Large and hitherto uncatalogued life-form on the windscreen, Jim.
Outline

1. Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2. Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
Outline

1. Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2. Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
Outline

1. Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2. Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
What's the idea?

A relatively recent new idea...

In 1976 Diffie and Hellman published the paper “New Directions in Cryptography”, which first introduced the idea of public key cryptography.

Public key cryptography relies on the use of enciphering functions which are not realistically invertible unless you have a deciphering key.

Easy to do one way - hard to do the other way.

The discrete logarithm problem:

easy to calculate \( n = g^k \mod p \) given \( g, k \) and \( p \),
hard to calculate \( k \) in the same equation, given \( g, n \) and \( p \).
Diffie-Hellman key agreement

Two separated users create and share a secret key. A third party is not realistically able to calculate the shared key.

Alice

\[ p, g, a \]

\[ g^a \mod p \]

\[ g^b \mod p \]

Bob

\[ p, g, b \]

\[ g^b \mod p \]

\[ g^a \mod p \]

Ted

\[ p, g \]

\[ g^a \mod p \]

\[ g^b \mod p \]
Knowledge is different

Consider what each participant knows...

- All participants know two system parameters $p$, and $g$
- Alice and Bob each have a secret value (Alice has $a$ and Bob has $b$)
- Alice and Bob each calculate and exchange a public key ($g^a \mod p$ for Alice and $g^b \mod p$ for Bob).
- Ted knows $g$, $p$, $g^a \mod p$ and $g^b \mod p$, but not $a$ or $b$. 
Diffie-Hellman key agreement

So what does each party do?

Both Alice and Bob can now calculate the value $g^{ab} \mod p$.

1. Alice calculates $(g^b \mod p)^a \mod p = (g^b)^a \mod p$.
2. Bob calculates $(g^a \mod p)^b \mod p = (g^a)^b \mod p$.

Shared key is $(g^b)^a \mod p = (g^a)^b \mod p = g^{ab} \mod p$.

Ted has a much more difficult problem.

It is difficult to calculate $g^{ab} \mod p$ without knowing either $a$ or $b$. The algorithmic run-time of the (so-far best) algorithm is:

$$O(e^{c\sqrt{r \log r}})$$

where $c$ is small, but $\geq 1$, and $r$ is the number of bits.
By contrast, the enciphering and deciphering process may be done in $O(r)$.

<table>
<thead>
<tr>
<th>Bit size</th>
<th>Enciphering</th>
<th>Discrete logarithm solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>1,386,282</td>
</tr>
<tr>
<td>1,000</td>
<td>1,000</td>
<td>612,700,000,000,000,000,000,000,000</td>
</tr>
</tbody>
</table>
Outline

1 Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2 Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
Uses of asymmetric encryption

What use is asymmetric encryption?

1. Generating encrypted passwords with 1-way functions
2. Checking integrity by appending digital signature
3. Checking the authenticity of a message.
4. Encrypting timestamps with messages to prevent replay attacks.
5. Exchanging a key.

Note that...

- Participants each have private and public keys
- Keys cannot be derived from each other
Asymmetric encryption

A model for public/private keys

- $K_{pub}$ is public key for Bob, $K_{priv}$ is his private key.
- Only Bob can decrypt.
Encryption - asymmetric systems
Protocols

Public Key Systems
Asymmetric encryption
RSA

Asymmetric authentication

A model for asymmetric authentication

Alice

P
(Plaintext)

J_{priv}

J_{pub}

Alice creates a pair of J keys
Alice uses J_{priv} to encrypt

(Encrypted)

J_{priv} [P]

Harry the hacker

Bob

P
(Plaintext)

J_{pub}

K_{priv}
K_{pub}

Bob decrypts with J_{pub}

J_{priv} is private key for Alice, J_{pub} is her public key.

Only Alice could have encrypted message
Authentication and encryption

Can both authenticate and encrypt...

Making sure of digital signatures. Hash function...
Outline

1. Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2. Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
RSA (Rivest, Shamir, Adelman)

RSA is a well known public key encryption technique:

This public key system relies on the difficult problem of trying to find the complete factorization of a large composite\(^a\) integer whose prime factors\(^b\) are not known.

\(^a\)An integer larger than 1 is called *composite* if it has at least one divisor larger than 1.

\(^b\)The *Fundamental Theorem of Arithmetic* states that any integer \(N\) (greater than 0) may be expressed uniquely as the product of prime numbers.
RSA hacks

How easy is it to crack?

Two RSA-encrypted messages have been cracked:

- The inventors of RSA published a 129-digits (430 bits) RSA public key. In 1994, it was factored with 5000 MIPS-years of computing time.

- A year later, a 384-bit PGP key was cracked. It needed 1300 MIPS-years to factor the key in three months.

Note that these efforts each only cracked a single RSA key.
RSA coding algorithms

The four processes needed for RSA encryption:

1. Creating a public key
2. Creating a secret key
3. Encrypting messages
4. Decoding messages
To create public key $K_p$ and private key $K_s$

**Step 1 - create public key**

1. Select two different large primes $P$ and $Q$.
2. Assign $x = (P - 1)(Q - 1)$. (Does this ring a bell?)
3. Choose $E$ relative prime to $x$.
4. Assign $N = P \times Q$.
5. $K_p$ is $N$ concatenated with $E$.

**Step 2 - create private/secret key**

1. Choose $D$: $D \times E \mod x = 1$ (i.e. multiplicative inverses) or - put another way: $DE = k(P - 1)(Q - 1) + 1$
2. $K_s$ is $N$ concatenated with $D$. 
To encode and decode plain text $m$

**Step 3 - encoding**
1. Pretend $m$ is a number.
2. Calculate $c = m^E \mod N$.

**Step 4 - decoding**
1. Calculate $m = c^D \mod N$.
2. ....WHY?....
Note that it decodes back to $m$

$$c^D \mod N = m^{ED} \mod N$$
$$= m^{k(P-1)(Q-1)+1} \mod PQ$$
$$= m \cdot m^{k(P-1)(Q-1)} \mod PQ$$
$$= m$$

- $m^{P-1} \mod P = 1$, so $(m^{(P-1)})^{k(Q-1)} \mod P = 1$
- $m^{Q-1} \mod Q = 1$, and so (tutorial) $(m^{(P-1)})^{k(Q-1)} \mod PQ = 1$. 
RSA code

Perl script that (kind of) does RSA

```perl
#!/usr/bin/perl -sp0777i<X+d*lMLa^*lN%0]dsXx++lMIN/dsM0<j]dsj
$/=unpack('H*',$_);$_=‘echo 16dio\$k"SK$/SM$n\EsN0p[IN*1
IK[d2%Sa2/d0$^lxp"|dc`;s/\W//g;$_=pack('H*));//((..*))$/

and then

- echo "squeamish ossifrage" | ./rsa.perl -k=10001
  -n=1967cb529 > msg.rsa
- ./rsa.perl -d -k=ac363601 -n=1967cb529 < msg.rsa
```
Consider the Internet - routers and hosts

Lots of opportunity to snoop, modify, DoS, etc etc
Routing and packets...

Internet traffic sent in packets...

Little message
Routing info added (to and from addresses, size, type of message, sequence number)

Lots of opportunity to modify routing information, spoof etc
Outline

1 Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2 Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
Kerberos/Cerberus

Three headed dog which guards the gates of hell...
Kerberos

What is it?

- Network **authentication** protocol.
- Strong authentication for client/server applications using public key cryptography.
- Kerberos is freely available in source form.
- Kerberos is also available in commercial products.
- Client can prove its identity to a server (and vice versa) across an insecure network connection.
Authentication... and encryption.

After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

Must have a Key Distribution Center (KDC)

Kerberos uses Needham-Schroeder protocol.
Sequence of messages for granting tickets

1. Client sends a request to the KDC.
2. KDC authenticates the client and generates a ticket.
3. KDC sends the ticket to the client.
4. Client sends the ticket to the server.
5. Server verifies the ticket and grants access.

Kerberos

Authentication Ticket granting

Server
Client
KDC

Hugh Anderson
CS3235 Seventh set of lecture slides
Kerberos

Sequence of messages for granting tickets

When a client first authenticates to Kerberos, she:

1. Talks to KDC, to get a *Ticket Granting Ticket*
2. Uses that to talk to the *Ticket Granting Service*
3. Uses the ticket, to interact with the server.

This way a user doesn’t have to reenter passwords every time they wish to connect to a Kerberized service. If the Ticket Granting Ticket is compromised, an attacker can only masquerade as a user until the ticket expires.
Kerberos protocol

Two sorts of credentials: tickets and authenticators:

1. A *ticket* $T_{c,s}$ contains the client’s name and network address, the server’s name, a timestamp and a session key. This is encrypted with the server’s secret key (so that the client is unable to modify it).

2. An *authenticator* $A_{c,s}$ contains the client’s name, a timestamp and an optional extra session key. This is encrypted with the session key shared between the client and the server.
Kerberos protocol

Notation

A key $K_{x,y}$ is a session key shared by both $x$ and $y$. When we encrypt a message $M$ using the key $K_{x,y}$ we write it as $\{M\}K_{x,y}$. 
Kerberos protocol

Alice wants session key for communication with Bob:

- Alice sends message to Ted containing her identity, Ted’s TGS identity, and one-time value \( (n) : \{a, tgs, n\} \).
- Ted responds with a key encrypted with Alice’s secret key (which Ted knows), and a ticket encrypted with the TGS secret key:
  \[ \{K_{a,tgs}, n\}K_a \{ T_{a,tgs}\}K_{tgs}. \]
  Alice now has ticket and session key: \( \{T_{a,tgs}\}K_{tgs}, K_{a,tgs} \)
- Alice can prove her identity to the TGS, as she has session key \( K_{a,tgs} \), and Ticket Granting Ticket: \( \{T_{a,tgs}\}K_{tgs} \).
Later, Alice can ask the TGS for a specific service ticket:

- When Alice wants a ticket for a specific service (say with Bob), she sends an authenticator along with the Ticket Granting Ticket to the TGS: \( \{A_{a,b}\}K_{a,tgs} \{T_{a,tgs}\}K_{tgs}, b, n. \)
- The TGS responds with a suitable key and a ticket: \( \{K_{a,b}, n\}K_{a,tgs} \{T_{a,b}\}K_{b}. \)
- Alice can now use an authenticator and ticket directly with Bob: \( \{A_{a,b}\}K_{a,b} \{T_{a,b}\}K_{b}. \)
Properties of the protocol

Host security: Kerberos makes no provisions for host security; it assumes that it is running on trusted hosts with an untrusted network.

KDC compromises: Kerberos uses a principal’s password (encryption key) as the fundamental proof of identity.

Salt: This is an additional input to the one-way hash algorithm.
Outline

1 Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2 Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
A voting protocol is one in which...

- independent systems vote in a kind of election, and
- afterwards we can check that the vote was correct.
- Each voter is only allowed a single vote, and
- the system should be corruption-proof.
A voting protocol...

Example with Alice, Bob and Charles (!), who vote and then encrypt and sign a series of messages using public-key encryption. For example, if Alice votes $v_A$, then she will broadcast to all other voters the message

$$R_A(R_B(R_C(E_A(E_B(E_C(v_A)))))$$

where $R_A$ is a random encoding function which adds a random string to a message before encrypting it with A’s public key, and $E_A$ is public key encryption with A’s public key.
At the end, everybody can check...

Each voter then signs the message and decrypts one level of the encryption.
At the end of the protocol, each voter has a complete signed audit trail and is ensured of the validity of the vote.
Outline

1 Encryption - asymmetric systems
   - Public Key Systems
   - Asymmetric encryption
   - RSA

2 Protocols
   - Kerberos
   - Voting protocols
   - Coin tossing, oblivious transfer and signing contracts
Tossing a coin

Let's pretend we are gambling...

- Alice and Bob want to toss a coin
- Alice calculates two primes $p, q$ and calculates $N = pq$, sends $N$ to Bob. $N = 35 = 5 \times 7$
- If Bob can factorize the number, then Bob wins a coin toss.
- Bob selects random $x$, and sends $x^2 \mod N = y$ to Alice. $y = 31^2 \mod 35 = 16$
Tossing a coin

And then Alice does this:

Alice calculates the four square roots of 16:

1. $4^2 \text{ mbox } 35 = 16$
2. $31^2 \text{ mbox } 35 = 16$
3. $24^2 \text{ mbox } 35 = 16$
4. $11^2 \text{ mbox } 35 = 16$

This is easy for Alice, as she knows the prime factors of $N$. She then sends one of these back to Bob.
So what does Bob do?

- If Bob receives $x$ or $-x$, then he learns nothing, but
- if Bob receives either of the other values:

\[
\text{GCD}(24 + 31, 35) = \text{GCD}(55, 35) = 5
\]

- Alice is unable to tell she has divulged the factor
In an oblivious transfer, randomness is used to convince participants of the fairness of some transaction.

In a coin-tossing example, Alice knows the prime factors of a large number, and if Bob can factorize the number, then Bob wins a coin toss.

A protocol allows Alice to either divulge one of the prime factors to Bob, or not, with equal probability.

Alice is unable to tell if she has divulged the factor, and so the coin toss is fair.
Contract signing

Signing contracts can be difficult:

- If one party signs the contract, the other may not. We have one party bound by the contract, and the other not.
- In addition, both may sign, and then one may say “I didn’t sign any contract!” afterwards.
Use oblivious transfer for contract signing...

Oblivious transfer used for contract-signing where

- Up to a certain point neither party is bound
- After that point both parties are bound
- Either party can prove that the other party signed

Alice and Bob exchange signed messages, agreeing to be bound by a contract with ever-increasing probability.
What if someone tries to terminate early?

- In the event of early termination of the contract, either party can take the messages they have to an adjudicator, who chooses a random probability value (42% say) before looking at the messages.

- If both messages are over 42% then both parties are bound.

- If less then both parties are free.