CIS 433/533 - Computer and Network Security
Public Key Crypto/ Cryptographic Protocols

Professor Kevin Butler
Winter 2013
Key Distribution/Agreement

- **Key Distribution** is the process where we assign and transfer keys to a participant
  - Out of band (e.g., passwords, simple)
  - During authentication (e.g., Kerberos)
  - As part of communication (e.g., skip-encryption)

- **Key Agreement** is the process whereby two parties negotiate a key
  - 2 or more participants

- Typically, key distribution/agreement this occurs in conjunction with or after authentication.
  - However, many applications can pre-load keys
Key Distribution

• Say we used pairwise key distribution/agreement in this class (strictly symmetric cryptography)

• **Q: how many key negotiations would there be?**

• 36481 ASes in the Internet: how many negotiations for secure routing solutions?
Diffie-Hellman Key Agreement

• The DH paper really started the modern age of cryptography, and indirectly the security community
  ‣ Negotiate a secret over an insecure media
  ‣ E.g., “in the clear” (seems impossible)
  ‣ Idea: participants exchange intractable puzzles that can be solved easily with additional information

• Mathematics are very deep
  ‣ Working in multiplicative group G
  ‣ Use the hardness of computing discrete logarithms in finite field to make secure
  ‣ Things like RSA are variants that exploit similar properties
Definitions (Num. Theory)

• **Field:** set of numbers closed under addition and multiplication (also associative and commutative)

• **Finite Field:** field with finite elements: a set of numbers modulo $p$

• **Multiplicative Group modulo $p$:** finite field plus multiplication operation (numbers 1, ... $p-1$)

• **Subgroup:** some elements of a group: if group operation applied, element still in subgroup
  
  ‣ e.g., additive group mod 8, set \{0, 2, 4, 6\} forms subgroup
Diffie-Hellman Protocol

• For two participants $p^1$ and $p^2$
• Setup: We pick a prime number $p$ and a base $g (< p)$
  ‣ This information is public
  ‣ E.g., $p=23$, $g=5$
• Step 1: Each principal picks a private value $x (< p-1)$
• Step 2: Each principal generates and communicates a new value

\[ y = g^x \mod p \]

• Step 3: Each principal generates the secret shared key $z$

\[ z = y^x \mod p \]

• Perform a neighbor exchange.
**Diffie-Hellman**

**Params:** $p=23$, $g=5$ (public values)

<table>
<thead>
<tr>
<th></th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private: $x_A = 6$</td>
<td>Private: $x_B = 15$</td>
<td></td>
</tr>
<tr>
<td>$Y_A = g^{x_A} \mod p$</td>
<td>$Y_B = g^{x_B} \mod p$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$= 5^6 \mod 23$</td>
<td>$= 5^{15} \mod 23$</td>
</tr>
<tr>
<td></td>
<td>$= 8$</td>
<td>$= 19$</td>
</tr>
<tr>
<td>$Z = Y_B^{x_A} \mod p$</td>
<td>$Z = Y_A^{x_B} \mod p$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$= 19^6 \mod 23$</td>
<td>$= 8^{15} \mod 23$</td>
</tr>
<tr>
<td></td>
<td>$= 2$</td>
<td>$= 2$</td>
</tr>
</tbody>
</table>
Attacks on Diffie-Hellman

• This is key agreement, not authentication.
  ‣ You really don’t know anything about who you have exchanged keys with
  ‣ The man in the middle …

  ‣ Alice and Bob think they are talking directly to each other, but Mallory is actually performing two separate exchanges

• You need to have an authenticated DH exchange
  ‣ e.g., out of band
D-H Subtleties

• Generator: $g^i$ generates elements in group

• Primitive element: there is some $i$ such that $g$ generates all elements in a group

• Weakness: if $g$ is not a primitive element of the group then only a small subgroup may be generated

• Solution: safe primes

  ‣ prime $p$ of form $2q + 1$ where $q$ prime
  ‣ subgroups now $\{1\}, \{1, p-1\}$, size $q$, size $2q$
  ‣ 2, 5 are good values for generators
Public Key Cryptography

• Each key pair consists of a public and private component: $k^+$ (public key), $k^-$ (private key)

$$D(E(p, k^+), k^-) = p$$
$$D(E(p, k^-), k^+) = p$$

• Public keys are distributed (typically) through public key certificates
  ‣ Anyone can communicate secretly with you if they have your certificate
  ‣ E.g., SSL-based web commerce
RSA (Rivest, Shamir, Adelman)

• A dominant public key algorithm
  ‣ The algorithm itself is conceptually simple
  ‣ Why it is secure is very deep (number theory)
  ‣ Use properties of exponentiation modulo a product of large primes

“A method for obtaining Digital Signatures and Public Key Cryptosystems”,
RSA Key Generation

- Pick two large primes $p$ and $q$
- Calculate $n = pq$
- Pick $e$ such that it is relatively prime to $\phi(n) = (q-1)(p-1)$
  - “Euler’s Totient Function”
- $d \sim e^{-1} \mod \phi(n)$ or $de \mod \phi(n) = 1$

1. $p = 3, \quad q = 11$
2. $n = 3 \times 11 = 33$
3. $\phi(n) = (2 \times 10) = 20$
4. $e = 7 \mid \text{GCD}(20, 7) = 1$
5. “Euclid’s Algorithm”
   - $d = 7^{-1} \mod 20$
   - $d \mid d7 \mod 20 = 1$
   - $d = 3$
RSA Encryption/Decryption

• Public key $k^+$ is $\{e,n\}$ and private key $k^-$ is $\{d,n\}$

• Encryption and Decryption
  $E(k^+, P) : \text{ciphertext} = \text{plaintext}^e \mod n$
  $D(k^-, C) : \text{plaintext} = \text{ciphertext}^d \mod n$

• Example
  ‣ Public key (7,33), Private Key (3,33)
  ‣ Data “4” (encoding of actual data)
    ‣ $E(\{7,33\},4) = 4^7 \mod 33 = 16384 \mod 33 = 16$
    ‣ $D(\{3,33\},16) = 16^3 \mod 33 = 4096 \mod 33 = 4$
RSA Recap

• Pick two primes: $p, q$
• Modulus $n = pq$
• Euler’s totient $\phi(n) = (p-1)(q-1)$
• Public exponent $e$ selected s.t. $\gcd(e, \phi(n)) = 1$
• Private exponent $d = e^{-1} \mod \phi(n)$
• For message $m$, ciphertext $c$, plaintext $p$
  ‣ $c = m^e \mod n$
  ‣ $p = c^d \mod n$
Encryption using private key ...

- Encryption and Decryption

\[
E(k^-, P) : \text{ciphertext} = \text{plaintext}^d \mod n \\
D(k^+, C) : \text{plaintext} = \text{ciphertext}^e \mod n
\]

- E.g.,

- \( E(\{3,45\},4) = 4^3 \mod 33 = 64 \mod 33 = 31 \)
- \( D(\{7,45\},19) = 31^7 \mod 33 = 27,512,614,111 \mod 33 = 4 \)

- Q: Why encrypt with private key?
“Textbook” RSA

• Safe to use?

• NO!!

• “Multiplicative homomorphism”
  ‣ For \( c = E(m) = m^e \mod(n) \)
  ‣ \( E(m_1) \times E(m_2) = E(m_1 \times m_2) \)

• Avoid structure by encoding values with PKCS#1

• Use different exponents for encryption and signing (typically \( e=3 \) for enc., 5 for signing)

• don’t use small \( d \), insecure (small exponent attack)
Common Modulus Attack

• Alice uses $n, e_a$
• Bob uses $n, e_b$ (i.e., share common modulus)
• If $e_a$ and $e_b$ are relatively prime then eavesdropper Eve does the following:

- Use Euclidean algorithm to find $r,s$ where $e_a \cdot r + e_b \cdot s = 1$ (the e values are public)
- Now:
  
  $$(m^{e_a} \mod n)^r \cdot (m^{e_b} \mod n)^s = M^{e_a \cdot r + e_b \cdot s} \mod n = M \mod n$$
Digital Signatures

• Models physical signatures in digital world
  ‣ Association between private key and document
  ‣ … and indirectly identity and document.
  ‣ Asserts that document is authentic and non-reputable

• To sign a document
  ‣ Given document d, private key k-
  ‣ Signature $S(d) = E(k^-, h(d))$

• Validation
  ‣ Given document d, signature $S(d)$, public key $k^+$
  ‣ Validate $D(k^+, S(d)) = H(d)$
Secret vs. public key crypto.

- **Secret key cryptography**
  - Symmetric keys, where a single key (**k**) is used for encryption (E) and decryption (D)
    - \[ D( E( p, k ), k ) = p \]
  - All (intended) receivers have access to key
  - Note: Management of keys determines who has access to encrypted data
    - E.g., password encrypted email
  - Also known as symmetric key cryptography

- **Public key cryptography**
  - Each key pair consists of a public and private component:
    - **k+** (public key), **k-** (private key)
    - \[ D( E(p, k+), k- ) = p \]
    - \[ D( E(p, k-), k+ ) = p \]
  - Public keys are distributed (typically) through public key certificates
    - Anyone can communicate secretly with you if they have your certificate
    - E.g., SSL-based web commerce
Meet Alice and Bob ..... 

• **Alice** and **Bob** are the canonical players in the cryptographic world.
  ‣ They represent the end points of some interaction
  ‣ Used to illustrate/define a security protocol

• Other players occasionally join ...
  ‣ **Trent** - trusted third party
  ‣ **Mallory** - malicious entity
  ‣ **Eve** - eavesdropper
  ‣ **Ivan** - an issuer (of some object)
Some notation ...

• You will generally see protocols defined in terms of exchanges containing some notation like
  ‣ All players are identified by their first initial
    • E.g., Alice=A, Bob=B
  ‣ $d$ is some data
  ‣ $pw^A$ is the password for A
  ‣ $k_{AB}$ is a symmetric key known to A and B
  ‣ $K^+_A, K^-_A$ is a public/private key pair for entity A
  ‣ $E(k,d)$ is encryption of data $d$ with key $k$
  ‣ $H(d)$ is the hash of data $d$
  ‣ $\text{Sig}(K^-_A,d)$ is the signature (using A’s private key) of data $d$
  ‣ “+” is used to refer to concatenation
Some interesting things you want to do …

… when communicating.

- Ensure the **authenticity** of a user

- Ensure the **integrity** of the data
  - Also called **data authenticity**

- Keep data **confidential**

- Guarantee **non-repudiation**
Basic (User) Authentication

• Bob wants to authenticate Alice’s identity
  ‣ (is who she says she is)

\[
[pw^A]
\]

1

Alice

\[[Y/N]\]

2

Bob
Hash User Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)

```
[ h(pw^A) ]
```

1. Alice
2. Bob

[Y/N]
Challenge/Response User Authentication

• Bob wants to authenticate Alice’s identity
  ‣ (is who she says she is)

\[
[h(c+pw^A)]
\]

Alice

Bob

[c]

[Y/N]
User Authentication vs. Data Integrity

• User authentication proves a property about the communicating parties
  ‣ E.g., I know a password

• Data integrity ensures that the data transmitted...
  ‣ Can be verified to be from an authenticated user
  ‣ Can be verified to determine whether it has been modified

• Now, let's talk about the latter, data integrity
Simple Data Integrity?

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)

\[ [d, h(d)] \]
HMAC Integrity

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)

\[ \text{Alice} \rightarrow [d, \text{HMAC}(k,d)] \rightarrow \text{Bob} \]
Signature Integrity

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)

\[ [d, \text{Sig}(K_{A^-}, d)] \]
Data Integrity vs. Non-repudiation

• If the integrity of the data is preserved, is it provably from that source?
  ‣ Hash integrity says what about non-repudiation?
  ‣ Signature integrity says what about non-repudiation?
Confidentiality

• Alice wants to ensure that the data is not exposed to anyone except the intended recipient (confidentiality)

\[ [E(k_{AB}, d), \text{HMAC}(k_{AB}, d)] \]
Question

• If I already have an authenticated channel (e.g., the remote party’s public key), why don’t I simply make up a key and send it to them?
Confidentiality

• Alice wants to ensure that the data is not exposed to anyone except the intended recipient (confidentiality)
• But, Alice and Bob have never met!!!!

\[ [E(k_x, d), \text{hmac}(k_x, d), E(K_{B^+}, k_x)] \]

• Alice randomly selects key \( k_x \) to encrypt with
Real Systems Security

• The reality of the security is that 90% of the frequently used protocols use some variant of these constructs.
  ‣ So, get to know them … they are your friends
  ‣ We will see them (and a few more) over the term

• They also apply to systems construction
  ‣ Protocols need not necessarily be online
  ‣ Think about how you would use these constructs to secure files on a disk drive (integrity, authenticity, confidentiality)
  ‣ We will add some other tools, but these are the basics
Cryptanalysis and Protocol Analysis

• Cryptographic Algorithms
  ‣ Complex mathematical concepts
  ‣ May be flawed
  ‣ What approaches are used to prove correct/find flaws?

• Cryptographic Protocols
  ‣ Complex composition of algorithms and messages
  ‣ May be flawed
  ‣ What approaches are used to prove correct/find flaws?
• Needham-Schroeder Public Key Protocol
  ‣ Defined in 1978

• Assumed Correct
  ‣ Many years without a flaw being discovered

• Proven Correct
  ‣ BAN Logic

• So, It’s Correct, Right?
Needham-Schroeder Public Key

• Does It Still Look OK?

**Nonce**

• Message a.1: \( A \to B : A,B, \{N_A, A\}_{PKB} \)
  ‣ A initiates protocol with fresh value for B

• Message a.2: \( B \to A : B,A, \{N_A, N_B\}_{PKA} \)
  ‣ B demonstrates knowledge of \( N_A \) and challenges A

• Message a.3: \( A \to B : A,B, \{N_B\}_{PKB} \)
  ‣ A demonstrates knowledge of \( N_B \)

• A and B are the only ones who can read \( N_A \) and \( N_B \)
Gavin Lowe Attack

• An active intruder X participates...

• Message a.1: \( A \rightarrow X : A, X, \{N_A, A\}_{PK_X} \)

• Message b.1: \( X(A) \rightarrow B : A, B, \{N_A, A\}_{PK_B} \)
  ‣ X as A initiates protocol with fresh value for B

• Message b.2: \( B \rightarrow X(A) : B, A, \{N_A, N_B\}_{PK_A} \)

• Message a.2: \( X \rightarrow A : X, A, \{N_A, N_B\}_{PK_A} \)
  ‣ X asks A to demonstrate knowledge of \( N_B \)

• Message a.3: \( A \rightarrow X : A, X, \{N_B\}_{PK_X} \)
  ‣ A tells X \( N_B \); thanks A!

• Message b.3: \( X(A) \rightarrow B : A, B, \{N_B\}_{PK_B} \)
  ‣ X completes the protocol as A
What Happened?

- X can get A to act as an “oracle” for nonces
  - Hey A, what’s the $N_B$ in this message from any B?

- A assumes that any message encrypted for it is legit
  - Bad idea

- X can enable multiple protocol executions to be interleaved
  - Should be part of the threat model?
Dolev-Yao Result

• Strong attacker model
  ‣ Attacker intercepts every message
  ‣ Attacker can cause one of a set of operators to be applied at any time
    • Operators for modifying, generating any kind of message
  ‣ Attacker can apply any operator except other’s decryption
• Common model to show security against
Basic truths of cryptography ...

- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
    - Use of crypto commonly is … (e.g., WEP)
  - Vetted through crypto community
  - Avoid any “proprietary” encryption
  - Claims of “new technology” or “perfect security” are almost assuredly snake oil
Why Cryptosystems Fail

• Typically, not because of crypto algorithms
ATMs

• Consider ATM systems
  ‣ some public data, also some high value information
  ‣ banks tend to be interested in security

• How do they work?
  ‣ Card: with account number
  ‣ User: provides PIN
  ‣ ATM: verifies that PIN corresponds to encryption of account number with PIN key (offset can be used)

• Foundation of security: PIN key
  ‣ This is a trusted part of the system
ATM Fraud

• Insiders
  ‣ Make extra card; special ops allow debit of any account

• Outsiders
  ‣ Shoulder surfing; fake ATMs, replay pay response

• PINs
  ‣ Weak entropy of PIN keys; limit user PIN choices; same PIN for everyone

• User-chosen PINs
  ‣ Bad; Store encrypted in a file (find match); Encrypted on card
Fake & Compromised ATMs

• China, 2010: fake ATMs that look real but give error messages when a card put in
• Russia: March 2009
  • criminals install a Trojan on Diebold ATMs
  • Skimmer-A code steals PINs from cards entered in machine and skims money in a variety of currencies
More Complex Issues

• PIN key derivation
  ‣ Set terminal key from two shares
  ‣ Download PIN key encrypted under terminal key

• Other banks’ PIN keys
  ‣ Encrypt ‘working keys’ under a zone key
  ‣ Re-encrypt under ATM bank’s working key

• Must keep all these keys secret
Product Insecurity

• Despite well understood crypto foundations, products don’t always work securely
  ‣ Lose secrets due to encryption in software
  ‣ Incompatibilities (borrow my terminal)
  ‣ Poor product design
    • Back doors enabled, non-standard crypto, lack of entropy, etc.
  ‣ Sloppy operations
    • Ignore attack attempts, share keys, procedures are not defined or followed
  ‣ Cryptanalysis sometimes
    • Home-grown algorithms!, improper parameters, cracking DES
Problems

• Systems may work in general, but...
  ‣ Are difficult to use in practice
  ‣ Counter-intuitive
  ‣ Rewards aren’t clear
  ‣ Correct usage is not clear
  ‣ Too many secrets ultimately

• Fundamentally, two problems
  ‣ Too complex to use
  ‣ No way to determine if use is correct
Solutions?

• Suggestions from Anderson:
  ‣ Determine exactly what can go wrong
    • Find all possible failure modes
  ‣ Put in safeguards
    • Describe how preventions protect system
  ‣ Correct implementation of safeguards
    • Implementation of preventions meets requirements
  ‣ Decisions left to people are small in number and clearly understood
    • People know what to do

• These are general problems of security!
System Design Principles

• Don’t design your own crypto algorithm
  ‣ Use standards whenever possible

• Make sure you understand parameter choices

• Make sure you understand algorithm interactions
  ‣ E.g. the order of encryption and authentication (in certain instances, authenticate then encrypt is risky)

• Be open with your design
  ‣ Solicit feedback
  ‣ Use open algorithms and protocols
  ‣ Open code? (jury is still out)
Common issues that lead to pitfalls

- Generating randomness
- Storage of secret keys
- Virtual memory (pages secrets onto disk)
- Protocol interactions
- Poor user interface
- Poor choice of key length, prime length, using parameters from one algorithm in another