CS61A Lecture #39: Cryptography

Announcements:

- Homework 13 is up: due Monday.
- "Homework" 14 will be judging the contest.
- HKN surveys on Friday: 7.5 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
tyler␣duniabcfghjkmopqsvwxz
```

- 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:      vnchkjskruwisn|tjcdktjdjsahtjkdhjrizn|akjqltpotpfhsdjrsqieha...
Cipher:   vfvhmtrkjtzin |gxrvjvjqlwlglqkwgxhlcd|acbqncowkoghuniee
```

  (key of 'z' means 'a' $\mapsto$ 'z', 'b' $\mapsto$ '␣', 'c' $\mapsto$ 'a', 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: $x \oplus y \oplus x = y$. So, $01100011 \oplus 10110101 = 11010110$;
  $11010110 \oplus 10110101 = 01100011$.

Using Random-Number Generators

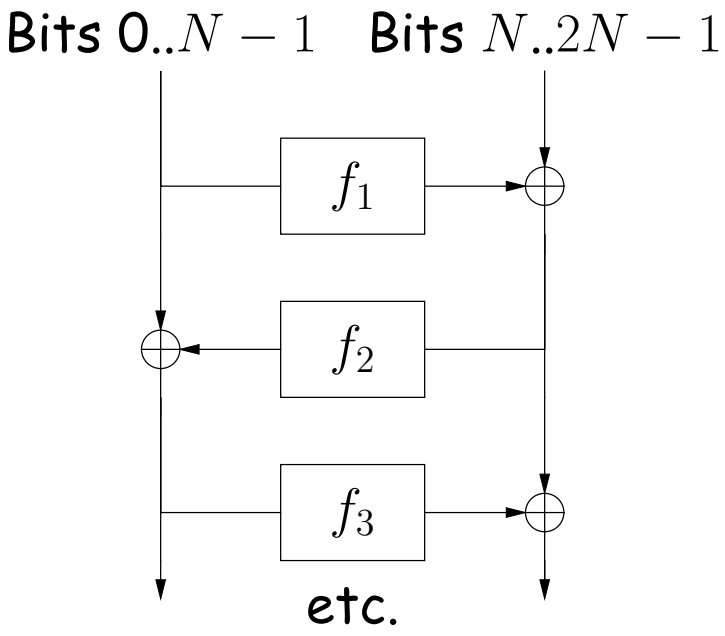
- Python provides a pseudo-random number generator (used for the Pig project, e.g.): from an initial value, produces any number of "random-looking" numbers.
- Consider a function that creates pseudo-random number generators that produce bits, e.g.:

```
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.
- Advantage: key can be much shorter than total amount of data.
- Disadvantage: stream of bits isn't really random; may be subject to clever attack (cryptanalysis).
- Very common technique: used in SSL (Secure Socket Layer) for "secure" web communications.

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. Then, put the result through a number of *rounds:*



  - Each f_i is a "random function" on N-bit blocks chosen by your key.
  - f_i does not have to be invertible.
  - Nice feature: to decrypt, run backwards.
  - If the f_i are really chosen randomly enough, these are very good ciphers with 4 or more rounds.
- The Data Encryption Standard (DES) uses this strategy with 12 rounds.

Chaining

- It's possible to abuse a good cipher, making messages vulnerable.
- If you simply break a message into pieces and then encrypt each piece, an eavesdropper (traditionally named Eve) can tell that two messages you send are the same, even if she doesn't know what the messages are.
- E.g., in advance of the Battle of Midway (WWII), the Allies determined that the target of the Japanese operation was, in fact, Midway by arranging to have the Japanese intercept and retransmit in coded form a message containing the word "Midway." This allowed them to determine what island other encoded Japanese communications were referring to.
- Fix: make every encryption of the same text different using various techniques:
  - Add salt: Intersperse random bits at predetermined locations (ignored on decryption).
  - Chaining: before encrypting a block, xor it with the encoding of the previous block. Start the process off with a throw-away random block.

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.
- 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 $K_Y = g^{S_Y} \bmod p$ 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 $(K_b)^{s_a} \bmod p$ as the encrypting key.
- Bob, seeing a message from Alice, computes $(K_a)^{s_b} \bmod p$.
- But $K_b^{s_a} \equiv (g^{s_b})^{s_a} \equiv g^{s_b \cdot s_a} \equiv (g^{s_a})^{s_b} \equiv (K_a)^{s_b} \bmod p$, so both Bob and Alice have the same key!
- Nobody else knows this key, because of the difficulty of finding x such that $p^x = y \bmod p$.

Other Public-Key Methods

- General idea with public-key methods is that everyone publishes a public key, K_p, while retaining a secret private key, K_s.
- 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 $C = M^e \bmod pq$.
- 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 $S = f(h(M), K_s)$, 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 $f'(S, K_p) = h(M)$.
- So I, by computing $h(M)$ and comparing it to $f'(S, K_p)$, 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.