions
- Prev. block: Transactions
- Prev. block: Transactions
- Prev. block: Transactions
- Prev. block: Transactions
- ...
Hashing

Uses SHA256, applied twice
Generates 32 bytes
Similar to private key systems.
Here is one round

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H \\
\end{array}
\]

\[
\begin{array}{cccccccc}
W_t & \downarrow & K_t \\
\end{array}
\]

\[
\begin{array}{cccccccc}
Ch & \rightarrow & & & & & \\
\Sigma 1 & \rightarrow & & & & & \\
Ma & \rightarrow & & & & & \\
\Sigma 0 & \rightarrow & & & & & \\
\end{array}
\]
Mining coins: Proof of Work

Come up with a “nonce” such that the header hashes to a “small number”.
Threshold for the number is set ever 14 days. Set so that new block generated every 10 minutes or so. Seems to require brute force. Currently around $10^{19}$ tries.

Header:

<table>
<thead>
<tr>
<th>version</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous block hash (reversed)</td>
<td>17975b97c18ed1f7e255adf297599b55330edab87803c81701000000000000000</td>
</tr>
<tr>
<td>Merkle root (reversed)</td>
<td>8a97295a2747b4f1a0b3948df3990344c0e19fa6b2b92b3a19c8e6badc141787</td>
</tr>
<tr>
<td>timestamp</td>
<td>358b0553</td>
</tr>
<tr>
<td>bits</td>
<td>535f0119</td>
</tr>
<tr>
<td>nonce</td>
<td>48750833</td>
</tr>
</tbody>
</table>
"Mining" coins: Proof of work

Nonce is only 32-bits, so uses “secondary nonce” which goes into the “coinbase” transaction. Each block generates 12.5 coins (about $150K). New block about every 10 minutes. This is a “BIG” business: $21 Million/day
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

1. Request *ticket-granting-ticket* (TGT)
2. <TGT>
3. Request *server-ticket* (ST)
4. <ST>
5. Request service
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: $\{K_{C,S}\}K_{C,TGS}, T_{C,S}$
5. Client to Server: $A_{C,S}, T_{C,S}$

Possibly repeat
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
Blind Signatures

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

Sounds dangerous, but can be used to give “value” to an anonymous message

- Each signature has meaning:
  $5$ signature, $20$ signature, ...
Blind Signatures

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
  \[ m' = m \cdot r^e \mod n \]
  and sends it to Trent.
  This is called \textbf{blinding} \( m \)
- Trent signs it: \( s(m') = (m \cdot r^e)^d \mod n \)
- Alice calculates:
  \[ s(m) = s(m') \cdot r^{-1} = m^d \cdot r^{ed-1} = m^d \mod n \]
Patented by Chaum in 1990.
An anonymous online scheme

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

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