Announcements:
• HKN surveys on Friday: 5 bonus points for filling out their survey on Friday (yes, that means you have to come to lecture).

Cryptography: Purposes
• Source: Ross Anderson, Security Engineering.
• Cryptography—the study of the design of ciphers—is a tool used to help meet several goals, among them:
  - Privacy: others can't read our messages.
  - Integrity: others can't change our messages without us knowing.
  - Authentication: we know whom we're talking to.
• Some common terminology: we convert from plaintext to ciphertext (encryption) and back (decryption).
• Although we typically think of text messages as characters, our algorithms generally process streams of numbers or bits, making use of standard encodings of characters as numbers.

Substitution
• Simplest scheme is just to permute the alphabet:
  `abcdefghijklmnopqrstuvwxyz`  `tylerduniabcfghjkmopqsvwxz`
• So that "so long and thanks for all the fish" =» "ohtchgutygrtpnygbotdhmtycctpn tdion"
• Problem: If we intercept ciphertext for which we know the plaintext (e.g., we know a message ends with name of the sender), we learn part of the code.
• Even if we have only ciphertext, we can guess encoding from letter frequencies.

Stream Ciphers
• Idea: Use a different encoding for each character position. Enigma was one example.
• Extreme case is the One-Time Pad: Receiver and sender share random key sequence at least as long as all data sent. Each character of the key specifies an unpredictable substitution cipher.
• Example:
  Messages: `attack at dawn` | `oops cancel that order` | `attack is back on`
  Key: `vnchajkrwusiatjcdxtdjahtjedkrijzomajkqliptpyfhadijraqioba...`  `vfhntnkjrtzjgkgrjsjglqpgshlidjachpcookoghanese`
  (key of 'z' means 'a' ↦ 'z', 'b' ↦ 'a', 'c' ↦ 'b', etc.)
• Unbreakable, but requires lots of shared key information.
• Integrity problems: If I know message is "Pay to Paul N. Hilfinger $100.00" can alter it to "Pay to Paul N. Hilfinger $999.00" [How?]

Aside: A Simple Reversible Combination
• The cipher in the last slide essentially used addition modulo alphabet size as the way to combine plaintext with a key.
• Usually, we use a different method of combining streams: exclusive or (xor), which is the "not equal" operations on bits, defined on individual bits by \( x \oplus y = 0 \) if \( x \) and \( y \) are the same, else 1.
  Fact: \( x \oplus y \oplus x = y \).
  `01100011`  `11010110`
  `⊕`  `10110101`  `⊕`  `10110101`
  `11010110`  `01100011`
• In Python, C, and Java, this operation is written \( x \sim y \).

Using Random-Number Generators
• Python provides a pseudo-random number generator (used for the Hog project, e.g.): from an initial value, produces any number of "random-looking" numbers.
• Consider a function that creates pseudo-random number generators that produce bits, e.g.:
  ```python
  import random
def bit_stream(seed):
    r = random.Random(seed)
    return lambda: r.getrandbits(1)
  ```
• If two sides of a conversation share the same key to use as a seed, can create the same approximation to a one-time pad, and thus communicate secretly.
**Block Ciphers**

- So far, have encoded bit-by-bit (or byte-by-byte). Another approach is to map blocks of bits at a time, allowing them to be mixed and swapped as well as scrambled.

- Feistel Ciphers: a strategy for generating block ciphers. Break message into $2N$-bit chunks, and break each chunk into $N$-bit left and right halves, $B_L$ and $B_R$. Then, put the result through a number of rounds:

```plaintext
B_L  B_R

f_1  f_2

B_L  B_R

etc.
```

- Each $f_j$ is some function mapping $N$-bit blocks to $N$-bit blocks that is chosen by your key.
- $f_j$ does not have to be invertible.
- Nice feature: to decrypt, run backwards.
- If the $f_j$ are really chosen well enough, these are very good ciphers with enough rounds.

- The Data Encryption Standard (DES) used this strategy with 12 rounds.

---

**Encryption, Decryption**

<table>
<thead>
<tr>
<th>$i$</th>
<th>$B_L$</th>
<th>$B_R$</th>
<th>$K_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4865</td>
<td>6c6c</td>
<td>K_1</td>
</tr>
<tr>
<td>1</td>
<td>5c32</td>
<td>9a4a</td>
<td>4353</td>
</tr>
<tr>
<td>2</td>
<td>73da</td>
<td>6f72</td>
<td>96a4</td>
</tr>
<tr>
<td>3</td>
<td>13b1</td>
<td>6572</td>
<td>3631</td>
</tr>
<tr>
<td>4</td>
<td>3c52</td>
<td>83b9</td>
<td>8cf7</td>
</tr>
<tr>
<td>5</td>
<td>ba95</td>
<td>6576</td>
<td>5b32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$B_L$</th>
<th>$B_R$</th>
<th>$K_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6f2c</td>
<td>2077</td>
<td>K_1</td>
</tr>
<tr>
<td>1</td>
<td>5c32</td>
<td>8cf7</td>
<td>4353</td>
</tr>
<tr>
<td>2</td>
<td>73da</td>
<td>6f72</td>
<td>96a4</td>
</tr>
<tr>
<td>3</td>
<td>13b1</td>
<td>6572</td>
<td>3631</td>
</tr>
<tr>
<td>4</td>
<td>3c52</td>
<td>83b9</td>
<td>8cf7</td>
</tr>
<tr>
<td>5</td>
<td>ba95</td>
<td>6576</td>
<td>5b32</td>
</tr>
</tbody>
</table>

---

**Public Key Cryptography**

- So far, our ciphers have been symmetric: both sides of a conversation share the same secret information ($a$ key).
- If I haven’t contacted someone before, how can we trade secret keys so as to use one of these methods?
- One idea is to use public keys so that everyone knows enough to communicate with us, but not enough to listen in when others communicate with us.
- Here, information is asymmetric: we publish a public key that everyone can know, and keep back a private key.
- Rely on it being easy to decipher messages knowing the private key, but impractically difficult without it.
- Unfortunately, we haven’t actually proved that any of these public-key systems really are essentially impractical to crack, and quantum computing (if made to work at scale) would break the most common one.
- But for now, all is well.
Example: Diffie-Hellman key exchange

- Assume that everyone has agreed ahead of time about a large public prime number $p$ and another number $g < p$.
- Every person, $Y$, now chooses a secret number, $s_Y$, and publishes the value $Y = g^{s_Y} \mod p$ next to his name.
- If $A$ (Alice) wants to communicate with $B$ (Bob), she can look up Bob's published number, $K_B$, and use $(K_B)^x \mod p$ as the encrypting key.
- Bob, seeing a message from Alice, computes $(K_A)^y \mod p$.
- But $(K_A)^y \equiv (g^{s_A})^y \equiv g^{s_A \cdot y} \equiv (K_B)^y \mod p$, so both Bob and Alice have the same key!
- Nobody else knows this key, because of the difficulty of finding $x$ such that $a^x = b \mod p$ (for large $p$ and $x$).

Other Public-Key Methods

- General idea with public-key methods is that everyone publishes a public key, $K_u$, while retaining a secret private key, $K_v$.
- Typically these keys are very large numbers (hundreds of bits).
- A common method, RSA encryption, uses a public key consisting of the product $pq$ of two large prime numbers and a value $e$ that has no factors in common with $p-1$ and $q-1$. The private key is the two numbers $p$ and $q$.
- It is very hard to compute $p$ and $q$ from the product $pq$.
- To encrypt message $M$, compute $C = M^e \mod pq$.
- It is very hard to compute $M$ from $C$ unless you know $p$ and $q$ (not just $pq$). But it is "easy" (with a computer) if you do know them.
- The method uses Euler's generalization of Fermat's (Little) Theorem, but we'll let you wait until the CS170 series to find out how [plug].

Signatures

- Suppose I receive a message, $M$, that supposedly comes from you. How do I know it does?
- Using public-key methods, this is relatively easy.
- One approach (no details here) is that you first compute a condensation of $M$, $h(M)$, where it is very hard to find another message, $M'$ such that $h(M) = h(M')$ and $h(M)$ is a (big) integer in some limited range (say 128 bits).
- Now append to your message a value $S = f(h(M), K_v)$, where $f$ is a "signing function".
- We choose $f$ so that it has the property that there is an easily computed function $f'$ such that $f'(S, K_v) = h(M)$.
- So I, by computing $h(M)$ and comparing it to $f'(S, K_v)$, can tell whether you signed the message.

Special Effects: Playing Cards Over the Phone?

- How do I play a card game over the phone, so that neither side can (undetectably) cheat?
- To keep it simple, assume we have a two-person game between Alice and Bob where all cards get revealed.
- For each game, let each side choose a secret encryption key, and assume an algorithm that is commutative: if a message is encrypted by secret key $A$ and then by key $B$, it can be decrypted by the two keys in either order.

Playing Cards Over the Phone: Method

- Alice shuffles and encrypts a deck of cards, and sends them to Bob.
- Bob encrypts the encrypted cards, shuffles them, and sends them back to Alice (doubly encrypted).
- Alice deals cards to Bob by selecting and decrypting them, and sending them to Bob, who can decrypt them.
- Alice deals cards to herself by sending them to Bob, having him decrypt them and send them (now singly encrypted) back to Alice.
- At the end of the game, all information can be revealed, and both sides can check for consistency.
Zero-Knowledge Proofs

- Suppose I possess the answer to a puzzle, and want to convince you that I have the answer without revealing anything about what it is.
- This is an example of a zero-knowledge proof (Abadi, Goldwasser, and Rackoff).
- Many uses, such as authentication (I want to prove who I am), or enforcing honesty while maintaining privacy.
- Example: Prove that I know how to 3-color a graph.
- Given a graph (a network of nodes connected by edges) a 3-coloring is an assignment of colors to nodes (from a palette of three) such that no nodes joined by an edge have the same color.
- Don't always exist, and hard to find when they do.
- Can I prove to you that I know how to color a particular large graph without letting you know how?
- Demo: [http://web.mit.edu/~ezyang/Public/graph/svg.html](http://web.mit.edu/~ezyang/Public/graph/svg.html)