CS61A Lecture #35: Cryptography

Announcements:

- HKN surveys next 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

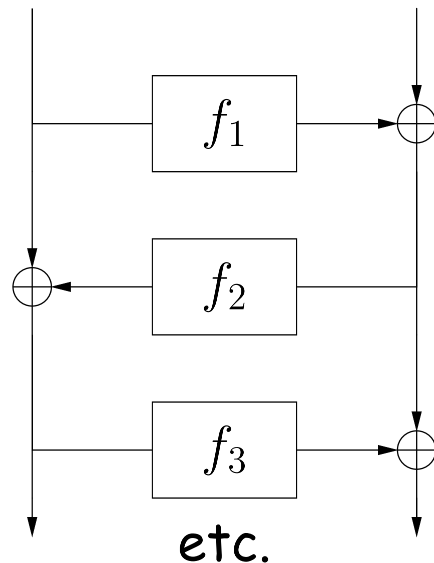
- Python provides a pseudo-random number generator (used for the Pig project, e.g.): from an initial value, produces any number of "random-looking" numbers.
- Consider a function that creates pseudo-random number generators that produce bits, e.g.:

```
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).
- Several possible encrytion modes used in TLS (Transport Layer Security) for "secure" web communications have this form.

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.

A Modern Block Cipher: AES

- Over time, the DES (with 56-bit keys) became increasingly vulnerable to attack.
- In 2001, NIST adopted a new set of ciphers—AES (Advanced Encryption Standard) with key sizes of 128, 192, or 256 bits, and a block size of 128 bits (16 bytes).
- AES approved by the NSA for Top Secret documents.
- The key is first converted into a number of 128-bit (16 byte) “subkeys”, one for each round.
- The number of rounds depends on the key size (10 for AES128, 12 for AES192, 14 for AES256).

An AES Round

Just to give the flavor of the thing, here's a sketch of what a round looks like:

- Arrange the 16 byte block into a 4×4 matrix of bytes.
- Apply a certain fixed function (the "S box") byte-by-byte to the bytes of the matrix.
- Rotate each row of the matrix left by a different amount in each row (0, 1, 2, and 3 positions, respectively).
- Multiply the matrix by a certain fixed matrix. The coefficient arithmetic is in GF(256) (the finite field with 256 elements).
- Add (i.e., xor) the subkey for the round.

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 product pq and a value d that can be computed knowing p, q, and e (specifically, $d \cdot e \equiv 1 \bmod (p-1)(q-1)$).
- It is very hard to compute p, q, or d (all kept secret) 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 d But it is "easy" (with a computer) if you do know:

$$M = C^d \bmod pq.$$

- Why? Uses Euler's generalization of Fermat's (Little) Theorem, if you really must know.

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.

Authentication

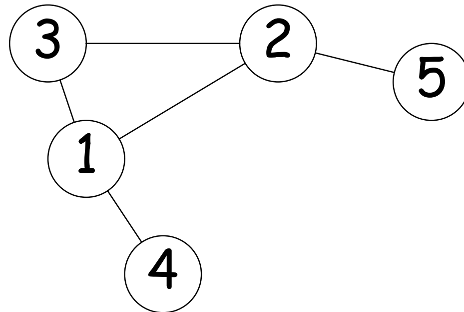
- The particular use of Diffie-Hellman key exchange earlier allowed me to be sure whom I am communicating with.
- Requires that I know a public key of everyone I want to talk to in advance.
- That's a lot of information to keep current (and uncorrupted). Can we cut it down?
- The web uses *certificates* from a *trusted source* for this purpose (in messages to URLs that start with "https").
- Abstractly, a certificate is a message that is signed by some trusted third party (a *certificate authority*, such as VeriSign). Everyone's browser knows the public key needed to validate the signature on an alleged certificate.
- The certificate says (in effect) "entity X (e.g., Amazon.com) has public key K_X."
- As we've seen, you can use that to send communications that only the intended recipient can understand.

Another Kind of Hiding: Zero-Knowledge Proofs

- Idea first put forward in 1985 by Shafi Goldwasser, Silvio Micali, and Charles Rackoff.
- How do I demonstrate to someone that something is true without revealing anything else (like, for example, the proof that it's true)?
- Applications to authentication, enforcing honesty while maintaining privacy, and others.

Example of Zero-Knowledge Proof

- Consider a huge network (graph) of vertices connected by paths:



- Is there a Hamiltonian path in this graph (a path that hits every vertex exactly once)? A very hard problem for large graphs.
- Suppose I know such a path, and want to convince you that I know without letting you know the path.
- You give me a sequence of *challenges* until you are convinced. I could cheat on any challenge, but you would catch me with probability 0.5.
- First, I choose a random renumbering of the graph (1 goes to 4, 2 goes to 3, etc.) and write it down where I can't change it later.
- Next, you randomly ask me to show either (1) that I have properly renumbered the graph, or (2) just the edges of a Hamiltonian path through the renumbered graph.