Intermediate Protocols
Part I

Lecture 2

Schneier: Ch 2, 3, 4
Big Picture

1. Encrypted
2. Building Blocks
3. ZK Proofs
4. Private Info Retrieval
5. Digital Payments
6. Anonymous Channels
7. Network Security
8. Web Privacy

Want access to data
Lecture Outline

- Crypto Zoology
- Intermediate Protocols
Crypto Zoology

- Building blocks that we will use in the rest of this class
- Will describe in more detail in later lectures
- Encryption
- Signature
- Hash
Basic Terminology

- Plaintext
  - Original message

- Ciphertext
  - Coded message

- Cipher or Encryption Algorithm
  - Algorithm for transforming plaintext to ciphertext

- Key
  - Info used in cipher known only to sender/receiver
Basic Terminology (cont’d)

- Encrypt (encipher)
  - Converting plaintext to ciphertext

- Decrypt (decipher)
  - Recovering plaintext from ciphertext
Basic Terminology (cont’d)

- Cryptography
  - Study of encryption principles/methods

- Cryptanalysis (codebreaking)
  - Study of principles/ methods of deciphering ciphertext *without* knowing key

- Cryptology
  - Field of both cryptography and cryptanalysis
Symmetric Cryptosystems

Encryption Key

Plaintext

Encryption Algorithm

Ciphertext

Decryption Algorithm

Decryption Key

Plaintext
Requirements

1. Strong encryption algorithm
2. Secret key known only to sender / receiver
   - Mathematically:
     \[
     \text{Ciphertext} = E(K, \text{Plaintext}) = E_K(\text{Plaintext})
     \]
     \[
     \text{Plaintext} = D(K, \text{Ciphertext}) = D_K(\text{Ciphertext})
     \]
3. Assume encryption algorithm is known!
4. Assume a secure channel to distribute key
In Real Life

Alice

K - secret

C = E(K, M)

Intercept

Bob

K - secret

M = Hi, A, B, “attack tomorrow”

Cannot Decrypt C!

Cannot Produce C’!

C’ = E(K, “Hi, A, B, postpone attack”)
Public Key Cryptography

Encryption Key

Decryption Key

Plaintext

Encryption Algorithm

Ciphertext

Decryption Algorithm

Plaintext
Public Key Crypto (PKC)

- Mathematically:
  
  \[
  \text{Ciphertext} = E(\text{pubKey}, \text{Plaintext})
  \]

  \[
  \text{Plaintext} = D(\text{privKey}, \text{Ciphertext})
  \]
PKC in Real Life!

1. $C = \text{Encrypt}(\text{pubKey}_B, M)$

2. **Intercept $C$**

   - **Malory**: Cannot Infer $\text{privKey}_B$ from $\text{pubKey}_B$!
   - **Bob**: $\text{pubKey}_B$ - public
     - $\text{privKey}_B$ - private
     - $M = \text{Decrypt}(\text{privKey}_B, C)$
   - **Alice**: Has message $M$
   - **Malory**: Cannot Obtain $M$!
Crypto Zoology

- Building blocks that we will use in the rest of this class
- Will describe in more detail in later lectures
  - Encryption
  - Signature
  - Hash
Digital Signatures

- Why do we need digital signatures?
  - Verify author of message
  - Authenticate message contents
- Verifiable by third parties
  - To resolve disputes
Digital Signature Model

Plaintext

Signature Algorithm

Signature

Verification Algorithm

Valid!

Invalid!

Private Key

Public Key
In Real Life

Alice

Bob

Malory

pubKey_B - public

privKey_B - private

\( M = I \text{ owe Alice } \$1000 \)

\( S = \text{Sign}(M, \text{privKey}_B) \)

Intercept

Verify(\( M, S, \text{pubKey}_B \)) = true!

\( M' = I \text{ owe Malory } \$1000 \) for same \( S \)

\( M' = I \text{ owe Malory } \$1000 \) for new \( S' \)

Bob’s Signature
In Real Life

Bob Cannot Deny
Signature $S$!

$S = \text{Sign}(M, \text{privKey}_B)$

$M = \text{I owe Alice $1000}$

Verify$(M, S, \text{pubKey}_B) = \text{true}!$
Digital Signature Requirements

- Depend on the message signed
- Use information unique to sender
  - Prevent both forgery and denial
- Easy to generate
- Easy to verify
- Computationally infeasible to forge
  - New message for existing digital signature
  - Fraudulent digital signature for given message
Crypto Zoology

- Building blocks that we will use in the rest of this class
- Will describe in more detail in later lectures
- Encryption
- Signature
- Hash Functions
Hash Functions

- Condenses message $M$ to fixed size
  - $h = H(M)$

- Assume hash function is public
- Used to detect changes to message

$M$ ($L$ bits) → Hash $H$ → Hash value $h$ (fixed length)
Hash Properties

- **Pre-image resistance:**
  - Given value \( h \), hard to find message \( M \) such that \( h = H(M) \)

- **Second pre-image resistance:**
  - Given message \( M_1 \), hard to find \( M_2 \) such that \( H(M_1) = H(M_2) \)

- **Collision resistance:**
  - Hard to find any \( M_1 \) and \( M_2 \) such that \( H(M_1) = H(M_2) \)
Example: MD5

\[
\text{md5\_digest("The quick brown fox jumps over the lazy dog") = 9e107d9d372bb6826bd81d3542a419d6}
\]

\[
\text{md5\_digest("The quick brown fox jumps over the lazy cog") = 1055d3e698d289f2af8663725127bd4b}
\]
Hashes to (not) use

- **MD5**
  - Output 128-bit
  - Designed by Ron Rivest, 1991
  - Wang et. al.: collision in 1 hr using cluster (2004)
  - Klima: collision with 1 min on laptop (2006)

- **SHA-1**
  - Output 160-bit
  - Designed by NSA
  - ”broken” by Wang et. al. – attack requires $< 2^{69}$ ops to find collision (exhaustive would take $2^{80}$) (2005)
Example: MD5

- Do not use at all the following:
  - MD5, SHA-0/1, any other obscure “secret” ones

- For use in civilian/.com setting (until 2010/15):
  - SHA-256/512
Lecture Outline

- Project Details
- Crypto Zoology
- Intermediate Protocols
Intermediate Protocols

- Secret Splitting
- Secret Sharing
- Bit Commitment
- Fair Coin Flips
- ...

Security & Privacy Protocols

Intermediate Protocols

Cryptography
Secret Splitting

- Have a secret – cannot trust one person with it
  - Nuclear launch codes
  - Secret recipe/formula
  - ...  
- Split the secret in multiple (N) shares
  - Give each person a share
  - N-1 or less shares cannot reconstruct the secret
  - N shares can
Secret Splitting: How?

- Secret $S$; split between two persons
  - 1$^{st}$ share: $S_1 = \text{Random } R$
  - 2$^{nd}$ share: $S_2 = S \oplus R$
- Any share is useless
- Need to combine shares to get $S$; How?
- $S_1 \oplus S_2 = S \oplus R \oplus R = S$
Secret Splitting: Generalization

- Secret S; split between N persons
  - 1\textsuperscript{st} share: $S_1 = \text{Random } R_1$
  - 2\textsuperscript{nd} share: $S_2 = \text{Random } R_2$
  - ...
  - N-1\textsuperscript{st} share: $S_{N-1} = \text{Random } R_{N-1}$
  - N\textsuperscript{th} share: $S_N = S \text{ xor } R_1 \text{ xor } R_2 \text{ xor } ... \text{ xor } R_{N-1}$

- Any share is useless
- Need to combine shares to get S; How?
- $S_1 \text{ xor } S_2 \text{ ... xor } S_N = S$