CS 161 Midterm 1 Notes — Jeffrey Shen

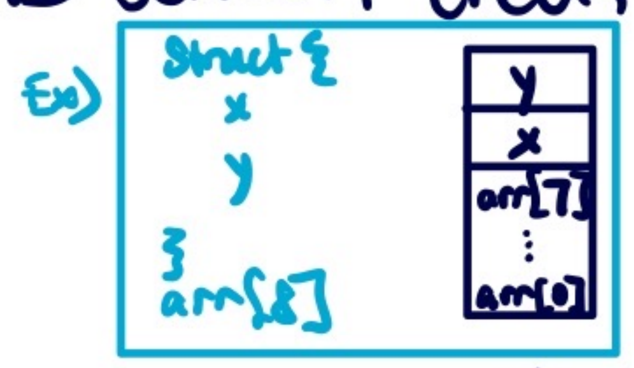
Security Principles

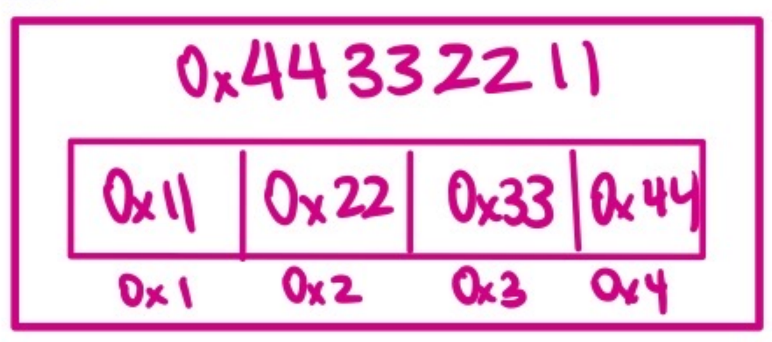
1) Know your threat model: who attacker, what resources
2) Consider Human Factors: make mistakes, user-friendly
3) Security is Economics: expected benefit proportional to attack
4) Detect if you can't prevent: learn attack happened, respond
5) Defense in Depth: layer multiple defenses, force breach all
6) Least Privilege: only as much privilege as needed
7) Separation of Responsibility: split privilege, 2+ to access
8) Ensure complete meditation: check all access, reference monitor
9) Shannon's Maxim: attacker knows system, no security through obscurity
10) Fail-Safe Defaults: choose defaults "fail-safe"
11) Design Security in from Start: Harder to retrofit existing app

Principles for Cryptographic Community

1) Trusted Computing Base (TCB): Part of system that must operate correctly for security goals
   - Unbypassable, Tamper-resistant, Verifiable
2) Time of Check to Time of Use (TOCTTOU): Race conditions between check and use

x86 Assembly and Call Stack





Little Endian: least sig byte at lowest address

Registers: eip: instruction, ebp: base pointer (address at top of stack), esp: stack pointer

Calling Convention:

1) Push args onto stack
2) Push old eip (rip) onto stack
3) Move eip
4) Push old ebp (sfp) on stack
5) Move ebp down
6) Move esp down
7) Execute the Function
8) Move esp up
9) Restore the old ebp (sfp)
10) Restore the old eip (rip)
11) Remove args from the stack

Memory Safety Vulnerabilities

1) Buffer Overflow vulnerabilities: no bounds checks, leading to out-of-bounds memory access
2) Stack Smashing: using long input, overwrite sfp, and overwrite rip
3) Format String vulnerabilites: restrict printf str from user input, can learn/write (%n) contents of mem
4) Integer Conversion vulnerabilites: verify unsigned/signed, C will implicitly cast, can go unnoticed
5) Off-by-one Vulnerabilities: can overflow byte after buffer, redirect sfp and run shell code
6) Other memory safety vulnerabilites: "Use after free", freed but accessible, overwrite C++ vtable ptr

Mitigating Memory-Safety Vulnerabilities

1) Use a Memory Safe-language: many modern languages are memory safe, prevent all
2) Writing Memory-Safe code: defensive programming and safe libraries, tedious
3) Building Secure Software: tools to analyze and patch insecure code, runtime checks
4) Exploit Mitigations: make common exploits harder, cause crashes instead, def in depth

Mitigations

1) Non-executable pages: make some portions of memory non-executable, prevent stack smashing
   - ⟶ counter: return to libc: use libc functions to exploit, pass in args to libc func address
   - ⟶ counter: return oriented programming: chain of return addresses at rip, point to gadget
2) Stack Canaries: dummy value placed on the stack above local var & below saved registers
   - ⟶ counter: guess the canary: 32 bit arch: 24 bits randomness, 64 bit arch, 56 bits, hard
   - ⟶ counter: leak the canary: find vulnerability that allows to read canary (ex. format str vulnerability)

3) Pointer Authentication: use extra bits to store secret Pointer Authentication Code (PAC)
use different PAC for each pointer, f(key, Address), need to find key

4) Address Space Layout Randomization (ASLR): causes absolute addresses of vars to be different (randomized) each time run, shuffle 4 segments mem

↳ counter: guess the address: 32 bit system: 16 bits randomness

↳ counter: leak the address: print values from stack

- Synergistic Protection where one mitigation helps strengthen another

Cryptography

Techniques for securing information & communication in presence of an attacker

Confidentiality: prevent adversaries from reading data, no additional info about M

Integrity: prevents adversaries from tampering w/ our data, attacker cannot change contents w/o being detected

Authenticity: let us determine who created message, message written by person claiming

	Symmetric-Key	Asymmetric key
Confidentiality	Block cipher w/ chaining ex.) AES-CBC	Public key encryption ex) El Gamal, RSA encryption
Integrity & Authentication	MACs ex) AES-CBC-MAC	Digital Signatures ex) RSA signatures

- Prove integrity before authenticity

Kerckhoff's Principle: secure when details except key known, make easy to change key

Threat Models

1) ciphertext-only: intercept single encrypted message
2) known plaintext: know partial information, intercepted encrypted message
3) replay: resend encrypted message, w/o knowing decryption
4) chosen-plaintext: Eve make Alice encrypt arbitrary messages, try to recover
5) chosen-ciphertext: Eve trick Bob into decrypting ciphertexts
6) chosen-plaintext/ciphertext: Eve trick Alice into encrypting some messages of Eve, trick Bob into decrypting

Symmetric-Key Encryption

Assume Alice and Bob share a secret key not known to anyone else

IND-CPA Security: Indistinguishability under chosen plaintext attack

1) Eve choses two messages M_0, M_1 sends both to Alice
2) Alice either encrypts M_0 or M_1, send $Enc(K, M_b)$ back to Eve
3) Eve can ask Alice for encryptions of chosen plaintext
4) Must guess whether encrypted message was M_0 or M_1

XOR Review

$x \oplus 0 = x$
$x \oplus x = 0$
$x \oplus y = y \oplus x$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

- Any deterministic scheme is not IND-CPA secure

One Time Pad (OTP): Alice Bob share n bit secret key

1) Key Generation: Alice & Bob choose shared random key K
2) Encryption algorithm: $C = M \oplus K$
3) Decryption algorithm: $M = C \oplus K$

↳ drawback: key cannot be reused, could $C \oplus C' = M \oplus M'$

Block Ciphers: k key for scrambling

1) Encryption: $Enc_K(M) \rightarrow C$

$E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$, deterministic

2) Decryption: $D_K(E_K(M)) = M$

invertible encryption

Advanced Encryption Standard (AES): block length $n = 128$, key $k = 128$ bits

- Block Ciphers are computationally indistinguishable from random permutation cannot learn anything about M without key

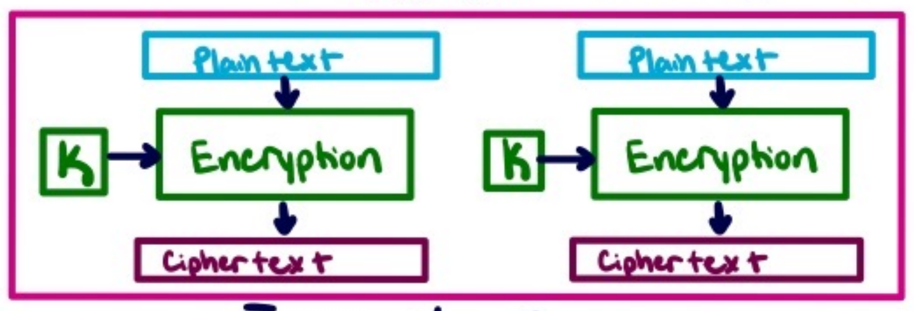
Block Cipher Modes of Operation: need to build algo bc fixed length & deterministic

1) ECB Mode (Electronic Code Book):
break into n-bit blocks $M_1 \dots M_l$

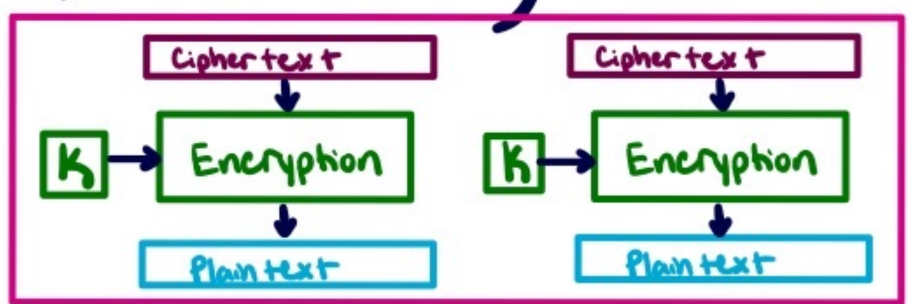
Encryption: $C_i = E_K(M_i)$, concate blocks

Decryption: $M_i = D_K(C_i)$

↳ flaw: leaks info, redundancy shows



Encryption

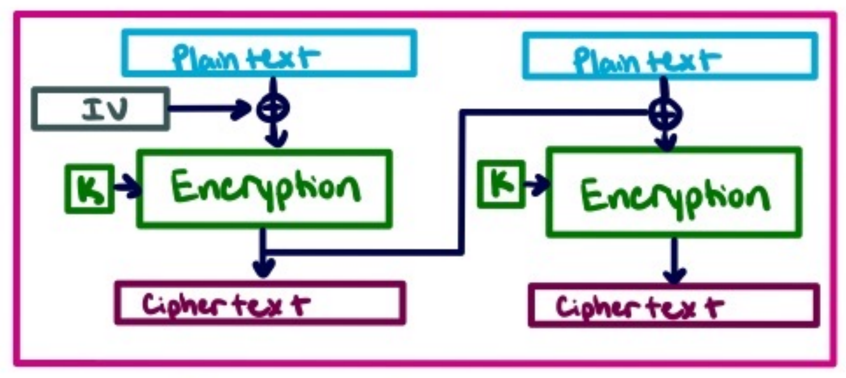


Decryption

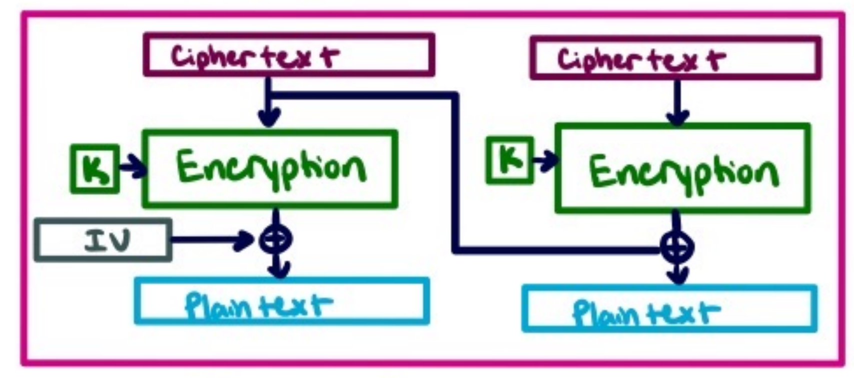
2) CBC Mode (Cipher Block Chaining): need to reveal IV
Choose random initial vector (IV), use prev

Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1} \oplus M_i)$ Not Parallel

Decryption: $P_i = D_K(C_i) \oplus C_{i-1}$ Parallelizable



Encryption

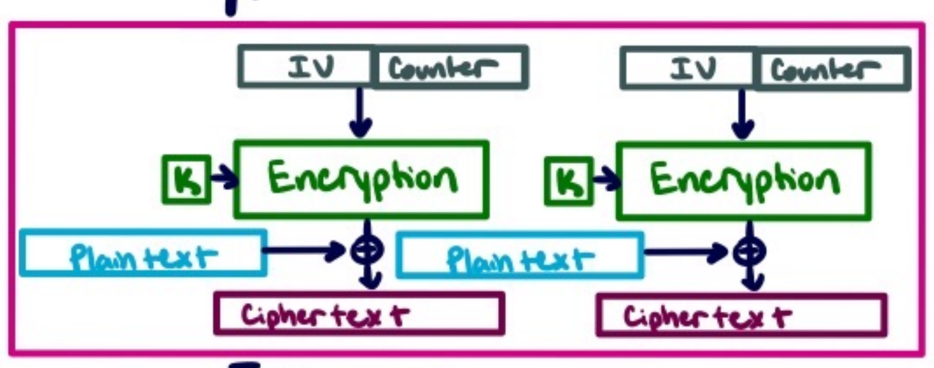


Decryption

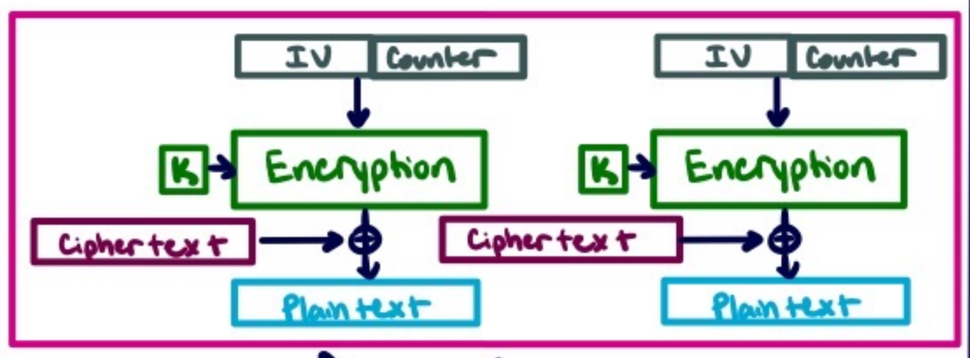
3) CTR Mode (Counter): counter initialized to IV, incremented & encrypted, nonce = IV

Encryption: $C_i = E_K(IV + i) \oplus M_i$ Parallelizable

Decryption: $M_i = E_K(IV + i) \oplus C_i$ Parallelizable



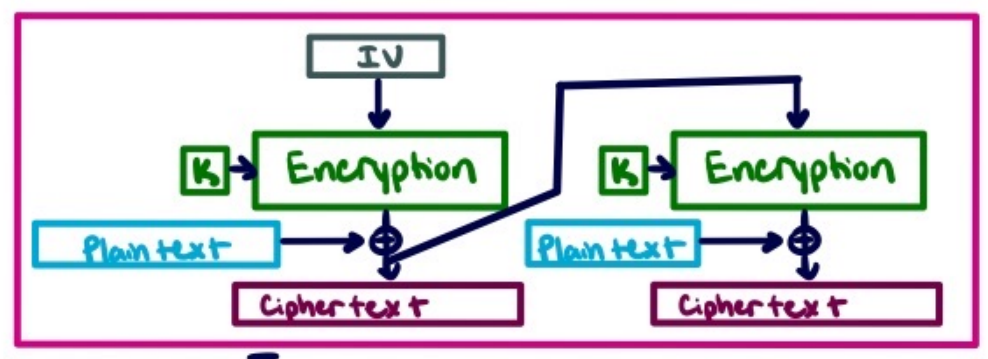
Encryption



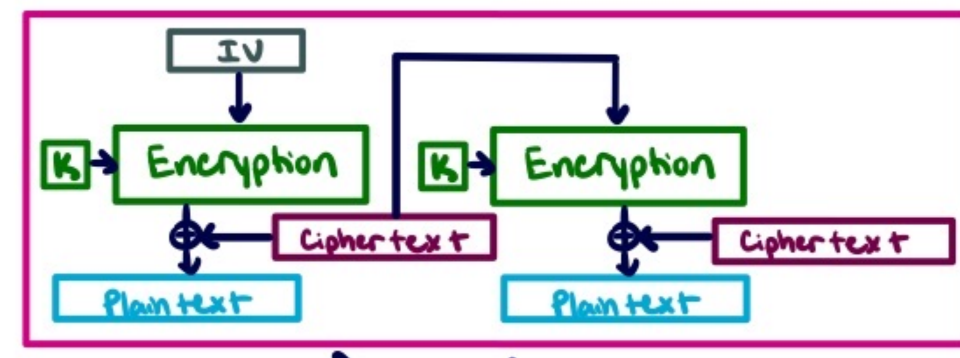
Decryption

4) CFB Mode (Ciphertext Feedback)
Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1}) \oplus P_i$

Decryption: $P_i = E_K(C_{i-1}) \oplus C_i$ no padding needed



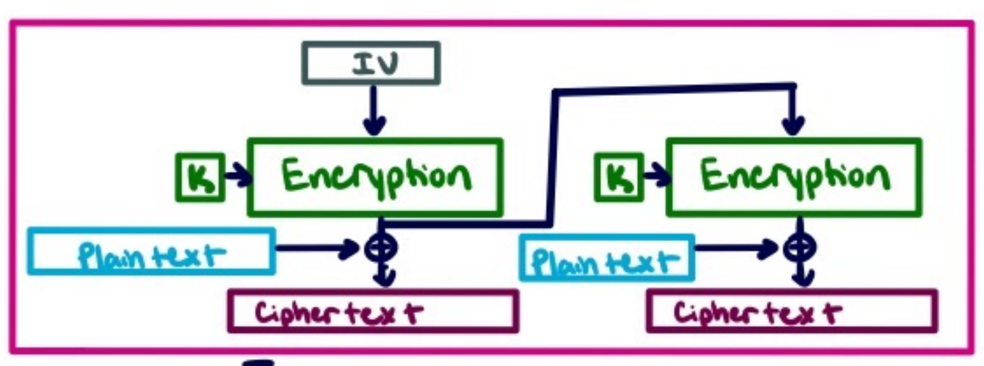
Encryption



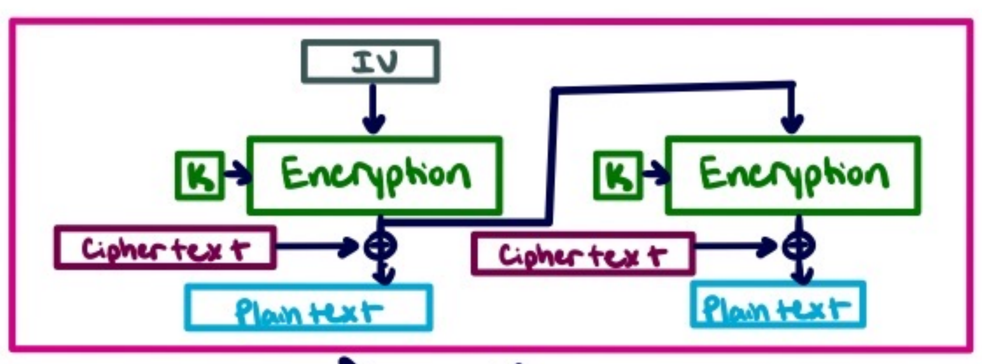
Decryption

5) OFB Mode (Output Feedback)
Encryption: $Z_0 = IV$, $Z_i = E_K(Z_{i-1})$, $C_i = M_i \oplus Z_i$

Decryption: $P_i = C_i \oplus Z_i$



Encryption



Decryption

Padding
Need to make sure message is multiple of block cipher, CTR, CFB doesn't need

PKCS #7: pad message by num of padding bytes

- Don't reuse IV, makes it IND-CPA insecure
- CTR fails catastrophically, CBC is contained

Cryptographic Hashes
used to generate fixed length "fingerprint" of data

Properties: Hash Function $H(M)$ is deterministic, not collision resistant

1) One-way: given output y, infeasible to find any input x $H(x) = y$
2) Second preimage resistant: given x, infeasible to find another $x' \neq x$, $H(x) = H(x')$
3) Collision Resistant: infeasible to find any pair messages x, x' st $H(x) = H(x')$

} implies

Ex) SHA2, SHA3, can be useful for verifing info, request lowest hash

Message Authentication Codes (MACS)
symmetric-key cryptographic primitive, guarantee integrity & authenticity

MAC: $MAC(K, M) \rightarrow T$ keyed checksum of message sent along, deterministic, not confidential

Authenticated Encryption: guarantees confidentiality and integrity

1) Encrypt-then-MAC: $\langle Enc_K(M), MAC_{K_2}(Enc_{K_1}(M)) \rangle$ guarantees ciphertext integrity, prefered
2) MAC-then-Encrypt: $Enc_{K_1}(M \,\|\, MAC_{K_2}(M))$ possibly suseptible to side channel attacks

Pseudorandom Number Generators
generate numbers computationally indistinguishable from rand

Entropy: measure of uncertainty for any random event ↑uniform ↑entropy

Pseudorandom Number Generator: algorithm takes in seed, random bits, generates deterministically

Stream cipher: encrypt and decrypt messages as they arrive one bit at a time ex) AES-CTR

Diffie-Hellman Key Exchange
exchange random value with an evesdropper

Diffie Hellman protocol   public: g, p ↳ attack: man-in-the middle can tamper

Alice	Eve	Bob
generate a		generate b
calc $g^a \bmod p$	g^a → ← g^b	calc $g^b \bmod p$
recieve g^b		recieve g^a
calc $(g^b)^a$	$g^a, g^b \not\Rightarrow g^{ab}$	calc $(g^a)^b$

shared: g^{ab}

Tips
1) Cryptographic hash not 1 to 1 has collisions
2) Stack canary, ASLR, non-executable pages doesn't prevent all buffer overflows
3) Deterministic chosen IVs make block cipher not secure
4) CTR secure w/ predictable nonce, no reuse counter
5) Pad CBC
6) Digital Signature: verifying key public, signing key private

Public-Key Encryption

Recipient has pubic key, sender can encrypt using public key

Trapdoor one-way: one-way but has special backdoor to allow easy to compute

RSA: public key to encrypt, private key to decrypt, deterministic, use padding ex) OAEP

El Gamal: modified Diffie-Hellman to exchange encrypted messages, depends discrete log

Encryption: $E_B(m) = (g^r \bmod p,\ m \times B^r \bmod p)$

g: $1 < g < p-1$
B: $g^b \bmod p$
b: Bob's private key
p: large prime

Decryption: $D_b(R, S) = R^{-b} \times S \bmod p$

↳ drawback: does not preserve integrity

Session Keys: use public key to generate sessions keys which can then be used for symmetric

Digital Signatures

guarantee integrity & authentication, public-key version of MACs

1) Key Generation: $(PK, SK) = KeyGen()$, new pair public, private pair
2) Signing: $S = Sign(SK, M)$ signature on message M
3) Verification: $Verify(PK, M, S)$ true if S is valid signature on M

Ex) RSA
prime p, q $n = pq$
$Sign(M) = H(M)^d \bmod n$
$Verify(M, S) = T$ if $H(M) = S^3 \bmod n$

Certificates

Digital Certificates: certificate signed by trusted authority (private key) validating another's public key

Certificate Authority (CA): party who issues certificates, must be trusted by everyone

Certificate chain: can chain multiple certificates together authenticating person below

Revocation issues: 1) validity periods, 2) Revocation Lists

Leap of Faith Authentication (Trust on first Use): trust public key first time you communicate

Passwords

One of most common security systems, risks associated

↳ Risks:
1) Online guessing
   ↳ rate-limit, CAPTCHAs
2) Phishing
3) Eavesdropping
   ↳ SSL
4) Client-side malware
   ↳ 2 factor
5) Server compromise
   ↳ password hashing: add random salt, slow hash

Bitcoin

digital cryptocurrency, no centralized party, use cryptography

- Each person has digital signature w/ private/public key to send/recieve
- Verify on public ledger, add to ledger using hash chains
- Reach consensus by solving hard problems in proof of work, solved when hash starts w/ N zero bits

CS 161 Final Notes Jeffrey Shen

① SQL Injection

When a web server uses user input as part of code it runs

For request that uses a SQL query, allow SQL to be run from user input

Ex) garbage'; SELECT password FROM passwords WHERE username = 'admin' --

↳ Defense: Escape Inputs: Replace potential input by escaping characters " → \"

Escapers can be hard to build and can be exploited w \"

can add "OR 1=1" to force SQL query to return something

↳ Better Defense: Parameterized SQL: compiles query first, then plugs in user input, compatibility issues but prevents

Intro to Web

URL (Uniform Resource Locator): identifies every resource on web

http://www.example.com/index.html?k1=v1&k2=v2

- protocol: tells browser how to retrieve resource (http/https)
- location: which web server to contact to retrieve
- path: which resource on the server
- Parameters: (optional) sends info to server

HTTP: (Hypertext Transfer Protocol) request-response model client issues request to server, server responds

GET: doesn't change server state POST: sending info to server

Structure:

method → POST /login HTTP/1.1 ← path, protocol

req header:
- Host: squigler.com
- Content-Length: 40
- Content-Type: application/x-url
- Dnt: 1

username=alice & password=123

Elements: HTML (Hypertext Markup Language): structured doc, CSS (Cascading Style Sheets): modify appearance JavaScript: powerful language, Just In Time Compiler

Browser converts HTML to DOM, JavaScript modifies DOM HTTP: port 80, HTTPS: port 443

Same-Origin Policy

Isolate each webpage in browser except same origin

Origin: determined by protocol, domain name and port, string matching on protocol, domain, port

JavaScript: page that loads, Images: page it comes from, frames: URL where frame is from

Cookies & Session Management

Help maintain state in stateless HTTP requests

Cookie: name value pair, Domain, Path: which URL, Secure: HTTPS, HttpOnly: No JS access, expires: when stop storing cookie

- Sends cookie when Domain is suffix of domain, Path is prefix of path
- when setting cookie, cookie's domain is URL suffix of server's URL, cannot set top-level-domain

Session tokens: after login, generate session token and send to user, user uses token in future req

Cross-Site Request Forgery (CSRF)

CSRF: force user to make request, browser will automatically attach session token, server accepts

↳ Defense: CSRF Token: random token include on page, must include CSRF token, and verify valid

↳ Defense: Referer Validation: URL the request was made from, can use to verify but privacy issues

Cross-Site Scripting (XSS)

Inject malicious JS onto webpage, runs JS when webpage loaded

Stored XSS: persistently store malicious JS on server ex) FB post "<script>alert("attack")<script/>

Reflected XSS: vulnerable webpage, server recieves user input and displays user input in response

malicious URL: https://google.com/search?&q=<script>alert("attack")<script/>

↳ Defenses: Sanitize Input, replace characters w) HTML encoding

↳ Defense: Content Security Policy: specifies list of allowed domains scripts can be loaded from

Clickjacking / User Interface (UI) Attacks

Fool victim into clicking on attacker input

Steal a click by deceiving the user, ex) download buttons, fake cursor

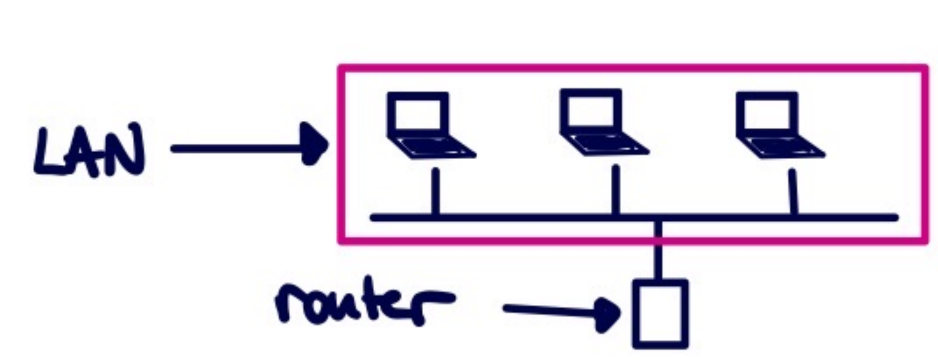
↳ Defenses: Confirmation Pop-ups, UI Randomization, Delay the click, Direct attention to click

CAPTCHAs

test to make sure user is human, not a bot

- typically machine vision problems, but as algorithm improves, defense gets worse
- reCAPTCHAS used to train AI, human or bot spending $0.10?

② Intro to Networking



Local Area Networks (LAN): group of computers all connected

Router: connects two or more LANs, allows wide area network

Internet layering: layers of abstraction on internet

7) Application
6.5) Secure Transport
~~6.5)~~
4) Transport
3) (Inter) Network: between LANs
2) Link: connect machines in LAN
1) Physical: move bits

- Every layer has protocols, agreements for communicating
- messages sent w/ headers containing metadata
- Different layers refer to machine in different ways
  - MAC Addresses: (layer 2) identify machine on LAN ex) ca:fe:f0:0d:be:ef
  - IP Addresses: (layer 3) identify machine globally ex) 128.32.131.10
  - Ports: (higher layers) allow multiple processes for one machine ex) 16-bit, 80
  - Packets: fixed length messages, all communication through packets

Network adversaries: can send packets, faking, spoofing packet headers
1) Off-path adversaries: cannot read or modify any packets sent over connections
2) On-path adversaries: can read, but not modify packets
3) In-path adversaries: can read, modify, and block packets, aka man-in-the-middle

Wired Local Networks: ARP
Address Resolution Protocol (layer 2): translates IP to MAC Address

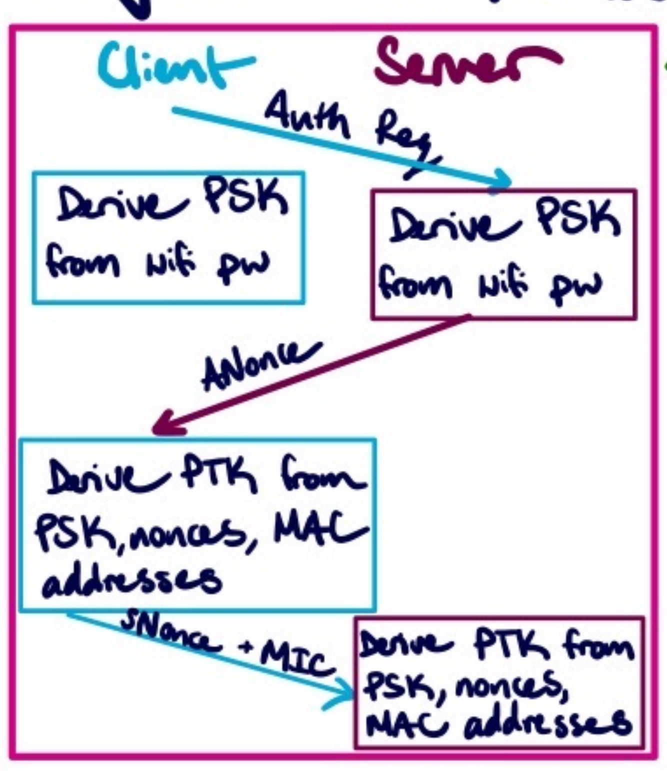
ARP Steps: 1) Alice broadcast to LAN: "What is MAC address of 1.1.1.1?"
2) Bob responds sending message to Alice "My IP is 1.1.1.1, MAC address is ca:fe:f0..."
3) Alice caches IP address to MAC address

↳ Attack: ARP Spoofing: send spoofed reply before Bob can send legitimate reply

↳ Defense: arpwatch can track IP address to MAC address pairings, can use switches instead of hubs w/ MAC caches, Virtual Local Area Networks (VLAN) have isolation

Wireless Local Networks: WPA2
WiFi Protected Access: enables secure communications over WiFi

To join WiFi Network, connects to network AP (Access Point) and announcing SSID (Service Set Identifr)



Pre-Shared Key (PSK): one password for all users, apply PBKDF2-SHA1 on SSID, pw
1) Access Point sends the ANonce
2) Client recieves ANonce & derives PTK (Pairwise Transport Keys), sends SNonce, MIC to AP
3) AP recieves SNonce and can derive PTK, sends GTK (Group Temporal Key) encrypted, MIC
4) Sends ACK for successful GTK

↳ Attacks: on path attacker eavesdrop on handshake if on WiFi, brute force pw if not

↳ Defenses: WPA2-Enterprise: authorized users a unique un, pw, presented w/ random Pairwise Master Key (PMK) instead of PSK, sent over encrypted channels

Dynamic Host Configuration Protocol (DHCP)
layer 2,3: set up configurations when first joining

Need: IP Address, DNS Server IP Address, router IP address
1) Client Discover: client broadcast config request
2) Server Offer: Any server able to offer IP addresses responds w/ configuration settings
3) Client Request: client broadcasts chosen configuration
4) Server Acknowledge: confirms chosen configuration

NAT (Network Address Translation): allows multiple computers on local network to share IP

↳ Attack: attacker sends forged configuration, MITM

↳ Defenses: rely on higher layers to defend, no trusted partner to rely on

Border Gateway Protocol (BGP)
layer 3: send messages globally by connecting local networks

Subnets: groups of IP w/ common prefix
1) If same subnet, ARP to translate IP to MAC address
2) If different subnet, send packet to gateway → Autonomous Systems (AS) identified by Autonomous System Numbers (ASNs), local network managed by organization

BGP: each AS advertises responsible networks to neighboring ASs.

↳ Attack: Malicious ASs can lie and act responsible for a network it isn't

↳ Defense: rely on TCP at higher levels to guarantee sent messages

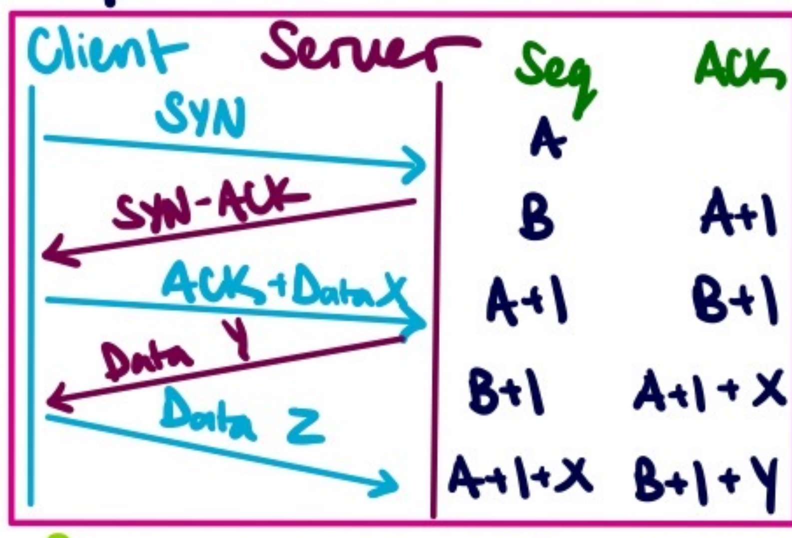
Transport Layer: TCP, UDP
layer 4 connections between individual processes on machines

User Datagram Protocol (UDP): best-effort transport layer protocol, not guaranteed

16	16
source port	dest port
length	checksum

Transmission Control Protocol (TCP): reliable, in-order, connected based stream protocol

- performs handshake to ensure resending dropped packets, rearranging,



1) Client sends SYN, set sequence num field to random ISN
2) If accept, send SYNACK, set SEQ to own ISN, ACK = client ISN+1
3) Client response w/ ACK packet, client ISN+1, ACK = Server ISN+1

16	16
source port	dest port
sequence num	
acknowledge num	
length	checksum

To end, send FIN, reply w/ FIN-ACK, Abort w/ RST, doesn't need ack
SEQ set to index of first byte in packet Client → Server: index in client to server byte stream

Pros: guarantees correctness, in order Cons: slower

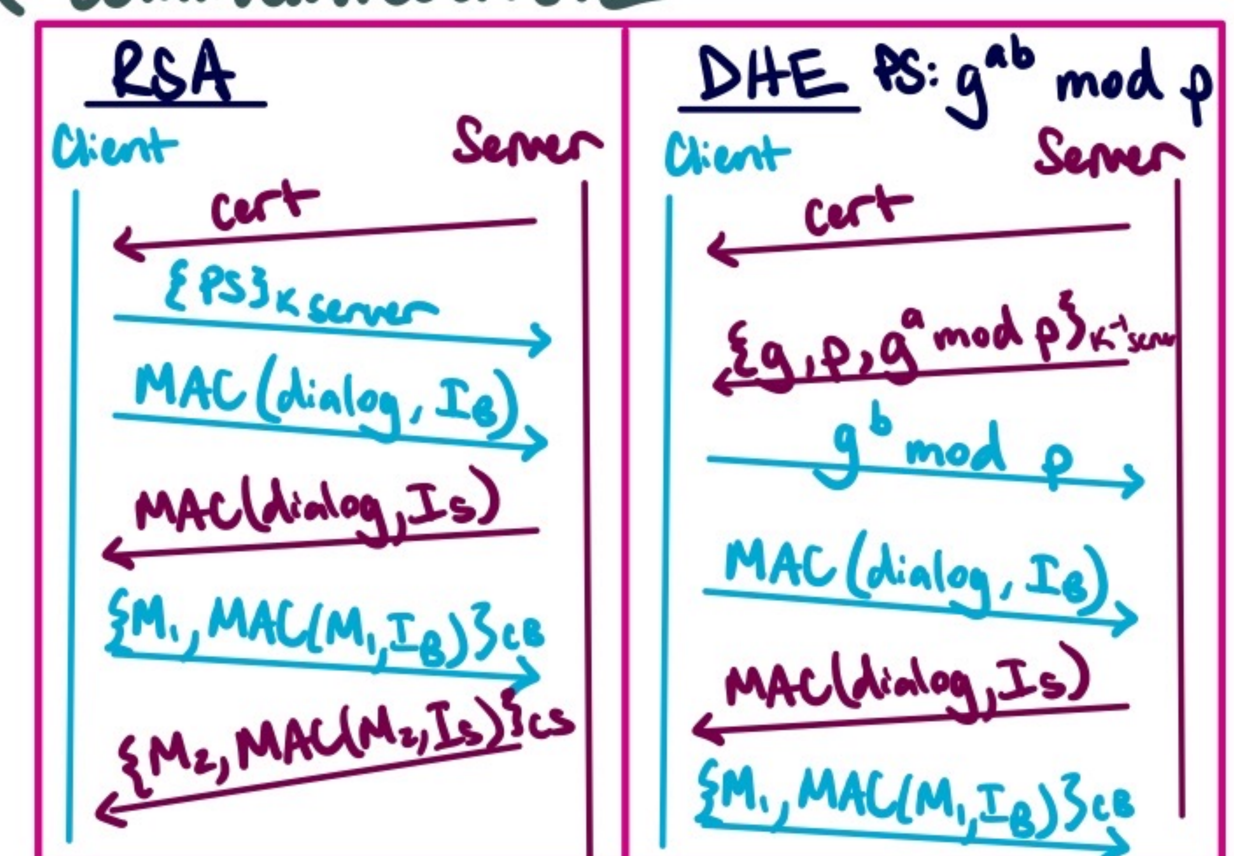
↳ Attack: Packet Injection: spoof malicious packet, fill in header, like packet came from TCP
1) Off-path: know or guess client IP, client port, server IP, server port, sequence num
2) on-path: can inject messages into TCP connection, must race legitimate packet
3) in-path: can block message from party and send their own

↳ Defenses: TCP has no confidentiality or integrity, use TLS, random ISNs

Transport Layer Security (TLS)

provides end-to-end encrypted communication

1) Starts w/ ClientHello random num R_B, supported encryption protocols
2) ServerHello, random num R_S, selected encryption, server's certificate server's pub key signed by certificate authority (CA)
3) Generate Premaster Secret (PS) w/ RSA or DHE
4) use PS to derive 4 shared symmetric keys, encryption C_B, C_S integrity I_B for client, integrity I_S



↳ Replay attacks: recording old messages send to server

↳ Defense: DHE forward secrecy: uncovering new private key has no effect on past security

Security guarantees: client talking to legitimate server, no one tampered w/ handshake, share symmetric key

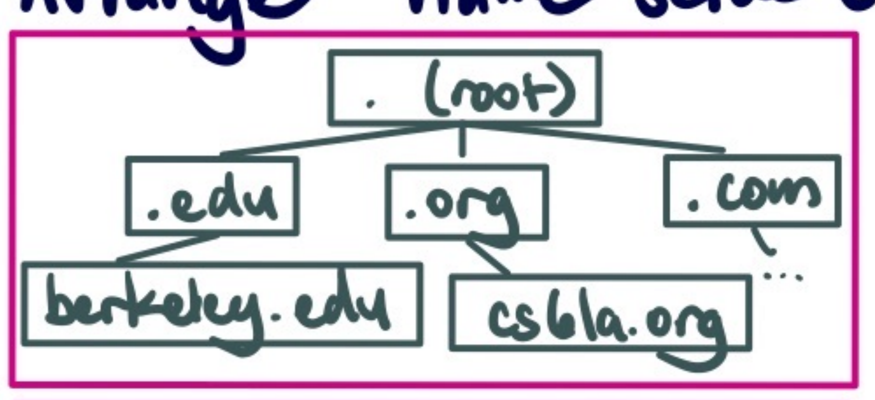
Certificate Authorities: manage proving public key for site

Domain Name System (DNS)

Protocol translates between human readable and IP addresses

Name Servers: servers dedicated to replying to DNS requests, DNS collection of name servers

Arrange name servers in a tree hierarchy based on zone, queries start at the root, direct to child



DNS Recursive Resolver: DNS lookup from ISP, sends queries, internal cache
DNS Stub Resolver: on computer sending req to recursive resolver
Uses UDP for requests

Identification	Flags
#Questions	#Answers
#Authority RR	#Additional RR
Questions	
Answers	
Authority	
Additional Info	

Identification: randomly selected, to match req to responses
Flags: query or response, successful
Record types 1) A type records: domains → IP, 2) NS type: zones → domain
Resource Records (RRs): 1) Question section: domain queried for 2) Answer section: direct answer to query 3) Authority section: zone & domain of next name server 4) Additional section: IP address of next server

↳ Attack: malicious name server, send addresses to malicious IP
↳ Defense: Bailiwick Checking: name server only provides records in its zone
↳ Attack: on path attackers can read, respond w/ malicious records
↳ Attack: offpath: Kaminsky attack: create response to a query for a nonexistent website, have website continously spam nonexistent site until malicious response answers first which can redirect to malicious IP since nonexistents are not cached
↳ Defense: UDP source port randomization: randomize 16 bit source port, offpath must guess ID & source port $1/2^{32}$

DNSSEC

provide integrity & authentication on DNS messages

Trust Anchor: delegate trust to others use root servers
- have signature on next name server's public key
- signing done offline to save time

Trust Anchor

root KSK
1. [DNSKEY] Public KSK
2. [DNSKEY] Public ZSK
3. [RRSIG] Signature on (2)

root ZSK
4. [DS] Hash of (6. child's public KSK)
5. [RRSIG] Signature on (4)

.edu KSK
6. [DNSKEY] Public KSK
7. [DNSKEY] Public ZSK
8. [RRSIG] Signature on (7)

.edu ZSK
9. [DS] Hash of (11. child's pub KSK)
10. [RRSIG] Signature on (9)

- DNSSEC name server generates two public/private key pairs, Key Signing Key (KSK) for signing zone signing key (ZSK) and ZSK for everything else.
- for nonexistent, displays hashes of interval of domains

Denial-of-Service (DoS) cause software to be unavailable typically overloading bottlenecks

Application level DoS can involve exhausting filespace, RAM, threads

↳ Defenses: authenticate users, isolation, user quota on resources

SYN Flood Attacks: send large # of SYN, w/ no ACK, wastes memory

↳ Defenses: overprovisioning: make sure server has alot of memory, filter packets

Distributed Denial of Service (DDoS): power of many machines (botnet) to overwhelm

↳ Defenses: analzye traffic coming in and drop packets

Firewalls block access to network services from running on internal machines

Security Policies: 1) Default-allow/blacklist: permitted unless listed as denied, fails-open: mistake is security flaw

2) Default-deny/whitelist: denied unless listed allowed, fail-closed: mistake is loss of functionality

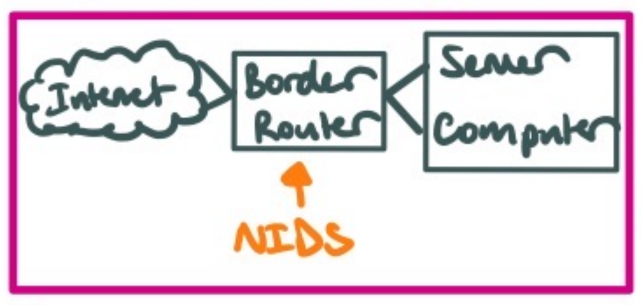
Stateful packet filter: checks packet against control policy, remembers open connections

Application layer firewalls: restrict traffic on data content

Pro: central control at chokepoint, easy deploy, solve security Cons: loss of functionality, malicious insider

Intrusion Detection Detect when attacks happen

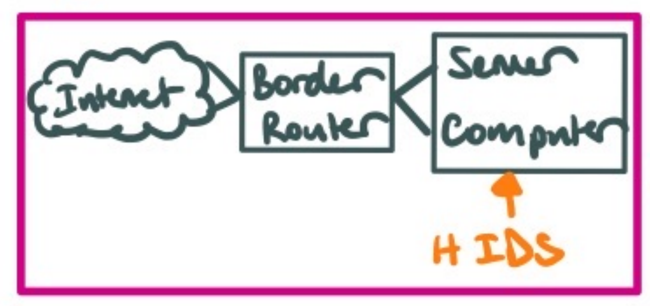
1) Network Intrusion Detection System (NIDS)



Pro: cheap, easy to scale, simple, small compute

Con: inconsistent, evasion attacks, unable to detect encrypted traffic

2) Host-Based Intrusion Detection System (HIDS)



Pro: less inconsistencies, works w/ encrypted, protect non network, performance scales

Con: expensive, still vulnerable to evasion attacks

3) Logging: analyze log of activity

Pro: access to end host data, cheap

Con: not real-time, possible evasion attacks

False Negatives: attack happened, no attack reported

False Positive: no attack, attack reported

1) Signature-Based Detection

matches known attack structure, blacklisting

Pro: good at know signatures,

Con: won't catch new attacks, variants

2) Anomaly-Based Detection

model of normal activity, flag any deviations

Pro: catch new attacks

Con: difficult to train model, can have FN, FP

3) Specification-based detection

specify normal activity, flag deviation (whitelist)

Pro: catch new attacks, FP can be very low

Con: time consuming to write specifications

4) Behavioral Detection

look for evidence of compromise

Pro: catch new attacks, can have low FP, cheap

Con: detected after started, use different behavior to execute

Malware attacker code that runs on victim computers

Virus: requires user action to propogate ex) infect startup code of application

↳ Detection: Signature based → Polymorphic & Metamorphic Code: encrypted code of malcode or code rewriter

Worm: does not require user action, alters code already running, spreads exponentially

Anonymity & Tor anonymous communication

proxy: intermediary to relay traffic

onion routing: use multiple proxy servers so at least one can be trusted, if not colluding, cannot connect

↳ Con: performance decreases