Early days

- Earliest uses of cryptography were to implement login
  - Systems like UNIX maintained a password file
  - Anyone could read it... but the passwords were in an encrypted form
  - When you logged in, they would compute the encryption of your password and see if it matched the file version
  - If so, allowed you to log in...

Hardware

- These days most computers include "trusted platform modules" or TPMs
  - Special hardware
  - It has a built-in key (we'll see what kind soon)
  - Effectively, the TPM can say "Dell.com vouches for this machine, it's name is Ken's Laptop"

- TPM can do some simple cryptographic operations
  - If widely adopted would result in much better security
  - But in fact not widely used today

The role of cryptography in O/S

- Core questions we've encountered:
  - I claim to be "Ken Birman". But can I prove this?
  - The web site claims to be "M&T Bank.com". But is it?
  - You make a purchase from Amazon.com and need to enter your credit card information. Can spies see it?
  - You and your friend are exchanging some very sensitive email. Can it be kept secret from third parties?
  - On a single machine, O/S provides protection using user/group IDs, permissions, and by ensuring that distinct processes have distinct address spaces

- We tend to turn to cryptographic techniques in networked settings where there are multiple machines

- Several questions arise
  - First, what "tools" can cryptography give us?
  - Then, how can we embed these tools into the network in convenient, safe, secure ways?
  - Finally, what sorts of limitations are we left with?
Basic setup
- We’ll think in terms of situations where there are two processes that need to communicate
- Call them Sally and Ted
- Let’s start by exploring ways that Sally and Ted can share secrets

Symmetric cryptography
- In this approach, Sally starts by creating a secret key and sharing it (somehow) in a secure way with Ted
- They both have the identical key.
- Then we can define some functions in terms of the key

On the Internet
- Encrypted messages look like random bits!
  - An intruder can't make any sense out of them at all
  - A good encryption scheme should have the property that even if you know what the message really says, you can't figure out the key without trying every possible key
- Goal: create a problem that is computationally infeasible today… and will stay that way tomorrow!

Symmetric cryptography
- Encrypt\(_K\) (m): encrypts message m using key K
- Decrypt\(_K\) (m): decrypts message m using key K
- Sign\(_K\) (m): computes a signature for message m
  - This is a short (usually 128 bit) number that is calculated from m and then encrypted with K
  - Uses to detect tampering, or as proof that “Sally saw m”

Asymmetric cryptography
- Also called “public key” cryptography
- A clever scheme that eliminates need to share the key initially
  - In practice a bit slow, so sometimes we start with asymmetric keys and then “exchange” them for symmetric ones
  - This would be one way for our symmetric keys to get shared between Sally and Ted....

Symmetric cryptography
- There are many popular implementations of this kind of cryptographic system
  - For example, US government recommends something called DES, the Digital Encryption Standard
  - For some purposes DES isn’t secure enough, but if you create three keys and apply DES three times, result is very robust (“triple DES”)
  - For signatures, many systems compute an “MD5 hash” and then encrypt it
  - Of course, Sally and Ted still have the problem of creating that initial shared key in a secure way!
Asymmetric cryptography

- Basic idea:
- Sally picks a public key K and a private key K*.
- There is a well known known function crypt s.t.:
  - cryptK (cryptK* (m)) = m
  - cryptK* (cryptK (m)) = m

- She publishes her public key Ksally
- Ted does exactly the same thing, using his own keys

---

RSA implementation?

- Basic idea:
- Sally selects two very big prime numbers p and q
- She computes
  1. A modulus n = p*q
  2. A totient \( \phi(n) = (p-1)*(q-1) \)
  3. She picks an integer e such that 1 < e < \( \phi(n) \), s.t. e and \( \phi(n) \) are coprime (share no divisor other than 1)
  4. She calculates d s.t. \( d*e \equiv 1 \mod \phi(n) \)
- Sally releases her public key as (e, n). She retains d as her private key.

---

Notes

- Notice that encrypt and decrypt are really the same computation but using different keys
  - X = Me mod n, to encrypt
  - M = Xd mod n, to decrypt

- Why does it work?
  - encrypt(decrypt(M)) = M^e*d mod n
  - Theorem (Gauss):
    - If d*e \equiv 1 \mod \phi(n) then (M^d mod n) = (M^e mod n) = M
  - ... hence encrypt(decrypt(M)) = M \text{ qed}

---

Asymmetric cryptography

- Let’s use S for Sally’s public key and \( K^s \) for her private key
- Similarly, T and \( T^t \) for Ted’s key pair
- For Ted to send a secret message m to Sally:
  - Ted computes \( X = \text{crypt}_T (\text{crypt}_S (m)) \)
  - Sally computes \( M = \text{crypt}_K (\text{crypt}_S (X)) \)
- Only Ted could have sent this. Only Sally can read it!
Using asymmetric keys

- Ted can send a message that only Sally can read
  - Just encrypt it with her public key first
  - Ted can send a message that only he can have sent
  - Just encrypt it with his private key first

- Or both.....

- An encrypted hash is often used as a signature

Pros and Cons

- With asymmetric keys one party can easily send things to the other party
  - We do need a way to publish the public information... but this turns out to be reasonably easy

- But these keys are slow (bignum arithmetic...)
  - So a common trick is for Ted to send Sally a proposed symmetric (shared and private) key
  - Once Sally accepts it, she and Ted switch to using that key, with symmetric cryptography, which is very fast

How to share public keys?

- There is an Internet standard for so-called “certificate repositories”
  - A certificate is a signed record that contains cryptographic information, like Sally’s public key
  - Who signs it? The “certificate authority”

- These are built as hierarchies, like the DNS

Trusted Platform Module

- This is one answer to the question... Remember the TPM?
  - What it contains is a private key (burned into hardware)
  - Public key can be obtained from Dell.com

- This lets us imagine software that “can only be executed on Ken’s Laptop” or “an image that Sophie’s Pentax Optio D-60 took in New Orleans at this GPS coordinate on Thursday May 11, 2003...”
  - But as mentioned, not widely used

A Public Key Infrastructure (PKI)

- Your O/S has a root key built in
  - That root “signs” for top-level CA such as Verisign
  - Amazon.com registers their certificate with Verisign

- So when you want to talk to Amazon.com... it tells you to get its certificate from Verisign
- Microsoft says you can trust Verisign... and Verisign gives you the Amazon certificate

What’s in a certificate?

- Name of the entity the key is for
  - Type of key (RSA in our examples)
  - Expiration time
  - Signature of the CA vouching for the certificate
HTTPS runs over a form of secured TCP
- This TCP layer is called the Secure Socket Layer or SSL
- Transport Layer Security, or TLS, has started to replace it

TLS involves three basic phases:
- Peer negotiation for algorithm support
- Key exchange and authentication
- Symmetric cipher encryption and message authentication

Negotiation Step
- The two end points agree on the cryptographic protocol suite they will use
- For example, RSA, Diffie-Hellman, etc
- Idea is to be flexible enough so that a bank, or the military, could use a scheme of its own

Key exchange step
- This works very much as in our examples
  - One peer selects a session key and creates a small certificate for it
    - Includes things like the key, the expiration time, a random number, the identity of the sender
    - Designed to prevent man-in-the-middle or replay attacks
  - Then uses PKI to obtain initial keys
  - Then securely send the certificate for the session key

Symmetric encryption/authentication
- Once the keys are in place, each message sent on the secured TCP connection is
  - Encrypted, to keep the bytes secret
  - Authenticated, to prevent injection of garbage, replay of old messages, etc

If correctly implemented, end-points can be confident that spies and attackers can't disrupt their communication

Common worries about PKIs
- There are actually no widely adopted standards for Ted to talk to Sally!
  - The standard lets Ted talk to Google via gmail
  - And it lets Sally talk to Google
  - But what if Ted and Sally don't trust Google?

- The entire model focuses on trusted vendors
  - Entities who can pay Verisign for certificates...
  - This makes sense for buying products on web sites
  - The right model for things like group collaboration (e.g. in a medical setting) doesn't really exist yet!
Single Sign-On

- A popular refinement
  - Issue: Ted ends up with accounts at 10 different places
  - He wants to sign on once as Ted and have the single sign-on work at all of those accounts

- For example: “MSN Live Passport”

- Idea of Single Sign On is that there can be a company that holds your keys for various sites
  - You log into it once (the single sign-on)
  - And it releases certificates you can use at those sites

So, how good is web security?

- Pretty bad, actually
  - The cryptographic part works fairly well
  - But all the stuff “surrounding” it has weaknesses

- Many machines are vulnerable to viruses that attack with simple things (like buffer overruns) or by exploiting known configuration weaknesses
  - Like standard preset passwords and passwords that are way too easy to guess
  - Some applications can even be tricked into running commands for an intruder! For example via automated patch install scripts...

- More issues
  - Web browsers have many security issues
  - Reflects a tension between wanting browser to be powerful (like able to attach files to email) and wanting it to be secured

- Overwhelming commercial pressures around advertising placement don’t help at all
  - Motivates companies to send you “adware” (== malware that isn’t exactly malicious but definitely isn’t desired!)
  - In-flight modifications of web pages, bad web proxies, other tricks and gotcha’s more and more common...