Some “Big Picture” Issues

Don’t rely on “security through obscurity”

- Easy to learn how locks work
  - Insiders
  - Tinkerers
- Easy to learn how software works
  - Insiders
  - Tinkerers
- Examples: DRM, reverse engineering software patches

Have an open, peer-reviewed (or at least outside expert-reviewed) design

Usability is a major challenge

- Locks:
  - If locks are too complicated, people may not use them
    - But then locks don’t provide any security
  - See Blaze’s “safecracking” paper for an example - class 1 safes are more secure, but have awkward security mechanisms
- Computers:
  - If security mechanisms are too difficult, people won’t use them
  - Example: Personal firewall or antivirus warnings
- Make “secure option” the “default” or “option of least resistance”
Some “Big Picture” Issues

Many potential ways to compromise security

- Physical security
  - Attack locks
  - Attack the door itself
  - Attack windows
  - Hide in bushes
- Computer security
  - Attack the cryptography (if done poorly)
  - Attack the configuration
  - Attack the implementation
  - Attack the user

“Security only as strong as the weakest link”

Systems are complex

Some “Big Picture” Issues

Defense in depth

- Physical world
  - Layers of locks in bank
  - Layers of protection mechanisms around jails
  - Castles: Moats, walls, arrows, ...
- Digital world
  - Same concepts apply

Deterrents

- Physical world
  - Video cameras
  - ADT (home security alarm system)
- Digital world
  - Digital forensics methods

Some “Big Picture” Issues

Not all systems require the same level of security

- Locks
  - Weak locks may be OK to protect you gym cloths
  - But may want stronger locks to protect the contents of your bank’s safes
- Computer security
  - Different assets, adversaries, protection mechanisms

“Security is risk management”

Some “Big Picture” Issues

Packaging (sometimes called “snake oil”)

- Physical world
  - May look secure, but may be easy to circumvent
- Digital world
  - May appear secure, but may actually be very insecure

How is a user supposed to figure out whether something is secure?

Some “Big Picture” Issues

Issues at all phases of development lifecycle

- Physical world
  - Requirements: Master keys (whether to have or not)
  - Design: Master keys (design choices, e.g., master pin depths)
  - Implementation: Lock picking
- Digital world
  - Same issues apply

Better to address security issues as early in the lifecycle as possible

Some “Big Picture” Issues

Denial of service

- Locks
  - Chewing gum
  - Super glue
  - Break a key
- Computers
  - Crash computer, consume resources

Accidents

- Locks
  - Keys on both sides (fire hazard)
- Computers
  - Encrypted filesystem (forget key)
  - ...
Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (maybe)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can "sign" a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

Some "Big Picture" Issues

- Many different adversaries
  - Insiders
  - Ex-insiders (past employees, with copies of keys)
  - Pranksters
  - Outsiders
  - ...

Some "Big Picture" Issues

- Big difference: Connectedness
  - Physical world
    - Not very connected
    - (Yes, some exceptions, e.g., postal system or air travel)
  - Digital world
    - Everyone is everyone else's "neighbor"
    - Plus quite a bit of anonymity

Some "Big Picture" Issues

- Arms race
  - Physical world
    - New lock designs, better safes
  - Digital world
    - New cryptography
    - New software development practices
    - Software updates

Some "Big Picture" Issues

- Big difference: Connectedness
  - Physical world
    - Not very connected
  - Digital world
    - Everyone is everyone else's "neighbor"

Basic Problem

- Given: Everybody knows Bob's public key
  - Only Bob knows the corresponding private key
- Goals: 1. Alice wants to send a secret message to Bob
  - 2. Bob wants to authenticate himself

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: \( p \) and \( g \)
  - \( p \) is a large prime number, \( g \) is a generator of \( Z_p^* \)
    - \( Z_p^* = \{1, 2, ..., p-1\} \) such that \( a \equiv g^b \mod p \)
    - Modular arithmetic: numbers "wrap around" after they reach \( p \)
- Alice:
  - Pick secret, random \( x \)
  - Compute \( g^x \mod p \)
- Bob:
  - Pick secret, random \( y \)
  - Compute \( g^y \mod p \)
- Compute \( k = (g^y)^x = g^{xy} \mod p \)
Why Is Diffie-Hellman Secure?

- **Discrete Logarithm (DL) problem:**
  - Given \( g^x \mod p \), it's hard to extract \( x \)
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- **Computational Diffie-Hellman (CDH) problem:**
  - Given \( g^x \) and \( g^y \), it's hard to compute \( g^{xy} \mod p \)
  - ... unless you know \( x \) or \( y \), in which case it's easy
- **Decisional Diffie-Hellman (DDH) problem:**
  - Given \( g^x \) and \( g^y \), it's hard to tell the difference between \( g^{xy} \mod p \) and \( g^r \mod p \) where \( r \) is random

Properties of Diffie-Hellman

- **Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers**
  - Eavesdropper can’t tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- **Diffie-Hellman protocol (by itself) does not provide authentication**

Requirements for Public-Key Crypto

- **Key generation:** computationally easy to generate a pair (public key \( PK \), private key \( SK \))
  - Computationally infeasible to determine private key \( SK \) given only public key \( PK \)
- **Encryption:** given plaintext \( M \) and public key \( PK \), easy to compute ciphertext \( C=E_{PK}(M) \)
- **Decryption:** given ciphertext \( C=E_{PK}(M) \) and private key \( SK \), easy to compute plaintext \( M \)
  - Infeasible to compute \( M \) from \( C \) without \( SK \)
  - Even infeasible to learn partial information about \( M \)
  - **Trapdoor function:** \( \text{Encrypt}(SK,\text{Encrypt}(PK,M))=M \)

Some Number Theory Facts

- **Euler totient function \( \phi(n) \)** where \( n \geq 1 \) is the number of integers in the \([1,n]\) interval that are relatively prime to \( n \)
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- **Euler's theorem:**
  - if \( a \in Z_* \), then \( a^{\phi(n)}=1 \mod n \)
- **Special case: Fermat's Little Theorem**
  - if \( p \) is prime and gcd\((a,p)=1\), then \( a^{p-1}=1 \mod p \)

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- **Key generation:**
  - Generate large primes \( p, q \)
    - Say, 1024 bits each (need primality testing, too)
  - Compute \( n=pq \) and \( \phi(n)=(p-1)(q-1) \)
  - Choose small \( e \), relatively prime to \( \phi(n) \)
    - Typically, \( e=3 \) or \( e=2^{16}+1=65537 \) (why?)
  - Compute unique \( d \) such that \( ed=1 \mod \phi(n) \)
  - Public key = (\( e,n \)); private key = \( d \)
- **Encryption of \( m \):** \( c=m^e \mod n \)
  - Modular exponentiation by repeated squaring
- **Decryption of \( c \):** \( c^d \mod n = (m^e)^d \mod n = m \)

Why RSA Decryption Works

- \( ed=1 \mod \phi(n) \)
- Thus \( ed=1+k\phi(n)=1+k(p-1)(q-1) \) for some \( k \)
  - Let \( m \) be any integer in \( Z_n \)
  - If gcd\((m,p)=1\), then \( m^{ed}=m \mod p \)
    - By Fermat's Little Theorem, \( m^{p-1}=1 \mod p \)
    - Raise both sides to the power \( k(q-1) \) and multiply by \( m \)
    - \( m^{1+k(p-1)(q-1)}=m \mod p \), thus \( m^{ed}=m \mod p \)
  - By the same argument, \( m^{ed}=m \mod q \)

- Since \( p \) and \( q \) are distinct primes and \( p\cdot q=n \), \( m^{ed}=m \mod n \)
Why Is RSA Secure?

- **RSA problem**: given $n=pq$, $e$ such that $\gcd(e, (p-1)(q-1))=1$ and $c$, find $m$ such that $m^e=c \mod n$
  - i.e., recover $m$ from ciphertext $c$ and public key $(n,e)$ by taking $e$th root of $c$
  - There is no known efficient algorithm for doing this
- **Factoring** problem: given positive integer $n$, find primes $p_1, ..., p_k$ such that $n=p_1^{e_1}p_2^{e_2}...p_k^{e_k}$
- If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring $n$

Caveats

- $e=3$ is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of $c$ to recover $m$
    - Even problems if "pad" $m$ in some ways [Hastad]
  - Let $c = m^3 \mod n$ - same message is encrypted to three people
    - Adversary can compute $m^1 \mod n_1n_2n_3$ (using CRT)
    - Then take ordinary cube root to recover $m$
- Don't use RSA directly

Integrity in RSA Encryption

- Plain RSA does not provide integrity
  - Given encryptions of $m_1$ and $m_2$, attacker can create encryption of $m_1m_2$
    - $(m_1^e) \cdot (m_2^e) \mod n = (m_1m_2)^e \mod n$
  - Attacker can convert $m$ into $m^3$ without decrypting
    - $(m^e)^3 \mod n = (m^3)^e \mod n$
- In practice, OAEP is used: instead of encrypting $M$, encrypt $M \oplus G(r) \oplus r \oplus H(M \oplus G(r))$
  - $r$ is random and fresh, $G$ and $H$ are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - ... if hash functions are "good" and RSA problem is hard

OAEP (image from PKCS #1 v2.1)

Digital Signatures: Basic Idea

- Public key is $(n,e)$, private key is $d$
- To sign message $m$: $s = m^d \mod n$
  - Signing and decryption are the same operation in RSA
  - It's infeasible to compute $s$ on $m$ if you don't know $d$
- To verify signature $s$ on message $m$: $s^e \mod n = (m^d)^e \mod n = m$
  - Just like encryption
  - Anyone who knows $n$ and $e$ (public key) can verify signatures produced with $d$ (private key)
- In practice, also need padding & hashing (why?)

RSA Signatures

- Given: Everybody knows Bob's public key
- Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message
1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key
### Encryption and Signatures

- **Book says:** Encryption and decryption are inverses.
- **That’s a common view**
  - True for the RSA primitive
- **But not one we’ll take**
  - To really use RSA, we need padding
  - And there are many other decryption methods

### Digital Signature Standard (DSS)

- U.S. government standard (1991-94)
  - Modification of the ElGamal signature scheme (1985)
- **Key generation:**
  - Generate large primes $p, q$ such that $q$ divides $p-1$,
    
  \[-2^{160} < q < 2^{160}, \frac{q}{2^{160}} < p < 2^{2161}\text{ where } 0 < cB\]
  - Select $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \mod p$
  - Select random $x$ such $1 \times x \in q - 1$, compute $y = g^x \mod p$
- **Public key:** $(p, q, g, y = g^x \mod p)$, private key: $x$
- **Security of DSS** requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract $x$ (private key) from $g^x \mod p$ (public key)

### DSS: Signing a Message

#### Algorithm:
1. **Message**
2. **Hash function** (SHA-1)
3. **Random secret** between 0 and $q$
4. Compute $r = (g^k \mod p) \mod q$
5. **Private key**
6. Compute $s = k^{-1}((H(M)) + x \cdot r) \mod q$
7. $(r, s)$ is the signature on $M$

### DSS: Verifying a Signature

#### Algorithm:
1. **Message**
2. **Signature**
3. **Public key**
4. Compute $w = s' - 1 \mod q$
5. Compute $(g^{H(M)w} \mod p)^r \mod q$
6. If they match, signature is valid

### Why DSS Verification Works

- **If $(r, s)$ is a legitimate signature,** then
  \[r = (g^k \mod p) \mod q; \quad s = k^{-1}((H(M)) + x \cdot r) \mod q\]
- **Thus $H(M) = -x \cdot r + k \cdot s \mod q$**
  - Multiply both sides by $w = s^{-1} \mod q$
  \[g^{H(M)w} + x \cdot r \cdot w = k \mod q\]
  - Exponentiate $g$ to both sides
  \[(g^{H(M)w})^w + x \cdot r \cdot w = g^k \mod p \cdot q\]
  - In a valid signature, $g^k \mod p \cdot q = r, g^k \mod p = y$
- **Verify** $g^{H(M)w} \cdot y^w = r \mod p \cdot q$

### Security of DSS

- **Can’t create a valid signature** without private key
- **Given a signature**, hard to recover private key
- **Can’t change or tamper with signed message**
- **If the same message is signed twice**, signatures are different
  - Each signature is based in part on random secret $k$
- **Secret $k$ must be different** for each signature!
  - If $k$ is leaked or if two messages re-use the same $k$, attacker can recover secret key $x$ and forge any signature from then on
Advantages of Public-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - No "chicken-and-egg" key establishment problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
    - Caveats to come
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
  - Reduce protection of information to protection of authenticity of public keys
    - No need to keep public keys secret, but must be sure that Alice's public key is really her true public key

Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - We'll see this in IPSec and SSL
- Keys are longer
  - 1024 bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - What if factoring is easy?
    - Factoring is believed to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions)

Authentication of Public Keys

Problem: How does Alice know that the public key she received is really Bob's public key?

Using Public-Key Certificates

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority's signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $\text{sig}_{\text{Verisign}}(\text{"UW"}, \text{PK}_{\text{UW}}); \text{sig}_{\text{UW}}(\text{"Alice"}, \text{PK}_{\text{Alice}})$
  - What happens if root authority is ever compromised?

Hierarchical Approach
Many Challenges

Many Challenges

Alternative: “Web of Trust”

X.509 Authentication Service

X.509 Certificate

Certificate Revocation
Certificate Revocation Mechanisms

- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor
- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a "delta CRL" containing only updates
- Question: does revocation protect against forged certificates?

X.509 Certificate Revocation List

- Certificate Revocation List
- Online revocation service
  - When a certificate is presented, recipient goes to an online revocation service to check whether it is valid
- X.509 Certificate Revocation List
  - Certificates are used across the internet and must be validated
  - Can issue a "delta CRL" containing only updates
- Question: does revocation protect against forged certificates?

X.509 Version 1

- Encrypt, then sign for authenticated encryption
  - Goal: achieve both confidentiality and authentication
  - E.g., encrypted, signed password for access control
- Does this work?

Attack on X.509 Version 1

- Receiving encrypted password under signature does not mean that the sender actually knows the password!

Authentication with Public Keys

- "I am Alice" and fresh random challenge C are authenticated with public keys.
- Only Alice can create a valid signature
- Signature is on a fresh, unpredictable challenge

Potential problem: Alice will sign anything
Early Version of SSL (Simplified)

- **Bob’s reasoning:** I must be talking to Alice because...
  - Whoever signed $N_b$ knows Alice’s private key... Only Alice knows her private key... Alice must have signed $N_b$... $N_b$ is fresh and random and I sent it encrypted under $K_{AC}$... Alice could have learned $N_b$ only if she knows $K_{AB}$... She must be the person who sent me $K_{AB}$ in the first message...

Breaking Early SSL

- **Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob**
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL

- **Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob**
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL

- **Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob**
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL

- **Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob**
  - Information signed by Alice is not sufficiently explicit
### Breaking Early SSL

- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

### Security Evaluation #2

- You’ll be looking at WinZip’s new AE-2 encryption scheme
  - Based on “Encrypt-then-MAC” (recall a few classes ago --- this is a provably secure mode)
  - But things aren’t always that simple
    - Many protocols seem secure but actually have problems
  - Your job: Analyze AE-2

### What is WinZip?


### WinZip encryption

WinZip has the ability to encrypt files. Lots of history, but we’ll look at the AE-2 method.
Zipping a file without AE-2 (high level)

File → Compression Algorithm → Compressed Data

Zipping a file with AE-2 (high level)

File → Compression Algorithm → Archive.zip
Zipping a file with AE-2 (high level)

File → Compression Algorithm → Compressed Data

Header
- compression type
- File date/time
- CRC-32
- Filename

CRC-32 = 0
compression type = AE

Zipping a file with AE-2 (high level)

File → Compression Algorithm → Compressed Data

Header
- compression type
- File date/time
- CRC-32
- Filename

CRC-32 = 0
compression type = AE
Zipping a file with AE-2 (high level)

File → Compression Algorithm

Passphrase → PBKDF
Zipping a file with AE-2 (high level)

File → Compression Algorithm → AES-CTR then HMAC-SHA1

File → Compression Algorithm

Passphrase → PBKDF

Passphrase → PBKDF

Passphrase → PBKDF

Passphrase → PBKDF

Passphrase → PBKDF

Passphrase → PBKDF
Zipping a file with AE-2 (high level)

File → Compression Algorithm → AES-CTR then HMAC-SHA1 → Encrypted and MACed Data

Passphrase → PBKDF → AES-CTR then HMAC-SHA1 → Encrypted and MACed Data

Header
- compression type = AE
- File date/size
- CRC-32 = 0
- Filename
- Version = 2
- Compression type
- Salt
- Key check val
- Encrypted and MACed Data

File date/size
- compression type
- Version = 2
- Compression type
- Salt
- Key check val
- Encrypted and MACed Data