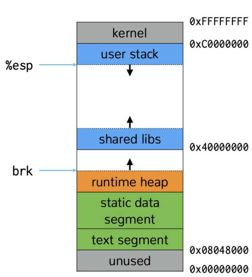
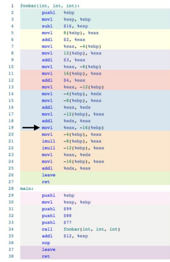
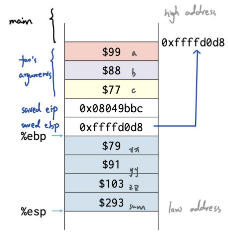
Stack Overflow







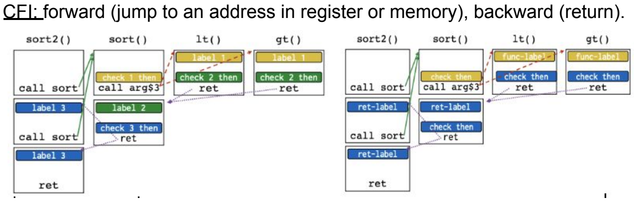
Avoid unsafe funct	Avoid strcpy, strcat, gets, etc	Good idea in general	Requires manual code rewrite Non-lib funct vulnerable No guarantee everything found Alternatives also error
Stack canary	Place canary between local var and fpointer. Check bef jmp.	No code changes Only recompile	Performance penalty per return Only protects against stack smashing Fails if can read memory. Modify prelogue.
ASLR	Randomize the address of different memory origins	No code changes or recompile	32-bit arch get limited protection Fails if can read memory Load-time overhead
W^X (Data Extraction Prevention)	Use memory page permission bits.	No code changes or recompile	Requires hardware support Defeat by ROP; Not protect JIT code;
CFI	Restrict indirect transfers of control. Direct transfer is hardcoded.	No code changes or Hardware support Protect many vulns	Performance overhead; data-only attacks Requires smarter compiler Need all code available (see)

Stack Canary:

-fstack-protector	Functions with character buffer ≥ ssp-buffer-size (8) Functions with variable sized alloca()
-fstack-protector-strong	Function with local arrays of any size Functions that have references to local stack variables
-fstack-protector-all	All functions

Separate Control Stack: control stack, safe stack (return, register, local vars), shadow stack (%ssp) then compares %esp.

CFI: forward (jump to an address in register or memory), backward (return).



Defeats: (above: left fine-grained, right coarse-grained)

ROP: x86 instruction variable length, start on any byte boundary. 0xc3 means ret.

UAF: overwrite the vtable of freed object so entry points to attacker's code

Stack canary	1. pointer subterfuge (Fix: buffer closer to canaries; args copied to top of stack, pointers loaded into register before strcpy()); 2. Use memcpy other than strcpy null; 3. Chained exploited / Brute Force servers to learn the canary (fork process and guess).
Separate stack	Find a function ptr and overwrite it to point to shellcode; Put buffers, &var, and function pointers on the **user stack** such that overwrite function pointers when c programs compile to WebAssembly
W^X	Jump to existing code; Inject code with JIT, JIT spraying on heap overflow pointer.
ASLR	1. Find base through guess to usleep() [base + offset], max try: 2^16 = 65,536 tries. sleep, succeed; crash, next guess; 2. Call system() with &buf "/bin/sh".
CFI	Jmp to funct that has the same label, then return to more sites

IntOverflow	Truncation (assign 64 to 32); arithmetic overflow (0xffffffff + 2); Signedness bugs (0xffffffff = -1 > some num)

Isolation & SideChannel:

Separate into isolated least privileged compartments. Units: physical → **virtual machine** → **OS processes** → library → function (coarse → fine grained).

Real user ID (RUID)	Same as the user ID of parent Used to determine which user started the proc
Effective user ID (EUID)	Setuid bit on the file being executed, or syscall Determines the permissions of the process
Saved user ID (SUID)	Used to save and restore EUID

SetUID: root = 0; setUID: EUID, setEGID: GID of file, sticky bit: on (only owner and root can rename or remove), off (then only if user has *w* permission)

Examples: Android apps has its own process ID, co limited using UNIX domain sockets + ref monitor checks permission; OKWS each server runs with unique UID, communication limited to structured RPC; modern browsers process; Qubes OS, trusted domain.

Memory Isolation: ACL, namespace (partition kernel), syscall filtering (scccomp)

Translation: MMU translates VA to PA, Page size = 2^{12} → multilevel page table. Each process/kernel has its own tree, context switch changes root.

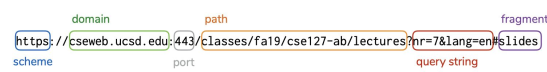
Translation LookAside Buffer: cache for address and access control. PCID for process context in the cached buffer. Extended nested page table entries (VPID for VMs).

ACL: page descriptors contain access control information (R, W, X).

Cache Side Channel: evict & time; prime & probe (time, slower means evicted); flush & reload (flush the cache, faster means evicted)

Virus	Code propagates by arranging itself to eventually be executed. Altering source code.
Worm	Self propagates by arranging itself to immediately be executed. Altering running code.
Rootkit	Program designed to give access to an attacker while actively hiding its presence.

Web Security Model



HTTP/2: allows pipelining requests for multi objects; multiplexing multiple requests over one TCP connection; header compression; server push

Cookies: a small piece of data server sends to browser, browser updates and sends it back with subsequent requests.

SOP: origin: isolation unit/trust boundary *(scheme, domain, port)*. Isolate content of different origins;

SOP for DOM: each frame has its own origin; can only access data with the same origin; communication using postmessage API

SOP for HTTP responses: prevents code from directly inspecting HTTP responses; documents: can load cross origin but not inspect or modify frame content; scripts: can load cross origin, exe with same privilege of the page; images, fonts, css: can render cross origin but not inspecting each pixel

SOP for cookies: origin *(scheme, domain, path)* browser makes cookie available to given domain + sub-domains; path + child-path

Cookie 1:
name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:
name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:
name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

	Cookie 1	Cookie 2	Cookie 3
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/my	No	Yes	No

CSRF: 1. use attacker's domain to interact with banks url with user's cookie; submit transfer form from attacker's site with user's cookie. 2. Cookies can also be inspected through HTTP communication. → state-changing requests (authenticate POST).

Attacks: drive-by pharm hack routers; native apps run local servers. Login CSRF.

CSRF Defense: 1. Secret Token Validation (session-dependent identifier or token so attacker cannot retrieve due to SOP - attacker's site has different origin); 2. Referer/Origin validation: includes url of the previous web page; 3. Samesite cookies: strict: never send cookie in any cross site browsing context; Lax: allowed when following a navigation link but blocks it in CSRF-prone request method; None: send cookies from any context; Secure: encrypted requests only; HttpOnly: cookie not in the document

Injection: Command injection**:** execute command on system bypassing unsafe data into shell (./head10 "myfile.txt; rm -rf /home");

Code injection: eval function

SQL injection**:** take user input and add it to SQL string; Defense: never build SQL commands by user.. Use parameterized (AKA ***prepared***, faster because of cached) SQL; ORM (object relational mappers) (provide interface between obj & DBs)

XXS: attacker inject scripting code into pages generated by a web application.

Reflected: the attacker script is reflected back to the user as part of a page from the victim site. E.g. paypal

Stored: the attacker stores the malicious code in a resource managed by the web application, such as a database. E.g. Samy, CSS for JS

DOM XSS: only allow sanitized TrustedHTML type values to end up in document.

CSP: whitelist valid domains of sources, scripts etc, as HTTP header or meta HTML object. Or whitelist trusted origins that iframe can talk to, e.g. password PW checker only gives PW to trusted origin. E.g. Content-Security-Policy: default-src 'self'; img-src *; media-src: media.com.

frame-ancestors	Specify valid parents that may embed this page E.g. 'none' = X-Frame-Options: deny
upgrade-insecure-requests	Rewrite HTTP url to HTTPS
block-all-mixed-content	Don't load any content over HTTP

IFrame Sandbox: whitelist privileges. → privilege separate pages into multiple frames

allow-scripts: allows JS + triggers (autofocus, autoplay, etc.)

allow-forms: allow form submission

allow-pointer-lock: allow fine-grained mouse moves

allow-popups: allow iframe to create popups

allow-top-navigation: allow breaking out of frame

allow-same-origin: retain original origin

HTTP Strict Transport Security (HSTS): never visit site over HTTP again; strict-transport-security: max-age=n (1-year = 31536000 seconds);
Subresource Integrity (SRI): CSP + HSTS can be used to limit damages but cannot really defend against malicious code; Idea: page author specifies hash of (sub)resource they are loading; browser checks integrity; When check fails: 1. Browser reports violation and does not render or execute resources; 2. CSP directive with integrity-policy directive set to report (report but may render or execute)
Cross-Origin Resource Sharing (CORS): Browser send origin header with XHR request; Server can inspect origin header and respond with access-control-allow-origin header; CORS XHR may send cookies + custom headers. DON'T use insecure JSONP.
Extension: different heap from main page; privilege separate between core extension script and content script; explicitly let user decide privileges
All of above are DAC (Discretionary access control) identity based
COWL is about corresponding security with data sensitivity. If a bc.u iframe now received sensitive data, it can't talk to bc.u anymore.

<u>Others</u>

Parameterized SQL allows you to pass in query separately from arguments

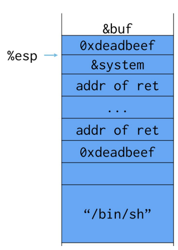
```
sql = "SELECT * FROM users WHERE email = ?"
cursor.execute(sql, ['nadiah@cs.ucsd.edu'])

sql = "INSERT INTO users(name, email) VALUES(?,?)"
cursor.execute(sql, ['Deian Stefan', 'deian@cs.ucsd.edu'])
```

Values are sent to server separately from command. Library doesn't need to try to escape

Benefit: Server will automatically handle escaping data

Extra Benefit: parameterized queries are typically *faster* because server can cache the query plan

Break ASLR, Stack Overflow to exe libc system "/bin/sh" Explained:



Browser Execution
Windows: load content, parse HTML, JS, fetch resources, respond to events.
Nested execution: frame as rigid visible division; iframe: floating inline frame. Delegate screen content from another soue.

SOP MORE

DOM	Each frame in a window has its own origin Frame can only access data with the same origin - DOM tree, local storage, cookies, etc.
HTTP	Pages can perform requests across origins: - Page can leak data to another origin by encoding it in the URL, request body, etc. SOP prevents code from directly inspecting HTTP response. - Except for documents, can often learn some information about the response.
Document	Can load cross-origin HTML in frames, but not inspect or modify the frame content.
Scripts	Can load scripts from across origins, but scripts execute with **privilege** of the page.
Images	Can render cross-origin images, but SOP prevents page from inspecting individual pixels.
Cookie	([scheme], domain, path). A page can set a cookie for its own domain or any **parent** domain (if the parent domain is not a public suffix). Browser will make a cookie available to the given domain, including any sub-domains.

More CSRF

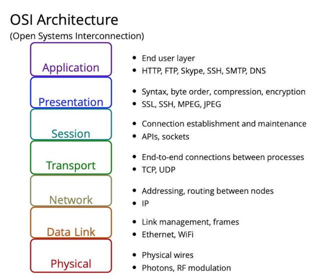
Defense:
- Header: SameSite = Strict;
    - A same-site cookie is only sent when the request originates from the **same site**.
- Header: Secure
    - A secure cookie is only sent to the server with an **encrypted** request over HTTPS protocol.
- Header: HttpOnly
    - The cookie is not in document.cookie

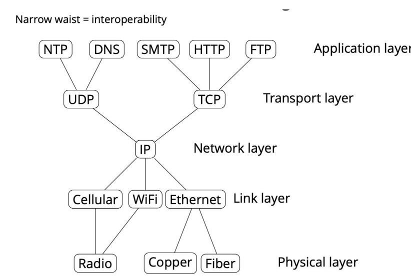
DOM SOP vs. Cookie SOP
- Cookies: cseweb.ucsd.edu/AAA can't see cookie for cseweb.ucsd.edu/BBB
- DOM: cseweb.ucsd.edu/AAA can access DOM of cseweb.ucsd.edu/BBB.
    - To access cookie:

```
const iframe = document.createElement("iframe");
iframe.src = "https://cseweb.ucsd.edu/~nadiah";
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```

- E.g. If a bank includes Google Analytics JavaScripts, it can accesss your bank's authentication cookie.

<u>Network</u>





Ethernet: every node has globally unique 6 byte **MAC** (Media Access Control), switched(no to all, to only one correct port),,message in the frame
ARP (Address Resolution Protocol)
Problem: How does a host learn what MAC addresses to send packets to? • ARP lets hosts build table mapping IP addresses to MAC addresses. • ARP request: source MAC, dest MAC, "Who has IP address N?" • ARP reply: source MAC, dest MAC, "IP address N is at MAC address M."
IP: Internet Protocol
Connectionless delivery model • "Best effort" = no guarantees about delivery, Packets might be lost, delivered out of order, delivered multiple times, fragmented
Routing: BGP (Border Gateway Protocol)
• Routers maintain global table of routes
TCP(Transmission Control Protocol)
Want abstraction of a stream of bytes delivered reliably and in-order between applications on different hosts.
Ports Each application is identified by a port number, 16bits, 1-65536
TCP Sequence Numbers: Bytes in application data stream numbered with 32-bit sequence number • Data sent in segments: sequences of contiguous bytes sent in a single IP datagram
TCP Sequence numbers and Acknowledgement: Two logical data streams in a TCP connection: one in each direction; Receiver acknowledges received data: acknowledgement number(ACK) is sequence number of next expected byte of stream in opposite direction
FIN: Closing TCP connections, FIN initiates a clean close of TCP connection, waits for ACK from receiver. • If a host receives a TCP packet with RST flag, it tears down the connection; Designed to handle spurious TCP packets from previous connection
UDP: User Datagram Protocol, offers no service quality guarantee, transport layer protocol that is a wrapper around IP, adds ports to let applications demultiplex traffic, useful for applications that only need best-effort guarantee
DNS: handle mapping between host names and IP addresses, a delegatable, hierarchical name space;13 root servers; cached for quicker response, queried progressively according to domain name hierarchy;
Packet Injection: ARP spoofing -->• ARP requests broadcast to local subnetwork-->Attacker on local network can impersonate any other host. (link level threats)
Jamming: Violates availability. physical/link layer threats,
Spoofing: Set arbitrary source address(IP packets offer no authentication. • Source address in IP set by sender. • In principle, can spoof packet from any host from anywhere on the internet)

Misdirection: BGP hijacking→ BGP protocol manages IP routing information between networks on the internet→ Routes are not authenticated: malicious or malfunctioning nodes may provide incorrect routing information that redirects IP traffic (network layer threats)
TCP threats: On-path injection, Connection hijacking": If an on-path attacker knows ports and sequence numbers, can inject data into the TCP connection. RST injection: can STOP connection, blid TCP spoofing: guess ACK-SYN response by knowing SYN
DNS threat: Malicious DNS server: Any DNS server in query chain can lie about responses. • Local/on-path attacker: Can impersonate DNS server and send a fake response. • Off-path attacker: Can try to forge response: needs to match 16-bit query ID. (Application layer threats)
Eavesdropping:tapping(ethernet), tcpdump
Network Perimeter Defense:
Firewalls:protecting or isolating one part of the network from other parts; Personal firewalls,Network firewalls, Filter-based(on packet headers),Proxy-based
A firewall enforces an **access control policy**(restrict external users, inbound); default allow: permit all services, shut off for specific problems; default deny: permit only a few well-known services, add more as users complain.
Packet filtering firewalls check every packet against rules and forward or drop based on the header of packet, such as source/Destination IP, port
Circumventing simple firewall rules:1. Send traffic on port that is for another service. 2 Tunneling. Encapsulate one protocol inside another
Network Address Translation (NAT): NATs map between two different address spaces. IP addresses need not to be globally unique.
Pros: only allow connection to outside from inside, no need for large external address space
Cons: rewrite IP addresses is not easy; breaks some protocols
Application Proxies: Control applications by requiring them to pass through proxy. The man in the middle; enforce policy for specific protocols(SSH,SMTP);careful on enterprise network, root certificates on employee
Network Intrusion Detection Systems (NIDS): passively monitor network traffic for signs of attack; Network based detection: look at network traffic, scanning HTTP requests
Pros: don't need to modify or trust end systems; Cover many systems with single monitor; Centralized management
Cons: expensive, vulnerable to evasion attacks(incomplete analysis,imperfect observation,things come in seperate packs)
host -based detection:instrument web server, scan arguments sent to back-end programs
Pros: the semantic gap is smaller; have understanding of urls; Don't need to intercept HTTPS
Cons: expensive; still have to consider unix filename semantics, other sensitive files, databases; Only help with web server attacks
Log analysis: run scripts to analyze system log files (e.g. fail2ban)
Pros: cheap, servers already have logging facilities; No escaping issues(logging done by serv)
Cons: reactive; detection delayed; can't block attacks; worry about unix filename semantics; Malware may be able to modify logs
Vulnerability scanning: rather than detect attacks, launch them yourself
Pros: accurate, it finds real problems; proactive, can prevent future misuse; intelligence: can ignore IDS alarms you know can't succeed
Cons: can take a lot of work; not helpful for systems you can't modify; dangerous for disruptive attacks
Honey pots: deploy a sacrificial system that has no operational purpose
Designed to lure attackers, any access is by definition not authorized, and is either an intruder or a mistake; provides opportunity to identify intruders, study what they're up to, divert them from legitimate targets

Timing Side Channels
Instruction: Floating point time variability → avoid variable-time instructions → use known, safe CT operations **Control Flow**: conditional statement → don't branch on secrets, fold control flow into data flow: select. **Memory access**: don't access memory based on a secret → loop over public bounds of array. **Methods**: design new programming language FaCT, program analysis, auto transform to CT code

Privacy and Anonymity
PGP: sign+encrypt, key management and web of trust; Yet, outdated cipher choices, doesn't authenticate encryption with a MAC
Modern: Diffie-Hellman to negotiate ephemeral keys, long-term authentication keys with out-of-band fingerprint verification, offer forward secrecy (no past info) + deniability.
TOR: Use: tor entry + tor relay + tor exit and internet sites. Service: 1. Bob pick introduction points IP and build circuits to them; 2. Advertise the service at database (IP, PK); 3. Alice sets up rendezvous point RP; 4. Alice writes a PK-enc message to bob listing the point and a one-time secret, for an IP to deliver it to bob; 5. Bob connects to Alice's RP and provides her one-time secret. 6. Use this Tor circuits.
3-layer tunnel ensures no-one in the connection knows both your IP address & yr request.
Track: Tracking contents, include tracking code in URLs
fingerprinting: profile your browsers,extensions,OS,hardware,etc; use privacy caring browser, privacy-enhancing xtnsn.anonymity achieved by proxy→ rewrite sender to anonymous, VPN services allow user to use tunnel traffic

Network Connections
When you type "ucsd.edu": DHCP (connect to local router), ARP (get MAC of local router), DNS (lookup on ucsd.edu), TCP (connect ack), GE

DHCP (Dynamic Host Configuration Protocol) to bootstrap the laptop on local network
a. Broadcasts DHCPDISCOVER to 255.255.255.255 with MAC
b. New host without IP address → DHCP server responds with config: lease on host IP, gateway router information, DNS server information

ARP requests to learn the MAC of the router
a. Your **laptop** encapsulates each IP packet in a WiFi Ethernet frame addressed to the local router
b. **Local router** decapsulated these frames and re-encoded them to forward on its fiber connection to **upstream ISP** or another part of the network
c. Each hop re-encodes the link layer for its own network.
d. Outside connection will be encapsulated in a link-layer frame designated at local router's MAC address.

DNS lookup on ucsd.edu
a. Learned IP from local DNS (from DHCP) or hard-coded server
   i. DNS query encapsulated in one or more UDP packets in one or more IP packets
   ii. Each response tells what authority to query, until it learns the final IP address.
b. Address is cached.

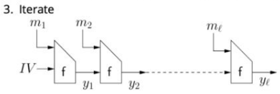
TCP connection to IP address 132.239.180.101
a. Each TCP triple handshake packet is encoded in an IP packet, encoded as Ethernet frames, that are decoded and re-encoded as they pass through the network.
b. Local router's routing table's IP prefixes: match against the IP address that tells it what address to forward the packets to.
c. The packet passes through a series of ASes: sbcgobal.net → att.net → level3.net → cenic.net → ucsd.edu

Laptop sends HTTP GET request inside TCP
Based on HTtP response, the laptop performs a new DNS lookup, TCP handshake, and HTTP GET requests for every resource in the HTML as it renders.

Symmetric-Key Encryption
Encryption (key, plain) → cipher; decryption (key, cipher) → plain; Inserve operation: $D_k(E_k(m)) = m$, one-time/multi-use keys
Stream cipher: pseudo random key: $E_k(m)$ = **PRG(k)** @ m; computationally hard to distinguish from random; danger: key cannot be used once.
Block cipher: permutation of fixed-sized inputs, each input mapped to exactly one output. 1. ECB: reveals original information; 2. CBC: subtle attacks abuse padding; 3. CTR: counter use block cipher as stream cipher.
AES: |m| = |c| = 128 bits, |k| = 128, 192, 256 (USE THIS, better than others)
Secrecy: all can be broken using brute force, complexity proportional to key space.
Hash Function: maps arbitrary length into a fixed-size string; bit security 128 bit but only 64 bit security; functions: MD5, SHA1 (160 bits), SHA2 (224, 256, 3844, 512 bits)
MAC: Validate message integrity based on shared secret: a = $MAC_k(m)$. MAC then Encrypt (SSL) / Encrypt and MAC (SSH): integrity to plaintext, decrypt then verify; Encrypt then MAC (IPSec): integrity for both plaintext and ciphertext.
HMAC: $MAC_k(m)$ = H(k@opad || H(k@ipad || m)), against extension hacks.
AHEAD: authenticated encryption with associated data (AES-GSM, AES-GCM-SIV)

Public-Key Cryptography
Message Authentication Code: should be unforgeable by an adversary. MAC(c) = GoodHash(c) not good because adversaries can compute H(m) for any m.
Length Extension Attack:

Merkle-Damgard Construction: construct a hash function that takes arbitrary length inputs from a fixed-length compression function
MD5:
- Mechanism
  1. Input $m = m_1||m_2||\ldots||m_\ell$ where m_i are 512-bit blocks.
  2. Append $1||000\ldots000||len(m)$ to the last block, where as many bits as necessary to make m_ℓ a multiple of 512.
  3. Iterate

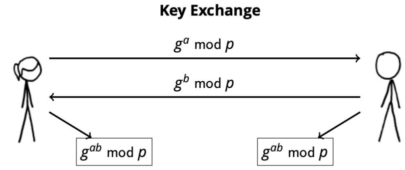


- Hack:
  - Observes $BadMAC_k(m)$ = H(k || m) for unknown k and m
  - Aims to forge $BadMAC_k(m || r)$ for arbitrary r
  - Guess the length of k || m, then reconstruct the padding and append additional blocks.
    - $BadMAC_k(m || padding || r)$

Conclusion: $MAC_k(m)$ = H(k || m) not a secure MAC, if H is MD5, SHA1, SHA2.
HMAC is a good choice for double computation of the key, can't forge without knowing the key.

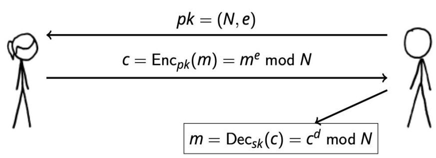
Public-key Encryption: $Enc_{pk}(m) = c$, (public key, plaintext) → ciphertext; $Dec_{sk}(c) = m$, (secret key, ciphertext) → plaintext; $Dec_{sk}(Enc_{pk}(m)) = m$.
Textbook Diffie-Hellman Key Exchange: passive attack: compute discrete log of public values, parameters *p* should be >= 2048 bits → elliptic curve Diffie-Hellman;

Key Exchange



Note: $(g^a)^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p(g^b)^a \bmod p$.

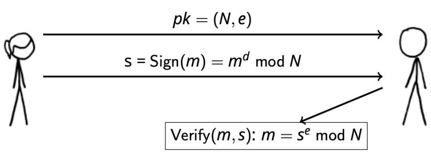
Computational complexity: no general-purpose factoring, efficient for small factors for random integers; modular exponentiation and inverse are efficient to compute.
Textbook RSA Encryption: attack algo: factor N and compute d, factoring is not efficient ingeneral, key size >= 2048 bits. Malleability: Given c = Enc(m) = m^e mod N, attacker can forge Enc(ma) = ca^e mod N for any a; Chosen Ciphertext Attack: Given c = Enc(m) for unknown m, attack asks for Dec(ca^e mod N) = d and computes m = da^{-1} mod N.



$\mathrm{Dec}(\mathrm{Enc}(m)) = m^{ed} \bmod N \equiv m^{1+k\phi(N)} \equiv m \bmod N$ by Euler's theorem.

Mitigation: ENC_{pk}(m) = $\mathrm{pad}(m)^e$ mod N, DEC_{sk}(c) = c^d mod N = pad(m).
Textbook RSA signatures: sign: (secret key, message) → signature, (public key, message, signature) → boolean.

Public Key pk	Secret Key sk
$N = pq$ modulus	p, q primes
e encryption exponent	d decryption exponent ($d = e^{-1} \bmod (p-1)(q-1)$)



Works for the same reason RSA encryption does.

Signature forage: In order to get sign(x), the attacker computes z = xy^e mod N for some y, then asks signer for s = sign(z) = z^d mod N. Then sign(x) = sy^{-1} mod N. Mitigation: use padding with RSA, sign hash of *m* but not raw message m.
Bleichenbacker:

Padding:

```
pad(m) = 00 01 [FF FF FF ...  FF FF] 00 [data H(m)]
```

- Signer hashes and pads message, then signs padded message using RSA private key.
- Verifier verifies using RSA public key, strips off padding to recover hash of message.

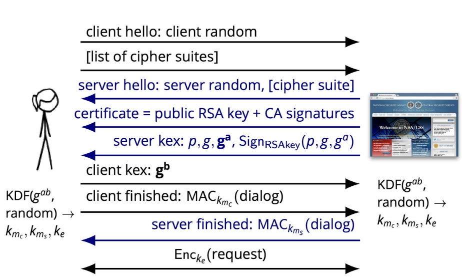
Bleichenbacker low exponent signature forage:
- Without padding length check and signature uses e = 3.
- Hacks:
  1. Construct a perfect cube over the integers, ignoring N, such that

     $s = 0001FF\ldots FF00$[hash of forged message][garbage]

  2. Compute x such that $x^3 = s$.
     (Easy way: $x = \lceil [\text{desired values}]000\ldots0000\rceil^{1/3}$.)
  3. Lazy implementation validates bad signature!

Public-key infrastructure: in-person, fingerprint(SSH), hard code public keys in software, certificate authorities (TLS), web of trust (PGP).
TLS:

TLS 1.2 with Diffie-Hellman Key Exchange
Step 8: The client and server can now send encrypted application data (e.g. HTTP) using their secure channel.



1. The client (browser) tells the server what kind of cryptography it supports. 2. The server tells the client which kind of cryptography it wishes to use. 3. The server sends over its certificate which contains the server's public key and signatures from a certificate authority. 4. The server initiates a Diffie-Hellman key exchange. 5. The client responds with half of the Diffie-Hellman key exchange. 6. The client and server derive symmetric encryption keys from the shared secret using a key derivation function. 7. The client and server verify the integrity of the handshake using the MAC keys they have derived. 8. The client and server can now send encrypted application data (e.g. HTTP) using their secure channel.
CA mitigation: revocation, certificate pinning, certificate transparency; stolen key: diffie-hellman and RSA both allows impersonate serves, but RSA also decrypt all traffic in the past.

Modular Arithmetic

Facts

$$\text{Add: } (a \bmod d) + (b \bmod d) \equiv (a + b) \bmod d$$
$$\text{Subtract: } (a \bmod d) - (b \bmod d) \equiv (a - b) \bmod d$$
$$\text{Multiply: } (a \bmod d) \cdot (b \bmod d) \equiv (a \cdot b) \bmod d$$

Modular Inverse

If $a \cdot b \bmod d = c \bmod d$ we would like $c/b \bmod d = a \bmod d$.

But if $3 \cdot 2 \bmod 4 = 2 \bmod 4$ this says $3 = 1 \bmod 4$. Problem!

Fix: For rationals, $\frac{a}{b} = a \cdot \frac{1}{b}$ $\quad b \cdot \frac{1}{b} = 1$.
Define modular inverse: $\frac{1}{b}$ means $b^{-1} \bmod d$.

- $b^{-1} \bmod d$ is a value such that $b \cdot b^{-1} \equiv 1 \bmod d$.
- Example: $3 \cdot (3^{-1} \bmod 5) \equiv 3 \cdot 2 \equiv 1 \bmod 5$.
- If $\gcd(a, d) = 1$ then a^{-1} is well defined.
- Efficient to compute.

Modular Exponentiation
- Over the integers, g^a = g * g * g * g .. a times, g^a mod d = (((g mod d) * g mod d) … g mod d) mod d.
- Efficient to compute using binary a

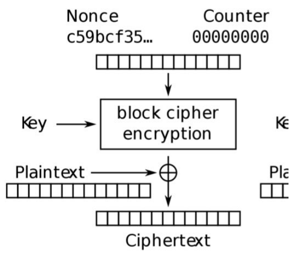
"Inverse" of modular exponentiation: discrete log
- Over the reals, if $b^a = y$ then $\log_b y = a$.
- Define discrete log similarly:
  Input b, d, y, discrete log is a such that $b^a \equiv y \bmod d$.
- No known polynomial-time algorithm to compute this.

More on SIde Channel
Cache: read to *newly-cached* location is fast, read to *evicted* location is slow.
Speculative executioN: branch predictor → mis speculation causes rollback of transient instructions.

Cryptography (authenticity, secrecy, integrity)
SSL: same key used to encrypt many packets
MAC: Message Authentication Code, validate message integrity based on shared secret.
Problem: OTP: one-time-password: once, long as mes
Solution: PRG: Pseudo random key generator

Dangers in using stream ciphers

- Can we use a key more than once?
  - E.g., $c_1 \leftarrow m_1 \oplus$ PRG(k)
    $c_2 \leftarrow m_2 \oplus$ PRG(k)
  - Yes? No?
  - Eavesdropper does: $c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$
  - Enough redundant information in English that: $m_1 \oplus m_2 \rightarrow m_1,\ m_2$



(can also guess from cipher→ "AF" in midway)
• **A block cipher = permutation of fixed-size inputs,** Each input mapped to exactly one output
Electronic Codebook(ECB) mode encryption (bad) can spot original pattern
Cipher Block Chaining(CBC) mode decryption (secure)
Counter (CTR) mode encryption (better)
AES: |m| = |c| = 128 bits, |k| = 128, 192, 256 (USE THIS, better than others)

Modular inverse: "Division" for modular arithmetic

If $a \cdot b \mod d = c \mod d$ we would like $c/b \mod d = a \mod d$.

But if $3 \cdot 2 \mod 4 = 2 \mod 4$ this says $3 = 1 \mod 4$. Problem!

Fix: For rationals, $\frac{a}{b} = a \cdot \frac{1}{b} \qquad b \cdot \frac{1}{b} = 1$.
Define modular inverse: $\frac{1}{b}$ means $b^{-1} \mod d$.

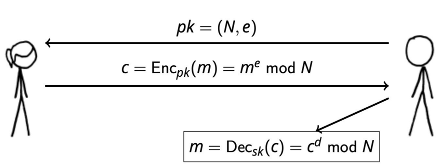
- $b^{-1} \mod d$ is a value such that $b \cdot b^{-1} \equiv 1 \mod d$.
- Example: $3 \cdot (3^{-1} \mod 5) \equiv 3 \cdot 2 \equiv 1 \mod 5$.
- If $\gcd(a, d) = 1$ then a^{-1} is well defined.
- Efficient to compute.

- a message authentication code (**MAC**) in crypto
- Mac(c) = H(c) for a collision-resistant hash f is not secure. Attackers can encode whatever msg they want.
- Can Combine MAC with encryption (integrity for plaintext & ciphertext)
c = Enc_ke(m), t = **MAC**_km(c) => bob verify t = Mac_km(c), m = Dec_ke(c)
- Is Mac_{km} = H(k||m) a secure MAC? A: Not if H is MD5, SHA-1, or SHA-2.
- MD5→ length extension attack, If adversary knows or can guess the length of k||m, they can reconstruct the padding and append additional blocks(MD5, padding 1||0..0|| 64 bits show length of m, total 512 bits) ->H(k||m) assignment

Textbook RSA Encryption
[Rivest Shamir Adleman 1977]

Public Key *pk*
$N = pq$ modulus
e encryption exponent

Secret Key *sk*
p, q primes
d decryption exponent ($d = e^{-1} \mod (p-1)(q-1)$)



$\mathrm{Dec}(\mathrm{Enc}(m)) = m^{ed} \mod N \equiv m^{1+k\phi(N)} \equiv m \mod N$ by Euler's theorem

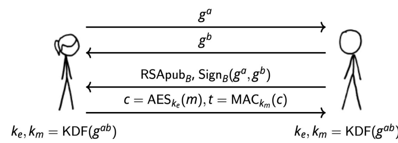
Diffie-Hellman Security: g^a mod p ⇔ g^b mod p. Shared g^(ab) mod p
RSA Security: Best algorithm to break RSA: Factor N and compute d.
Textbook RSA is insecure: multiply-homomorph. Malleability: given ciphertext c = Enc(m) = me mod N, attacker forge ciphertext Enc(ma) = ca^e mod N for any a.
Attack: Chosen ciphertext attack Given a ciphertext c = Enc(m) for unknown m, the attacker asks for Dec(ca^e mod N) = d and computes m = da^{-1} mod N.
RSA PKCS #1 v1.5 **padding**

- Encrypter pads message, then encrypts padded message using RSA public key: $\mathrm{Enc}_{pk}(m) = pad(m)^e \mod N$
- Decrypter decrypts using RSA private key, strips off padding to recover original data: $\mathrm{Dec}_{sk}(c) = c^d \mod N = pad(m)$

attack: perfect cube, place hash in the middle, giving match length of hash+ASN1, thus making the attack. Next section: **Signing**, verify(m, s)...

Constructing a secure encrypted channel

- To ensure confidentiality and integrity: Encrypt and MAC data
- To negotiate shared symmetric keys: Diffie-Hellman key exchange. Key Derivation Function (KDF) maps shared secret to symmetric key.
- To ensure authenticity of endpoints: Digital Signatures



How does Alice know to trust Bob's public signing key?

Attack: Signature forgery

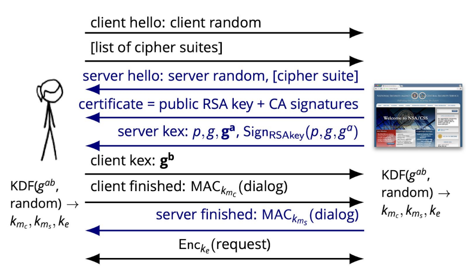
1. Attacker wants Sign(x).
2. Attacker computes $z = xy^e \mod N$ for some y.
3. Attacker asks signer for $s = \mathrm{Sign}(z) = z^d \mod N$.
4. Attacker computes $\mathrm{Sign}(x) = sy^{-1} \mod N$.

Countermeasures:
- **Always use padding with RSA.**
- **Sign hash of m and not raw message m.**

Positive viewpoint:
- Blind signatures: Lots of neat crypto applications.

To ensure an adversary can't reuse a signature later, add some random unique values ("nonces")
1. The client (browser) tells the server what kind of cryptography it supports. 2. The server tells the client which kind of cryptography it wishes to use. 3. The server sends over its certificate which contains the server's public key and signatures from a certificate authority. 4. The server initiates a Diffie-Hellman key exchange. 5. The client responds with its half of the Diffie-Hellman key exchange. 6. The client and server derive symmetric encryption keys from the shared secret using a key derivation function. 7. The client and server verify the integrity of the handshake using the MAC keys they have derived. 8. The client and server can now send encrypted application data (e.g. HTTP) using their secure channel.

TLS 1.2 with Diffie-Hellman Key Exchange
Step 8: The client and server can now send encrypted application data (e.g. HTTP) using their secure channel.



<< Cipher suites: list of options like: TLS_ECDHE_ RSA_WITH_ AES_128_ GCM_SHA256

CA cert: trusting CA, who, pub key info, where to check revoc.
What if a private key gets stolen or compromised?
If an adversary obtains a server certificate private key: • With Diffie-Hellman key exchange, the adversary can: • impersonate the server to anyone. • With RSA key exchange, the adversary can: • impersonate the server to anyone. • decrypt any traffic from now and any point in the past. (cuz RSA exchange use pms, a preshared secret)

Diffie-Hellman Example:

1. Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $a = 4$, then sends Bob $A = g^a \mod p$
   - $A = 5^4 \mod 23 = 4$
3. Bob chooses a secret integer $b = 3$, then sends Alice $B = g^b \mod p$
   - $B = 5^3 \mod 23 = 10$
4. Alice computes $s = B^a \mod p$
   - $s = 10^4 \mod 23 = 18$
5. Bob computes $s = A^b \mod p$
   - $s = 4^3 \mod 23 = 18$
6. Alice and Bob now share a secret (the number 18).

Both Alice and Bob have arrived at the same value s, because, under mod p,

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

More specifically,

$$(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$$

TLS: Transport Layer Security, provides an encrypted channel for application data

HMAC: Mac based on hash function, HMAC is constructed by hashing the XOR of the secret key K with the outer padding opad concatenated with the hash of the secret key K XORed with the inner padding ipad concatenated with the message (more secure, prevent length extension attack).

MAC then Encrypt(SSL), Encrypt and MAC(SSH) → hard to get right
Encryption then MAC(IPSec) → always right (means using the ciphertext to do mac and do c||mac(c))

Advanced side channels:
Padding branches does not work; do not branch on secrets; fold control flow into data flow:

```
if (secret) {
    x = a;
} else {
    x = b;
}
```

➡

```
x = secret * a
  + (1-secret) * x;

x = (1-secret) * b
  + secret * x;
```

Privacy on the web:
How do websites track users(privacy lecture):
third party cookies(cookies for evil.com sent with any request to evil.com)
Tracking contents, include tracking code in URLs
fingerprinting: profile your browsers,extensions,OS,hardware,etc
Can't really avoid, use privacy caring browser, privacy-enhancing xtnsn.
anonymity achieved by proxy→ rewrite sender to anonymous, VPN services allow user to use tunnel traffic
PGP: Pretty Good Privacy, fundamental insights: knowledge about cryptography is public. In theory citizens can circumvent government-mandated key escrow by implementing cryptography themselves (hard to get right)
Tor: Anonymous communication for TCP sessions.

Ethic:
If lawyers or law enforcement are involved, you have already lost. It doesn't matter if you could in theory win the case in the end.
Principle: minimizing harm
nmap is also an attack(unwanted traffic)--> network layer threats
TCP provides: explicit tear done, Reliable in-order byte stream,connection oriented protocol,congestion control,end host has multiple long lived dialogs
Hash collision: length extend of blob in MD5(prefix||blob||suffix)
DHCP response spoofing: racing local network and set victim DNS