CS 352
Network Security: Introduction

CS 352, Lecture 25.1
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana
Security and the Network Stack

Security: cuts across all parts of the network stack!
Why network security?

• The Internet is used for all sorts of things
  • Banking and commerce
  • Interconnecting electronic voting machines
  • Interacting with the Government, your employer, school, …
  • Shopping online, including essentials like milk or groceries
  • Sometimes, even basic social interactions require the Internet!

• But malicious people share your network
  • People who want to snoop, pretend, steal

• “Attacks” can be passive or active
  • Sit and snoop (e.g., credit card info)
  • Actively target (e.g., phishing)
Some key aspects of network security

Confidentiality: only the sender and the intended receiver should understand the message contents

Integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Authentication: confirm the identity of communicating parties

Non-repudiation: Once someone sends a message, or conducts a transaction, they can’t later deny the contents of that message

Availability: sender and receiver able to communicate at all
Friends and enemies: Alice, Bob, Trudy

• Two parties, Bob and Alice, want to communicate securely
  • Often used in network security examples
• Trudy (intruder) may intercept, delete, add messages
Who/what might Bob and Alice be?

- Real humans 😊
- Web browser/server for electronic transactions
  - e.g., on-line purchases, or online banking
- DNS clients and servers
- Routers exchanging routing table updates
- Two mail clients

- Many other examples!
What might Trudy do?

- **Eavesdrop**: intercept messages
- **Entity in the middle**: actively insert messages into connection
- **Impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **Hijacking**: “take over” ongoing connection by removing sender or receiver, inserting itself in place
- **Denial of service**: prevent service from being used by others (e.g., by overloading resources)
What we will learn in the next lectures

• Principles of network security
  • Primitives for confidentiality, authentication, integrity, non-repudiation

• How to apply these principles to secure:
  • An application: e-mail
  • Transport: TLS (Transport Layer Security for TCP)
Network security is a broad area

• Many exciting topics!
• Security for apps and transport protocols: e.g., QUIC
• Security at all layers: Network layer (e.g., IPSec, VPNs); Link layer (e.g., WPA)
• Security for protocols, e.g., DNSSEC, BGPSEC
• Operational security: how to secure a network
  • Firewalls, intrusion detection/prevention, data breach security, …
• Covering these and other topics in network & system security would require its own set of courses 😊
CS 352
Cryptography: Introduction

CS 352, Lecture 25.2
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana
Confidentiality

- **Confidentiality**: only the sender and the intended receiver should understand the message contents.
- How to achieve this goal?
  - Cryptography
- Sender encrypts a message, receiver decrypts it.
- An intermediate observer should just see random bytes!
m: plaintext message
$K_A$, Alice’s encryption key. Secret known only to Alice
$K_A(m)$ is ciphertext: m encrypted with key $K_A$
Encryption transforms the message so that it’s jumbled
Ideal: want $K_A(m)$ to be uncorrelated with m (Trudy can’t read the msg)
Terminology of Cryptography

K_B is Bob’s decryption key, a secret known only to Bob
m' = K_B(c), c decrypted with key K_B. K_B(c) is plaintext
Want Bob to retrieve the same plaintext as the one sent by Alice
Want m = K_B(K_A(m))
Encryption and decryption algorithms are also called ciphers.
Algorithms and Keys

• Cryptography requires **algorithms** (for encryption and decryption) and **keys** (parameters fed to the algorithms)

• Cryptography practice: **algorithms must be publicly known**
  • Inspires trust that it works: obvious flaws found sooner
  • Openness fosters innovation: techniques can be improved by everyone

• On the other hand, **keys are secret**
  • Keys must be hard to guess, e.g., 128-bit, 256-bit, 1024-bit

• Analogy: everyone knows how your house lock works, and they use a similar design for their house lock
  • “Everyone uses the same lock, so it must be a reliable lock”
  • But only you know the combination for your lock
Two kinds of cryptography

• $K_A$ and $K_B$ are the same: **symmetric key cryptography**
  • Next module

• $K_A$ and $K_B$ are different: **public key cryptography**
  • Next lecture!
Symmetric Key Cryptography

- Alice and Bob use the same (symmetric) key, $K_S$
- Abuse notation: $K_S(m)$ at Alice’s side is encryption, $K_S(c)$ at Bob’s side is decryption
- $m = K_S(K_S(m))$
- Techniques of symmetric key crypto: substitution and permutation
Substitution-based ciphers

• Monoalphabetic cipher: substitute one letter for another
• Example 1: **Caesar cipher.** Replace each letter by letter shifted by some number of characters in the alphabet
  • Successor(2): a → c, b → d, …
  • Predecessor(3): a → x, b → y, c → z, d → a, …
• Example 2. Generic substitution mapping cipher

  plaintext: abcdefghijklmnopqrstuvwxyz

  ciphertext: mnbvcxzasdfghjklpoiuytrewq

  “Easy” to guess the key by observing the ciphertext alone. statistically analyze the language. Some letters are more common in plaintext than others, e.g., e and s are more common than k, j, or z

• Key: mapping from 26 letters to 26 letters
Substitution-based ciphers

• Example 3. Polyalphabetic ciphers. Use N monoalphabetic substitution ciphers with a pattern to cycle between them
• n substitution ciphers, $M_1, M_2, \ldots, M_n$
• Cycling pattern:
  • e.g., n=4: $M_1, M_3, M_4, M_3, M_2$; $M_1, M_3, M_4, M_3, M_2$; ..
• For each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  • Ciphertext for “dog”: substitute d from $M_1$, o from $M_3$, g from $M_4$

🔑 • Key: n substitution ciphers, and the cyclic pattern
Substitution-based ciphers

• Example 4. One-time pad.
  • XOR each bit of the plaintext with one bit of the shared key to generate the ciphertext: ciphertext[i] = message[i] ⊕ key-bits[i]

• Key: a truly random bit string, same size as the message, never reused, held secret, and shared ahead of time
  • Polyalphabetic cipher taken to an extreme: moving randomly through randomly-chosen substitution ciphers

• Statistically very hard to break:
  • All plaintexts are equally likely, since the key is truly random
  • Guessing one part of the plaintext reveals nothing about other parts

• Claude Shannon: a cipher that achieves “perfect secrecy”
Permutation-based ciphers

- Instead of substituting letters in the plaintext, we change their order
- Key: the new order. Convenient to use a word to induce an order

Say the key = ANDREW.
Sorted in alphabetical order, this is ADENRW.
We need to permute each 6-letter part of the message as follows:
1st letter of plaintext → 1st letter of ciphertext
2nd letter of plaintext → 4th letter of ciphertext
3rd letter of plaintext → 2nd letter of ciphertext, etc.

thisisamessageiwouldliketoencryptnow ⇒ tiihssaesmsagioewullkdiетedcnrytopnw

Possible to guess the key by analyzing structure of language and common letters.
Stream and Block Ciphers
Two types of symmetric ciphers

• Stream ciphers
  • Encrypt one bit at time, possibly with some dependence on prior bits

• Block ciphers
  • Break plaintext message in equal-size blocks
  • Encrypt each block as a unit, typically independently
Stream Ciphers

- Combine each bit of keystream with bit of plaintext to get one bit of ciphertext
- $m(i) = i^{\text{th}}$ bit of message, $ks(i) = i^{\text{th}}$ bit of keystream, $c(i) = i^{\text{th}}$ bit of ciphertext
- Encryption: $c(i) = ks(i) \oplus m(i)$ (\(\oplus = \text{XOR}\))
- Decryption: $m(i) = ks(i) \oplus c(i)$
- Very similar to one-time pad, except that the key is generated using a \texttt{pseudorandom} keystream generator

This strategy adopted by the RC4 cipher, deployed in early WiFi security standards (WEP and WPA); later deemed \texttt{insecure}
Block ciphers

• Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).

• Example block substitution cipher: 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>110</td>
</tr>
<tr>
<td>001</td>
<td>111</td>
</tr>
<tr>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>111</td>
<td>001</td>
</tr>
</tbody>
</table>

Ciphertext for 0 1 0 1 1 0 0 0 1 1 1? 101 000 111 001
Block ciphers

• How many possible $k$-bit block substitution ciphers exist?
  • There are $2^k$ values that are permuted amongst themselves: $2^k!$
  • $k=3$-bit inputs: $8! \Rightarrow 40,320$. Not that many.
  • But huge for $k=64$.

• Using a table for substitution is impractical
  • $k=64$: need $2^{64}$-entry table; each entry has 64 bits

• Instead, use a function that simulates a randomly permuted table
• Some heavily used symmetric ciphers are block-based, e.g., AES
Summary of symmetric key ciphers so far

• Assume a pre-shared key between two communicating parties
• Key techniques: substitution and permutation
• Practical ciphers use a complex combination of the two
• Data Encryption Standard (DES)
  • Multiple iterations of substitution and permutation using a 56-bit key
• Advanced Encryption Standard (AES)
  • State of the art for symmetric key encryption. Hardware accelerated
  • A cool animation to understand the steps in AES: https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html
CS 352
Improving Symmetric Key Crypto

CS 352, Lecture 25.4
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana
Review: Symmetric Key Cryptography

- Shared key at both ends, $K_S$
- Algorithms are typically easy to understand and implement
- Achieves confidentiality: harder for Trudy to break ciphertext
- However, fails to provide integrity, authentication, and non-repudiation
- Requires a pre-shared key between Alice and Bob
Attempting authentication with symmetric key crypto
An example: Login system

• Bob runs a login server to provide access to protected resources

• Alice must present a password to login

• Exchange of password implemented using symmetric key cryptography on top of block ciphers
Simple authentication strategy

Alice → “Login: Alice” → Bob

Password please

Kₚₐ(Alice’s password) → Trudy

• Alice’s password is encrypted, and hence protected from Trudy
• Assuming Bob is trusted, Bob can decrypt the password using the shared secret key Kₛ
However, subject to replay attack

\begin{itemize}
  \item Trudy can store the observed ciphertext $K_S(\text{password})$, and \textit{replay it later} to gain access to Bob’s server
\end{itemize}
Preventing replay attacks

• Key idea: Vary the ciphertext for the same plaintext sent at different times.

• Make the ciphertext depend on a one-time value, randomly chosen by Bob.
  • e.g., a random number generated by Bob

• **Nonce**: a “number used once only”

• Alice must combine the password with the nonce before encryption
Challenge-Response with Nonce

"Login: Alice"

"Password please" + Nonce

$K_S$(Alice’s password, Nonce)

Alice

Bob

Trudy

• The nonce changes each authentication attempt
• Trudy cannot reply an earlier ciphertext to produce a valid password
• The nonce is different, so the expected ciphertext is different
• Nonces don’t have to be confidential
Protecting against general replay attacks
Generally, repeated ciphertext is bad

• Real network protocols often have repeated plaintext
  • e.g., the same web page content for the login screen
  • e.g., application headers, like HTTP/1.1 GET
  • The problem is more general: not just about repeating passwords!

• If the same plaintext shows up as the same ciphertext repeatedly, that can be used to break the cipher

• Example: Block substitution ciphers: finding the mapping for one part of one block means other ciphertext can be reversed to guess plaintext of other blocks, and so on…

• Idea: Can we use nonces for all messages?
  • Yes!
However, naïve nonces are inefficient!

• Suppose nonce is used as follows:
  • Alice performs $K_S$ (message $\oplus$ nonce) before transmitting
  • If Alice must send N bits of plaintext, Bob must send N bits of nonce
    • Doubles the number of bits exchanged overall!

• Want to generate nonces automatically & randomly @ Alice, but still have Bob agree on the nonces. How?

• **Cipher block chaining:** use the previous ciphertext as a nonce for the next plain text block

• The first block uses an **Initialization Vector (IV):** only first nonce is sent explicitly by Bob
Cipher block chaining: encryption @ Alice

- IV: Sent by Bob
- C1 depends on the first nonce, IV, not just the plaintext M1
- Ciphertext
- Plaintext
Agreeing on a shared key
How to agree on a shared secret key?

• In reality: two parties may meet in person or communicate “out of band” to exchange shared key

• Often, communicating parties may never meet in person
  • It’s very common not to meet someone you talk to over the Internet
  • Amazon? Your bank?

• And what if the shared secret is stolen?
  • Must exchange keys securely again!

• Q: how to exchange keys securely over an insecure network?

Next lecture: Public key cryptography