CSE543 - Introduction to Computer and Network Security

Module: Applied Cryptography

Professor Trent Jaeger
Public 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-base web commerce
Public Key Crypto Uses

• Public key cryptography algorithms have different uses
  ‣ Unlike secret key cryptography and hash functions that all achieve the same thing
  ‣ RSA, ECC - encryption and digital signatures
  ‣ ElGamal, DSS - digital signatures
  ‣ Diffie-Hellman - establish shared secret
  ‣ Zero-knowledge proof systems - authentication

• Common feature
  ‣ Pair of related keys, one secret and one public
Trapdoor Function

• All public-key algorithms rely on *trapdoor functions*
  ‣ $f$ is a trapdoor function if
    • $y = f(x)$ is easy to compute (by anyone) given public $x$, but $x = f^{-1}(y)$ is computationally infeasible (*One-way*)
    • $x = f^{-1}(y)$ is easy to compute given some secret information (known as the *trapdoor*)

• Q. Are hash functions trapdoor? One-way?
• Q. Are MAC functions trapdoor? One-way?
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
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
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., $p=13$, $g=4$

• 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.
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
  ‣ The parties sign the exchanges (more or less)
  ‣ See Schneier for a intuitive description
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

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”
- \( d \approx e^{-1} \mod \phi(n) \) or \( de \mod \phi(n) = 1 \)

1. \( p=3, q=11 \)
2. \( n = 3\times11 = 33 \)
3. \( \phi(n) = (2\times10) = 20 \)
4. \( e = 7 \mid \gcd(20,7) = 1 \)

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)

  ‣ $E(\{7,33\},4) = 4^7 \mod 33 = 16384 \mod 33 = 16$
  ‣ $D(\{3,33\},16) = 16^3 \mod 33 = 4096 \mod 33 = 4$
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: What is RSA’s trapdoor function and trapdoor?

• Q: Why encrypt with private key?
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 \( D(k^+, S(d)) = H(d) \)
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?
Cryptanalysis of RSA

• Survey by Dan Boneh
  ‣ http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html
  ‣ Real heavy math

• Results
  ‣ Fascinating attacks have been developed
  ‣ None devastating to RSA

• Cautions
  ‣ Improper use
  ‣ Secure implementation is non-trivial
Cryptanalysis of RSA

• Premise: Breaking RSA == Factoring Large Integers
  ‣ Factoring Large Integers is Hard
  ‣ \( N = pq \); if \( N \) is known, can we find \( p, q \)?

• Some Known (to cryptanalyst)
  ‣ If \( (p-1) \) is product of prime factors less than \( B \)
  ‣ \( N \) can be factored in time less than \( B^3 \)

• Best Known Approach: General Number Field Sieve
  ‣ Significant early application by Arjen Lenstra
Cryptanalysis of RSA

• Fundamental tenet of cryptography
  ‣ Lots of smart people have tried but not (yet) figured out how to break RSA => RSA is secure

• RSA Laboratories challenge (Mar 1991)
  ‣ Factor $N$ into semiprimes (vary from 100 to 619 decimal digits).
  ‣ Challenge ended in 2007
    • 16 of 54 listed numbers were factored, people still trying
  ‣ Current: up to 232 decimal digits factored
    • Using variations of sieve algorithms
Misuse of RSA

• Common Modulus Misuse
  ‣ Use the same N for all users
  ‣ Since all have a private key for same N
    • Anyone can factor
    • Exposing d is same as factoring N

• Blinding Misuse
  ‣ Suppose adversary wants you to
    • Sign an arbitrary message M
  ‣ You don’t sign
  ‣ Adversary generates innocent $M’$
    • Where $M’ = r^e \cdot M \mod N$
    • Adversary can generate M signature from $M’$ signature
RSA Exponent Problems

• Small Private Exponent
  ‣ Speeds decryption time

• However, Known Attacks Exist on Small Private Keys
  ‣ Due to Mike Wiener, can recover private key
  ‣ Result: If N is 1024 bits, d of private key must be at least 256 bits
  ‣ Some workarounds are known (e.g., based on Chinese Remainder Theorem), but not proven secure

• Small Public Exponent
  ‣ Speed signature verification time
  ‣ Smallest possible value is 3, but recommend $2^{16} + 1$
  ‣ Can recover $M$ encrypted with multiple, small public keys
  ‣ Can recover private key from small public + bits of private
Timing Attacks

• Use timing behavior of system to extract secret [Kocher]
  ‣ Time for repeated squaring (algorithm used to encrypt/decrypt using private key) depends on bits of private key

• Suppose a smartcard stores your private key
  ‣ By precisely measuring the time it takes to perform private key ops, we can recover the key
  ‣ At most $2^n$ operations required, where $n$ is the number of bits in the key

• Attack summary
  ‣ Adversary asks smartcard to generate signatures on several messages
  ‣ Recover one bit at a time starting with least significant
  ‣ Compare times to those measured offline
Timing Attacks

• Boneh and Brumley (2003) showed how to derive RSA private keys from the OpenSSL library implementation
  ‣ No need for local access!

• Solution to timing attacks: blinding!
Power Analysis Attacks

• Also, Discovered by Kocher
  ‣ Power usage is higher than normal in these computations
  ‣ Measure the timing of high power consumption

• Simple Power Analysis
  ‣ Direct interpretation of power measurements
  ‣ Reveals instructions executions
  ‣ Some crypto ops may be sensitive to data, e.g., DES S-boxes

• Differential Power Analysis
  ‣ Statistical analysis of power data correlations

• Solution: Gotta change the code
Power and Timing

• What is the threat model in power/timing attacks?
• How does this conflict with the trust model?
• What is the vulnerability?
Review: secret vs. public key crypto.

- Secret key cryptography
  - Symmetric keys, where a single key (k) is used is used for E and 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
The symmetric/asymmetric key tradeoff

• Symmetric (shared) key systems
  ‣ Efficient (Many MB/sec throughput)
  ‣ Difficult key management
    • Kerberos
    • Key agreement protocols

• Asymmetric (public) key systems
  ‣ Slow algorithms (so far …)
  ‣ Easy (easier) key management
    • PKI - public key infrastructures
    • Webs of trust (PGP)
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)
Basic (User) Authentication

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

\[ pw^A \]

Alice \[ \rightarrow \] Bob

Basic (User) Authentication

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

\[ [pw^A] \]

1

Alice

2

Bob

\[ [Y/N] \]
Hash User Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)
Bob wants to authenticate Alice’s identity
  ‣ (is who she says she is)
Hash User Authentication

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

[Diagram showing Alice and Bob with a hash function, \( h(pw^A) \), and responses [Y/N].]
• Bob wants to authenticate Alice’s identity
  ‣ (is who she says she is)
Challenge/Response User Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)
Challenge/Response User Authentication

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

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

Alice → Bob

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

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

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

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

\[ Y/N \]
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)
Simple Data Integrity?

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

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

\[ [d, \text{hmac}(k, d)] \]
• 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?
  ‣ HMAC 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
Key Distribution Revisited

• How do we distribute a key in an untrusted network?
  ‣ Diffie-Hellman
    • Beware of Man-in-the-Middle Attacks
  ‣ Public key
    • Can also run into Man-in-the-Middle Attacks
      ‣ Tell you how in a minute
  ‣ Symmetric key
    • Offline
    • How about online?
Needham-Schroeder

• Goal
  ‣ Two parties want to communicate securely

• Threat Model
  ‣ Network is untrusted
  ‣ Other nodes may be untrusted

• Requirements
  ‣ Mutual Authentication
  ‣ Prove that only the appropriate parties hold secrets

• Assumptions
  ‣ Trusted Third Party
N-S Protocol

- For Symmetric Key Cryptosystems
N-S Protocol detail

• Message 1: A --> S : A, B, NA
  ‣ A asks TTP S for a session key for A and B to use

• Message 2: S --> A : {NA, B, KAB {KAB, A}BS }AS
  ‣ S returns messages for A that includes the session key
  ‣ And a message for A to give to B

• Message 3: A --> B : {KAB, A}BS
  ‣ A passes “ticket” on to B

• Message 4: B --> A : {NB}AB
  ‣ B asks A to demonstrate knowledge of KAB through NB

• Message 5: A --> B : {NB-1}AB
  ‣ A does!
A Protocol Story

• 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?

- Message a.1: A --> B : A,B, \{N_A, A\}_{PKB}
  - A initiates protocol with fresh value for B

- Message a.2: B --> A : B,A, \{N_A, N_B\}_{PKA}
  - B demonstrates knowledge of N_A and challenges A

- Message a.3: A --> 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 demonstrates 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?
The Fix

• It’s Trivial (find it)

• Message a.1:  A --> B : A,B, \{N_A, A\}_{PK_B}
  ‣ A initiates protocol with fresh value for B

• Message a.2:  B --> A : B,A, \{N_A, N_B, B\}_{PK_A}
  ‣ B demonstrates knowledge of N_A and challenges A

• Message a.3:  A --> B : A,B, \{N_B\}_{PK_B}
  ‣ A demonstrates knowledge of N_B
Impact on Protocol Analysis

• Protocol Analysis Took a Black Eye
  ‣ BAN Logic Is Insufficient
  ‣ BAN Logic Is Misleading

• Protocol Analysis Had Become a Hot Topic
  ‣ Lowe’s FDR
  ‣ Meadow’s NRL Analyzer
  ‣ Millen’s Interrogator
  ‣ Rubin’s Non-monotonic protocols
  ‣ ....

• In the end, could find known flaws, but...
  ‣ attacker model is too complex
Dolev-Yao Result

• Strong attacker model
  ‣ Attacker intercepts every message
  ‣ Attacker can cause operators to be applied at any time
    • Operators for modifying, generating any kind of message
  ‣ Attacker can apply any operator except other’s decryption

• Theoretical Results
  ‣ Polynomial Time for One Session
  ‣ Undecidable for Multiple Sessions
  ‣ **Moral:** Analysis is Difficult Because Attacker Can Exploit Interactions of Multiple Sessions

• End Result: Manual Induction and Expert Analysis are the main approaches.
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 semester

• 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