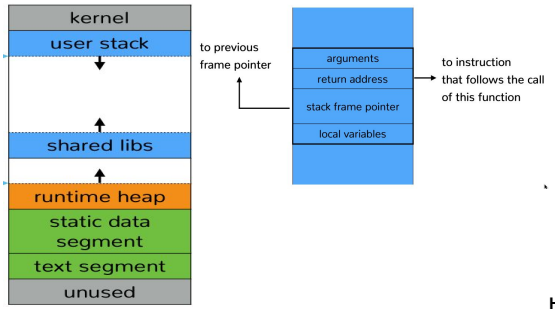
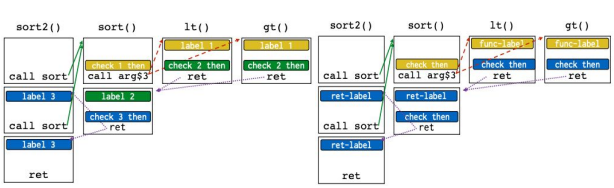


Avoid unsafe funct	Good idea in general	Requires manual code rewrite; Non-lib funct vulnerable; No guarantee everything found; alternatives also err
Stack Canary	No code changes; Only recompile	Performance penalty per return; Only protects against stack smashing; Fails if can read memory
ASLR	No code changes or recompile	32-bit arch get limited protection; fails if can read memory load-time overhead
W^X	No code changes or recompile	Requires hardware support; Defeat by ROP; Not protect JIT code
CFI	No code changes or hardware support; protect many vulns	Performance overhead Requires smarter compiler Need all code available (see)

Stack can	Use targeted write gadget (format strings); pointer subterfuge; overwrite funct ptr elsewhere; memcpy with fixed canary
Separate stack	Find a func ptr and overwrite it to point to shellcode; put buffers & var and func pointers on the user stack such that overwrite func ptrs when compiled to WebAssembly
W^X	Write to stack & jmp to existing code; find system call, replace args
ASLR	Derandomize (brute force for 32; heap spray for 64); find mapped region and call system() with replaced args
CFI	Jmp to func that has the same label, return to more sites
Int Overflow	Truncation (assign 64 to 32); arithmetic overflow (0xffffffff + 2); Signedness bugs (0xffffffff = -1 > some num)

ROP gadgets: overwrite saved %eip to pointer to the first gadget, then 2nd … Make shellcode out of existing code, ending in ret inst (ending in 0xc3 in mem) UAF: overwrite vtable so entry points to attacker gadget (tmp mem violation)

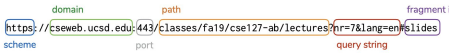
RUID	Inherit from parent; denote who starts the process
EUID	From setuid bit on file executed; determines permission
SUID	Save and restore EUID
SetUID	UID of procee; Setgid: set EGID; sticky: owner/user has the permission



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

Seccomp-bpf: browser side syscall filtering on args
MEM isolation: each process has its own VM
Use page table to translate VM to PM (multi-level page table walking - tree)
ACL - determines page's access control information (R, W, X)

TLB - small cache of recently translated (faster); flush when context switch Iso in VM: extended nested page table entries (VPID)
Defeat: find a bug in the kernel or hypervisor; find a hardware bug; exploit side-channels (cache based - use cache to learn about other process, VM)
Side-channels: evict & time; prime & probe (time, slower means evicted); flush & reload (flush the cache, faster means evicted)
Malware Virus: eventually be executed ⟷ Worm: immediately be executed
HTTP:



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

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

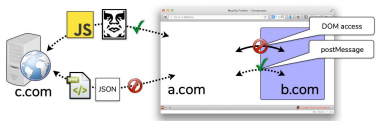
Cookie 2:
name = cookie2
value = mycookievalue
domain = site.com
path = /

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

	Cookie 1	Cookie 2	Cookie 3
heckout.site.com	No	Yes	No
ogin.site.com	Yes	Yes	No
ogin.site.com/my/home	Yes	Yes	Yes
ite.com/my	No	Yes	No

CSRF: use attacker's domain to interact with banks url with user's cookie; submit transfer form from attacker's site with user's cookie
Defense: secret token validation (session-dependent identifier or token so attacker cannot retrive due to SOP - attacker's site has different origin); referer or origin validation: includes url of the previous web page; 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
Http-only: cookie could be accessed by document.cookie. Set HttpOnly prevents leaking cookie.
Injection:
Command injection: execute command on system b ypassing 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; prevention: never build SQL commands by urself. Use parameterized (AKA prepared) SQL; ORM (object relational mappers) (provide interface between obj & DBs)
Cross-Site Scripting (XSS) App takes untrusted data and sends it to a web browser w/o proper validate

Reflected XSS: script reflected back to the user as part of a page from the victim site
Stored XSS: stores the malicious code in a resource managed by the web app (DB)
Defense: Content security policy (CSP): specify domains that browser should consider to be valid sources of exe scripts (set up in HTTP header); whitelist domains
Recall: SOP (prevent by CFI))
Isolate content from different origins



Not strict enough: Third-party libs run with the same privilege of the page; Code within page can arbitrarily leak data; Iframe isolation is limited **Not flexible enough**: Cannot read cross-origin responses

Modern Web Defense Mechanism
Iframe sandbox: Restrict actions iframe can perform;
Whitelisting privileges
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

Iframe sandbox can: Run content in iframe with least privilege; Privilege separate page into multiple iframes
CSP (Content Security Policy): Consider running library in sandboxed iframes (desired guarantee: checker cannot leak password);
Problem: sandbox does not restrict exfiltration; Restrict resource loading to a whitelist
HTTP strict transport security (HSTS): Attackers can force you to go to HTTP vs. HTTPS; HSTS: never visit site over HTTP again

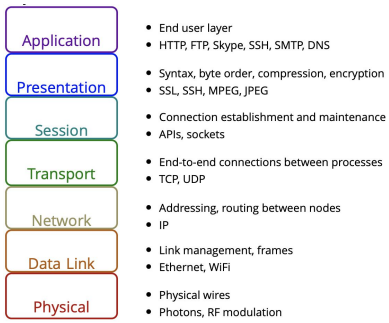
Subresource integrity (SRI): CSP + HSTS can be used to limit damages but cannot really defend against malicious code; Idea: page author specifies hash of (sub)resource they are loading; browser checks integrity; When check fails: 1. Browser reports violation and does not render or execute resource; 2. CSP directive with integrity-policy directive set to report (report but may render or execute)

Cross-origin resource sharing (CORS)
Recall: SOP is not flexible: Problem: cannot fetch cross-origin data Solution: cross-origin resource sharing (CORS);Data provider explicitly whitelists origins that can inspect responsesBrowser allows page to inspect response if its origin is listed in the header
How it works: 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
Example: amazon (multiple domains, Problem: amazon.com cannot read cross origin aws.com data; with CORS amazon.com can whitelist aws.com)

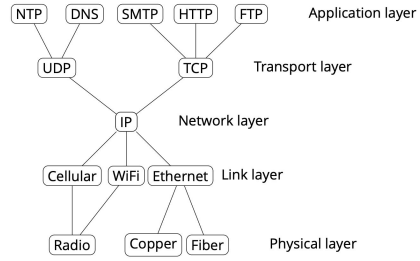
COWL
Provide means for associating security label with data Ensure code is confined to obey labels by associating labels with browsing contexts
Confining the checker with COWL: Express sensitivity of data (checker only receive pw if its context label is as sensitive as the pw); Use postMessage to send labeled pw (at time of sending source, specify the sensitivity of the data)
Other: A temporary err: violation by using a pointer whose ref has been deallocated; A spatial err: violation by dereferencing a pointer that refers to an address outside the bounds of its referent.
Heap corruption: Bypass security checks (isAytgebtucated, buffer size, isAdmin, etc); overwrite function pointers (vtables); each object contains a pointer to vtable; vtable is an array of function pointers; call looks up entry in vtable

OS defense level (coarse → fine): physical machine → VM → OS process → Library → function

Networking:
Protocols: agreement on how to communicate
Include syntax & semantics (how to commu)
Layering: lower layers provide service to layers above; higher uses service below; layers define abstraction boundaries.
Packet encapsulation: Protocol N1 can use service of lower layer N2; Packet P1 of N1 is encapsulated into a packet P2 of N2; the payload of P2 is P1; the control info of P2 is derived from that of P1
OSI architecture:



Basic internet architecture Hourglass

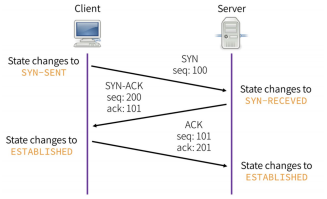


Link Layering: connecting host to local network: most common link layering protocol: **Ethernet** (messages organized into frames; every node has globally unique 6byte MAC media-access-control address; originally a broadcast protocol: every node received every packet → now: switch learns the physical port for each MAC addr and send packets to right port if known; WiFi similar to Ethernet, but nodes can move)
APR (addr resolution protocol): Used to map IP addr to MAC addr on local network; request broadcast to local network; anyone can send
IP: Internet protocol
"Best effort" - no guarantee on delivery; no attempt to recover from fail; packets might be lost, delivered out of order or multiple times, or fragment; Provides hierarchical addr scheme.

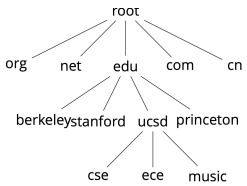
IPv4
- 32-bit host addresses
- Written as 4 bytes in decimal,
- e.g. 192.168.1.1

IPv6
- 128-bit host addresses
- Written as 16 bytes in hex
- :: implies zero bytes
- e.g. 2620:0:e00:b::53 = 2620:0:e00:b:0:0:0:53

Routing: BGP (bad gateway protocol)
Allows router to exchange info about routing tables; maintain global table of routes; each router announces what it can route to its neighbors; routes propagate thru network; Internet organized into (Auto System) with peer, provider or customer relationships between them; rough tree shape
TCP: want abstraction of bytes delivered between apps and hosts
Provides: reliable in-order byte stream; connection-oriented protocol; explicit setup/teardown; end host have multi concurrent long-lived dialogs; congestion control: adapt to network path capacity, receiver's ability to receive
Sequence num: bytes in app data stream numbered with 32-bits seq #; sent in segments; indicates where data belongs in byte seq; in packet header is the seq number if the 1st byte in the payload
Ack: receiver acknowledges received data
FIN/RST: clean close of TCP
Ports: each app is identified by a port number (16 bits); 80-HTTP(web); 443-HTTPS; 25-SMTP(mail); 67-DHCP(host config); 22-SSH; 23-telnet.



UDP: offers no service quality guarantee; a transport layer protocol (wrapper around IP); Useful for app that only requires best effort guarantee (DNS, NTP)
DNS: handle mapping between host names & IP addr; delegatable, hierarchical name space. **Details**: 13 main root services; responses are cached for faster responses; authorizes query based on hierarchy.



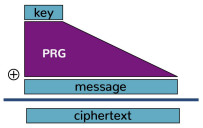
Using Internet Example: connect laptop to WIFI and type ucsd.edu
1. Uses DHCP (Dyna Host Config Proto) to bootstrap on the local network (new host has no IP, not know who to ask; broadcast to 255.255.255.255 with its MAC addr; DHCP responds with config)
2. Makes an ARP request to learn the MAC addr of local router(connection outside local network will be encapsulated in a link-layer frame with local router's MAC addr as destination; laptop encaps each IP packets in a WiFi Ethernet frame addressed to local router; local router decaps frames and re-encodes em to forward on its fiber connection to its upstream ISP; each hop reencodes the link for its own network)
3. Does a DNS lookup on ucsd.edu(laptop learned IP addr of local DNS server from DHCP; each request is a DNS query encapsed in one or more UDP packets encapsed in one or more IP packets; each response tells what authority to query until it learns the right IP for ucsd.edu)
4. Opens a TCP connection to the correct IP (each packet of TCP handsh -ake is encoded in an IP packet encoded as Ethernet frames that are decoded and re-encoded as they pass thru the network;)
5. Sends a HTTP GET request inside TCP connection
6. Based on the HTTP response, performs a new DNS lookup, TCP hands -hake, & HTTP GET request for every resource in the HTML as rendering.

Network Attacks
Goals: confidentiality; integrity; availability
Physical/link layer threats
Evesdropping: violates confidentiality
Injection: violates integrity (Ethernet packets unauthenticated: attacker who can inject traffic can create a frame with any addr they like; packets injection - ARP spoofing)
Jamming: violates availability (physical signals can be overwhelmed or disrupted; radio transmission depends on power and distance)
Network layer threats
Spoofing: set arbitrary source addr (IP packets offer no authentication; source addr in IP set by sender; easy for UDP-based proto, hard for TCP) Packet injection: DHCP response spoofing (DHCP-config host on network; DHCP requests broadcast to local network; local attacker can race real server for response, set victim's N gateway and DNS server to attacker-controlled values; allows attacker to act as invisible)
Set arbitrary destination addr: no authentication of traffic sender as N layer (application: **N scanning & unwanted traffic**)
Misdirection: BGP hijacking (BGP node has connections to a set of trusted neighbors; neighbors share routing info; routes not authenticated-**May provide incorrect routing info that redirects IP traffic**)
TCP threats
On-path injection: "connection hijacking"-if an on-path attacker knows ports and sequence #, can inject data into TCP connection; "RST injection"-can inject RST into connection to stop it, will be accepted if sequence number is within acceptable window (eg: China's firewall)
Blind spoofing: convince a victim to open a TCP connection with spoofed host (forges the initial TCP handshake SYN msg from an arbitrary source; attacker cannot see; random chance of guessing correctly 2^-32)
DNS spoofing:
Malicious DNS server: any DNS server in query chain can lie about responses; **local/on-path attacker:** can impersonate DNS server and send fake response; **off-path attacker:** can try to forge response

Network Defenses
The more N services you run, the more dangerous; so turn off N service unnecessary on each system
N Perimeter Defense: N defenses outside of organization
Firewalls: problem: isolating one part of N from others; Need to filter or limit N traffic;
Types: Personal firewall: run at the end of hosts, has more app/user info N firewalls: intercept and evaluate commu from many hosts; Filter-based: operates by filtering on packet headers; Proxy-based: operates at the level of the application (eg: HTTP web proxy)
N firewall: protect against bad commu; protect services from outside acess; provide outside services to host located inside
Access control policies: a firewall enforces ACP; tells **inbound**(attempts by external users to connect to services on internal machines) and **outbound**(internal users to external services); allow inside users to connect any service but restrict external users to service visible externally; for not mentioned traffic - default deny (add only when user complains)
Filtering Firewalls:define a list of AC rules; check packets against rules; take advan of (source IP & port, destination IP & port, flags-eg.ACK); limitations: a stateless packet filter cannot tell packets associated with a connection from those are not
Circumventing simple F rules: send traffic on port usually allocated for another service; tunneling (encaps one proto inside another; recipient of outer proto decaps to recover inner proto)
Stateful packet filtering is hard: "root" may span packet boundaries and packets may be reordered
N addr Translation(NAT): IP addr do not need to be unique; NATs map between 2 diff addr spaces; NAT has a table of form <client IP><client port><NAT ID>; **protection:** for outgoing packets (check client's IP & port; If found, replace port with NAT ID; else allocate a new NAT ID--> set port); for incoming packets (check destination port as NAT ID; if found, replace destination addr & port with client; else reject); **Pros:** only allows connections to outside that are established from inside; don't need large external addr space; **Cons:** rewriting IP addr is hard; breaks some protos
Application proxies: control apps by requiring em to pass thru proxies Proxy is app-level man-in-the-middle; eg: (SMTP: scan for virus; SSH: log authentication; HTTP: block forbidden URLs)
N Intrusion Detection Systems(NIDS): passively monitor N traffic for signs of attack (eg. look for /etc/passwd); has a table of active connections and maintains a state for each; for unknown connections → **detection**
1.N-Based detection: scan for HTTP requests (**pros**: no need to modify or trust end systems; cover many systems with single monitor; centralized management; **cons:** expensive - 10Gbps link 1M packets/sec; vuln to evasion attacks - incomplete analysis & imperfect observability)
2. Host-based detection: scan args sent to backend programs (**pros:** semantic gap is smaller; no need to intercept HTTPS; **cons:** expensive - add code to each server; still have to consider UNIX filename semantics & other sensitive files & databases; only helps with web server attacks)
3. Log analysis: run scripts to analyze system log files (**pros:** cheap; no escaping issues-logging done by server; **cons:** detection delayed, cannot block attacks; still need to worry about UNIX filename semantics; malware may be able to modify logs)
detection accuracy: FP rate = P[A|not *I*] FN rate = P[not A| *I*] where ***I*** is the event of intrusive behavior, A is the event of detector generating an alarm
Detection tradeoffs: want effective balance (FP & FN) depends on cost of each type of error and on the rate at which attacks occur
Vulnerability scanning: launch attacks yourself. Probe internal systems with a range of attacks. patch/fix/block any that succeed (**Pros:** accurate, proactive - prevent future misuse, intelligence - ignore IDS alarms you know cannot succeed; **Cons:** take a lot of work; not helpful for systems cannot modify; dangerous for disruptive attacks)
Honeypots: 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; **Pros:** provides opportunity to identify intruders, and study what we want to do, divert em from legimate targets.

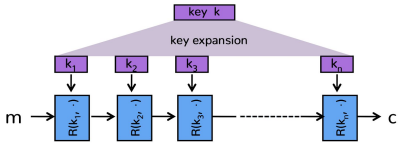
Symmetric Cryptography
Def: a tremendous tool; the basis for many security mechanisms; Not a soln to all security problems, not reliable unless implemented and used properly.
Secure commu: authenticity - parties cannot be impersonated, secrecy - no one else can read msg, integrity - msg cannot be modified
Real-world crypto: SSL/TLS - 1. Browser and web server run "handshake proto"; 2. Browser and web server use negotiated key to symmetrically encrypt data (record layer)
File encryption: files are symmetrically encrypted with a secret key stored encrypted or in tamper-proof hardware, and pw is used to unlock the key so the data can be decrypted.

Symmetric-key encryption: (key, plaintxt) → ciphertxt [Ek(m) = c]
Decryption: (key, ciphertxt) → plaintxt [Dk(c) = m]
One-time key: used to encrypt one msg (email, new key / email)
Multi-use key: used to encrypt multi msgs (SSL same key for packets)
Secrecy against passive eavesdropper: ciphertxt reveals nothing about plaintxt; given Ek(m1) and Ek(m2), cannot tell which plaintxt was encrypted without key.
OTP security: Shannon - without key, ciphertxt reveals no info about plaintxt; problems: can only use key once, key is as long as the msg
Computational cryptography: want the size of secret to be small (theorem: if size of keyspace smaller than size of msg space, info-theoretic security is impossible) **soln: weaken security requirement**
Stream ciphers: problem: OTP key is as long as msg; **soln:** pseudo random key → [Ek(m) = PRG(k) ⊕ m]



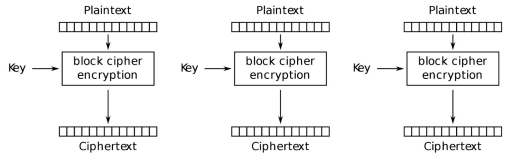
Examples: ChaCha, Salsa…
Dangers in using stream ciphers:
Cannot use a key more than once
Eg: If
c1 ← m1 ⊕ PRG(k)
c2 ← m2 ⊕ PRG(k)
C1 ⊕ C2 → m1 ⊕ m2

Chosen plaintxt attacks: attacker can learn encryptions for arbitrary plain
Block ciphers: crypto work horses: block ciphers operate on fixed-size blocks (3DES: |m|=|c|=64bits |k|=168bits; AES: 128 & 128, 192, 256 bits); a block cipher = permutation of fixed-size inputs (each input mapped to exactly one output) **correct cipher choice: AES**

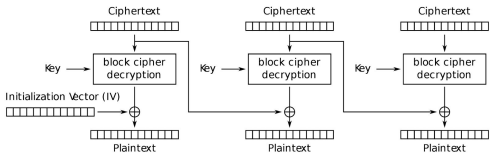


R(k,m): round function for AES-128 (n=10)

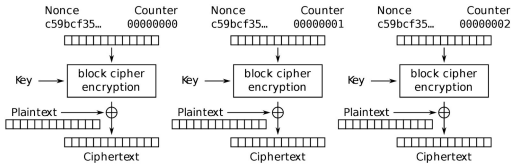
Challenges with block ciphers: block ciphers operate on single fixed-size block; for long msg, and msg that are not block-aligned is hard
Electronic codebook mode encryption - insecure (ECB)



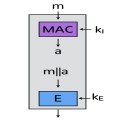
Moderately secure usage of block cipher (CBC)



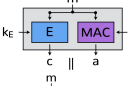
Better block cipher usage (Counter mode encryption - CTR)
Essentially use block cipher as stream cipher



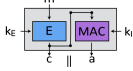
What if ciphertxt can be modified by Eve? Symmetric encryption alone is not enough to ensure security (need to protect integrity of ciphertxt)
Hash functions: maps arbitrary length input into a fixed-size string (|m| can be large but output string is 128-512 bits usually); **properties:** finding a preimage is hard (given h, find m s.t. H(m) = h); finding a collision is hard (find m1, m2 s.t. H(m1) = H(m2))
Bit security: 128-bit hash has 64 bits of security due to **birthday bound**
MD5: msg digest - output 128 bits → **broken!**
SHA1: secure hash algorithm 1 - output 160 bits → **broken!**
SHA2: secure hash algorithm 2 - output 224, 256, 384, 512 bits
SHA3: secure hash algorithm 3 - output arbitrary size
MACs - msg authentication code: validate msg integrity based on shared secret; hard to compute funct without knowing the ke; security: MACk(c) should be unforgettable by an adversary
HMAC: MAC based on hash [MACk(m) = H(k⊕opad || H(k⊕ipad ||m))]
Combining MAC with encryption:
MAC then encrypt (SSL)
- Integrity for plaintxt not ciphertxt
- Issue: need to decrypt before you can verify integrity
- Hard to get right



Encrypt and MAC (SSH)
- Integrity for plaintxt not ciphertxt
- Issue: need to decrypt before you can verify integrity
- Hard to get right

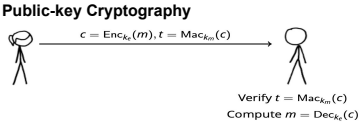


Encrypt then MAC (IPSec)
- integrity for plaintxt and ciphertxt
- always right



Correct encryption soln: use AEAD construction: authenticated encryption with associated data; always use an authenticated encryption mode (combines mode of operation with integrity protection/MAC in the right way)

Public-key Cryptography

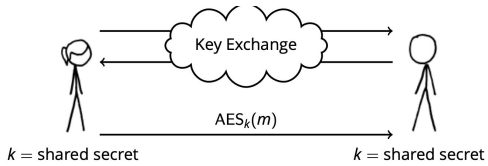


Length extension attacks
Is MACk(m) = H(k || m) a secure MAC? Not if H is MD5, SHA1, 2
Merkle-demgard Hash funct construction: construct a funct that takes arbitrary length inputs from a fixed length compression funct
For MD5:
1. Input m = m1||m2||...||m_l where m are 512-bit blocks
2. Append 1||000...000||len(m) to the last block, as many bits as necessary to make m_l a multiple of 512
3. Iterate
Adversary observes BadMACk(m) = H(k || m) for unknown k and m; adversary would like to forge BadMACk(m ||r) for r of the adversary's choice; a length extension attec allows the adversary to construct BadMACk(m||padding||r) for r of their choice; If adversary can guess the length of k||m then they can reconstruct the padding and append additional blocks corresponding to r to **Merkle-demgard construction**
Secure solution: use a good MAC construction - HMAC

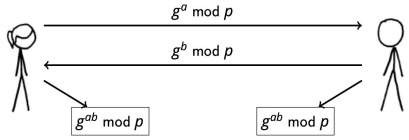
Asymmetric cryptography/public-key cryptography
Insight: separate keys for different operations; keys come in pairs and are related to each other by the specific algorithm
Public key encryption:
Encryption: (public key, plaintxt) → ciphertxt Epk(m) = c
Decryption: (secret key, ciphertxt) → plaintxt Dsk(c) = m
Properties: inverse operations - Dsk(Epk(m)) = m; secrecy - cipher does not reveal plain; anybody with your PK can send secret msg

Modular todo

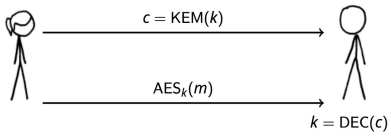
Symmetric cryptography - Public key crypto
Idea # 1: key exchange



Textbook Diffie-Hellman Key exchange:
public param: p a prime, g an integer mod p

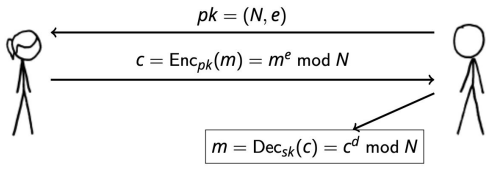


Most efficient algorithm for passive eavesdropper to break
Param selection: p should be ≥ 2048 bits
Insecure against man-in-the-middle: Active adversary can modify Diffie-Hellman msg in transit and learn both shared secrets; allows transparent MITM attack against later encryption; **need to authenticate msg to fix**
Idea # 2: key encapsulation/public-key encryption



Textbook RSA encryption

Public key pk	N = pq modulus; e encryption exponent
Secret key sk	p,q primes; d decrypt exponent (d=e^{-} mod(p-1)(q-1))



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

RSA security
Best algorithm to break RSA: factor N and compute d; factoring is not efficient in general; current key size recommendation: N should be ≥ 2048
Textbook RSA is super insecure: unpadded RSA encryption is homomorphic under multiplication.
Attack: malleability - given a ciphertxt c = E(m) = m^e mod N, attacker can forge ciphertxt E(ma) = ca^e mod N for any a.
Attack: chosen ciphertxt attack - given a ciphertxt c = E(m) for unknown m, attacker asks for D(ca^e mod N) = d and computes m = da^{-} mod N
So use padding on messages (RSA PKCS # 1 v1.5 padding)

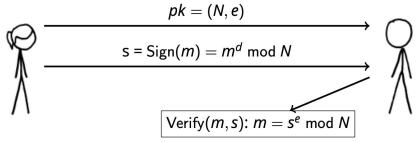
```
pad(m) = 00 02 [random padding string] 00 [m]
```

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

PKCS#1v1.5 padding is vulnerable to a number of padding attacks. It is still commonly used in practice.

Idea # 3: digital signatures
Bob wants to verify Alice's signature using only a public key (signa verifies that Alice was the only one who can have the msg; signa verifies the msg has not been modified in transit)
Signing: (secret key, msg) → signature [Signsk(m) = s]
Verification: (public key, msg, signa) → bool [Verifypk(m, s) = T/F]
Signa properties: verification on signed msg always succeeds - Verifypk(m, Signsk(m)) = True; unforgettability: cannot compute signa for msg m that verifies with public key without corresponding secret key; anybody with your public key can verify if you signed something
Textbook RSA Signatures

Public key pk	N = pq modulus; e encryption exponent
Secret key sk	p,q primes; d decrypt exponent (d=e^{-} mod(p-1)(q-1))



Textbook RSA signature are super insecure
Attack: signa forgery
1. Attacker wants Sign(x)
2. Attacker computes z = xy^e mod N for some y
3. Attacker asks signer for s = Sign(z) = z^d mod N
4. Attacker computes Sign(x) = sy^{-1} mod N
Countermeasures:
1.always use padding with RSA
2. Sign hash of m and not raw msg m
Positive viewpoint: blind signatures: lots of neat crypto applications
RSA PKCS # 1 v1.5 signature padding

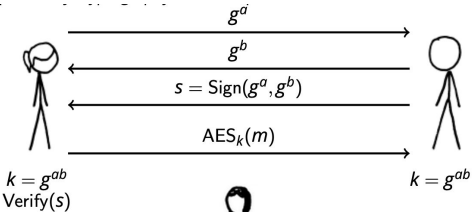
```
pad(m) = 00 01 [FF FF FF ...  FF FF] 00 [data H(m)]
```

- 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.

If a decrypter does not correctly check padding length? Then Bleichenbacher Low Exponent signa forgery attack
If victim shortcuts padding check: just looks for padding format but doesn't check length, and signature uses $e = 3$:
1. Construct a perfect cube over the integers, ignoring N, such that
   $s = 0001FF\ldots FF00[\text{hash of forged message}][\text{garbage}]$
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!

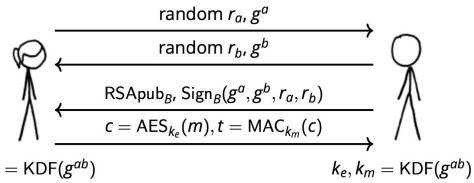
Security for RSA signatures - same as RSA encryption

Putting it all together

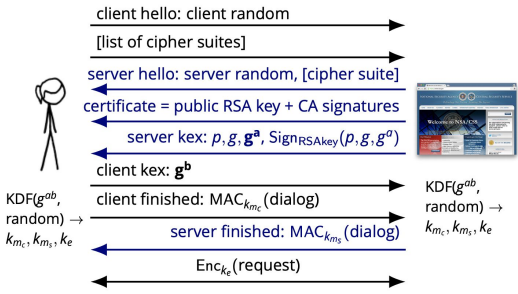


1. Difiie-Hellman used to negotiate shared session key
2. Alice verifies Bob's signature to ensure that key exchange was not man-in-the-middled
3. Shared secret used to symmetrically encrypt the data
All public key cryptography used involves 3 problems: factoring, discrete log mod primes, Elliptic curve discrete log, which can be solved by a general-purpose quantum computer

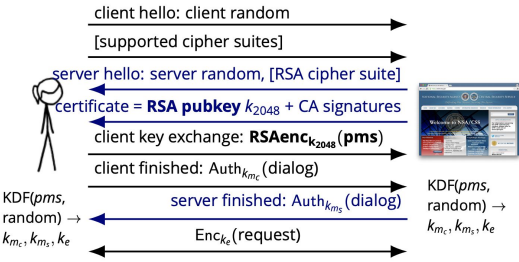
TLS
Constructing a secure encrypted channel
1. To ensure confidentiality and integrity: Encrypt & MAC data
2. To negotiate shared symmetric keys: Diffie-Hellman key exchange. Key Derivation Function (KDF) maps shared secret to symmetric key
3. To ensure authenticity of endpoints: Digital signatures
4. To ensure an adversary cannot reuse signa later, add some random unique values ("nonces")



Public key infrastructure: establishing trust in keys (Ways)
Fingerprint verification: verify a cryptographic hash of a public key through a separate channel, or "trust on first use" (TOFU); used by SSH for host keys; also used by encrypted msg apps like Signal
Hardcode public keys in software: "certificate pinning" used by browser
Certificate authorities: a kind of commercial trusted intermediary; verify public keys and sign em in exchange for money; if you trust the certificate Authority, then you indirectly or transitively trust keys it signs; used for TLS, software signing keys
Web of Trust: establish trust in intermediaries of your choice, then transitively trust the keys they sign again; used by PGP
TLS: transport Layer security: provides an encrypted channel for application data; used for HTTPS: HTTP inside of a TLS session;
TLS 1.2 with Diffie-Hellman Key Exchange
1. The client(browser)tells the server what kind of cryptography it supports
2. The server tells client which kind of cryptography it wishes to use
3. The server sends over its certificate which contains server's public key and signature from a certificate authority
4. The server initiates a Diffie-Hellman key exchange
5. The client responds with its half of 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 app data (eg. HTTP) using their secure channel



TLS prior to 1.3 also supported using RSA public key encryption



If a private key gets stolen?
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
TLS v.1.2 had a lot of vulnerabilities, TLS v.1.3 is being deployed
Major differences:
1. RSA key exchange removed - protects passive decryption attacks
2. Only secure D-H parameters are allowed - against attacks exploiting bad choices of parameters
3. Handshake encrypted immediately after key exchange - limits the amount of metadata visible to passive eavesdropper
4. Protocol downgrade protection - against proto being downgraded to prior insecure versions
TLS v.1.3 deployment difficulties → slower; major reasons
1. HTTPS proxies extremely common in industry
2. Many of them rely on RSA key exchange to make passive decryption and traffic analysis earlier
3. Removing RSA key exchange breaks all these boxes
4. Man-in-the-middle hardware is quite common
5. Bad implementations have hard coeded values like TLS versions and there is no way to update them

Side Channels and Constant-Time Code
Recall: timing info could leak secret info from a running program
Different types: electromagnetic radiation (voltage running thru a wire produces a magnetic field); power consumption (different paths thru a circuit might consume different amount of power); sound/acoustic attacks (capacitors discharging can make noises); timing (diff exec time due to program branches, cache timing attacks); err msg (err msg might reveal secret info to an attacker); fault attacks
Timing attacks on modular exponentiation: RSA performs modular exponentiation: m = c^d mod N; number of multiplications performed leaks Hamming weight of private key; secret-dependent program exe time; turn into full attack by cleverly choosing ciphertxt
Power analysis attacks on modular exponentiation: simple power analysis attacks plot power consumption over time
Memory caches and cache attacks: an attack program runs on the same processor as a victim program; the attack program measures memory access times to determine which data victim loaded into cache. Read to **newly cached** location is fast; read to **evicted** location is slow
Cache Attacks against AES: AES consists XOR, SHIFT, Substitutions; for speed, operations precomputed as a lookup table; table queries dependent on key values; a cache attack can reveal the lookup locations and thus the secret key
Fault Attacks: using mem err to attack a virtual machine

Types of RAM: volatile memory: data retained only as long as power is on; persistent memory like flash or magnetic disks retains data w/o power; **SRAM:** retains bit value as long as power is on w/o fresh; faster, lower density, higher cost; **DRAM:** requires periodic refresh to retain stored data; higher density, lowered cost

Mitigating timing side channels:eliminating all side-channels-impossible

But some are possible - **sweet spot - timing channels:**

good for attacker (remote attackers can exploit timing channels;
co-located attacker can abuse cache to amplify these attacks)
good for defense (can eliminate timing channels; performance overhead of doing so is reasonable)

What introduces time variability?

Floating point time variability (foo(1.0) vs foo(1.0e-323) - later longer)

Some instructions introduce time variability - certain instr take diff time depending on the operands, so input data might leak some info; soln: do not use variable-time instr

Control flow: if-statement on secrets are unsafe (if true takes longer) pad an else block does not work - instr are loaded from cache; hardware tried to predict where branch goes. Soln: do not branch on secrets **but we need branch; instead change to data flow (assume secrete = 1/0)**

Example: if (secret) x = a; → x = secret * a + (1 - secret) * x; problem: control flow that depends on secret data leak info (loops, if, early returns, break, function calls, …)

Mem access patterns introduce time variability: soln: only access mem at public index; example: expressing arr[secret] →

```
for(size_t i = 0; i < arr_len; i++)
    x = CT_SEL(EQ(secret, i), arr[i], x)
```

Overall solution: constant time programming: but writing CT code is unholy → fix: design new programming languages (FaCT is guaranteed); transform code when possible;

Privacy and Anonymity

Def of privacy: secrecy, anonymity, solitude

Pretty Good Policy (PGP)

Written by Phil Zimmermann in 1991
- Response to US Senate bill requiring crypto backdoors (didn't pass)

Public key email encryption "for the masses"
- Signatures, public key encryption, or sign+encrypt

Key management
- Public keyservers
- Web of trust: users sign other users' keys

Grand jury investigated Zimmermann 1993–1996
- No indictment issued, but was a subject for violating export controls

Fundamental insight: Knowledge about cryptography is public. In theory citizens can circumvent government-mandated key escrow by implementing cryptography themselves.

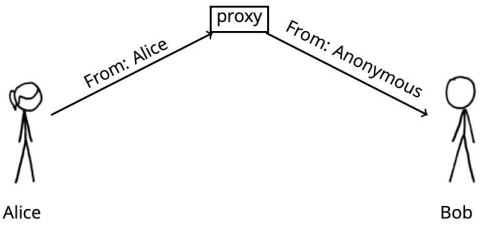
Vulnerabilities: outdated cipher choices; does not authenticate encryption with a MAC or authenticated encryption mode

In modern era: commercialized in the 90s, most recently developed by Symantec; GnuPGP and libgcrypt quite widely used;

Modern protocols typically: use diffie-Hellman to negotiate ephemeral keys; use long-term authentication key with out of band fingerprint verification; offer "forward secrecy" and "deniability"

Anonymity:

Via tunneling or proxies: a proxy can rewrite metadata

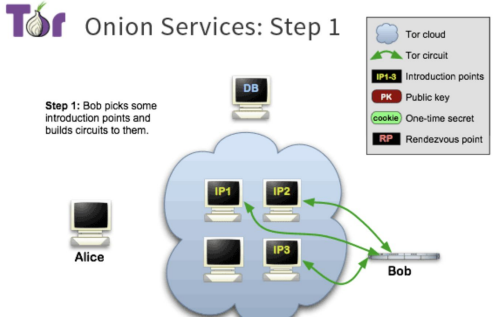


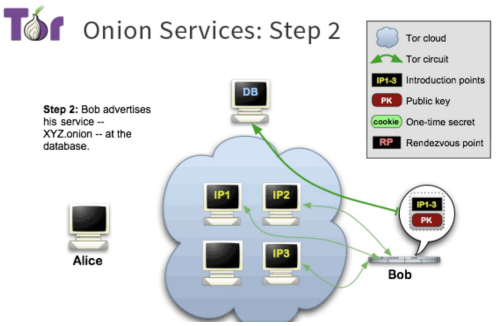
Tor: Anonymous commu for TCP sessions:

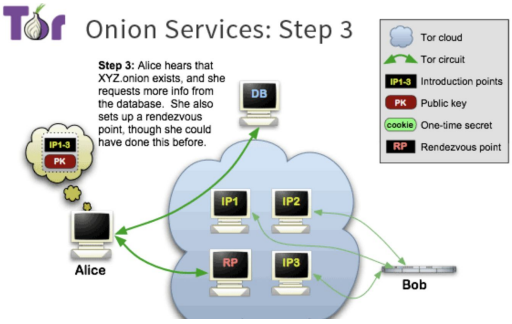
Desire properties: network attacker watching client traffic cannot see destination; destination server does not see client IP addr; network node cannot link client and server; fast enough to support TCP streams and network applications

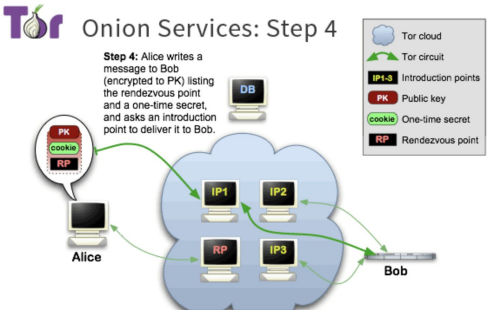
Current state: a non-profit organization, active academic research, deployed around the world

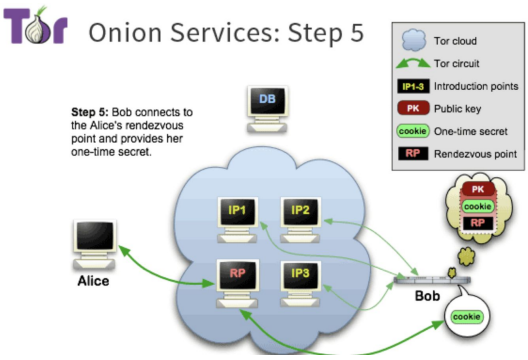
Tor allows anonymous servers

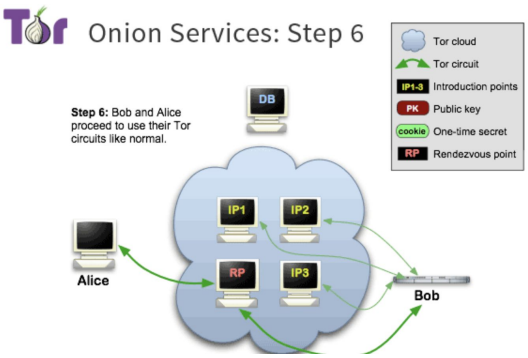












Privacy on the web

Companies like Google, Facebook, Twitter, Microsoft, Amazon, Target, Walmart, . . . make a lot of money from tracking users.

For some of these companies you are the product. So tracking you is their business.

How do websites track users?
- Third-party cookies: recall that cookies for `trackme.com` are sent with any request to `trackme.com`, even if you're on `cnn.com`.
- Tracking content: Sites include tracking code into URLs (e.g., advertisements, videos, marketing emails, etc.)
- Fingerprinting: sites profile your browser, extensions, OS, hardware, screen resolution, fonts you have installed, etc.

What can you do?

Cannot really avoid; use browser that cares about your privacy(FireFox)

Ethics, Law, and Policy

Ethics: try to be a good person

Legal issues: do not violate laws

Legal or Ethical Principles:

Respect other people's property: hacking your own password

On your own machines (probably ok, possible exception: DMCA)

On someone else's (get permission → ok; might be CFAA violation)

CFAA (computer fraud and abuse act)

Whoever intentionally accesses a computer without authorization or exceeds authorized access, and thereby obtains information from any protected computer…

The punishment for an offense…
- a fine under this title or imprisonment for not more than one year, or both…,
- a fine under this title or imprisonment for not more than 5 years, or both… if—
  (i) the offense was committed for purposes of commercial advantage or private financial gain;
  (ii) the offense was committed in furtherance of any criminal or tortious act…; or
  (iii) the value of the information obtained exceeds $5,000

Ethical principles: MIN the harm

Example: SYN scanning

Scanning public host is legal but generate many complaints

Used by attacker to find vulnerable hosts; used by researchers to measure networks; doing research on open network means:

1. Publicly identifying the purpose of the research
2. Providing an opt-out mechanism
3. Not launching attacks
4. Avoiding overwhelming your or other's networks or crashing hosts

Example: botherding - taking over a botnet

Interfering with a legal botnet is illegal

Digital Millennium Copyright Act (DMCA)

Personal and privacy rights

Principle: informed consent

Example: Jason Fortuny posted fake sex ad on Craigslist as a woman

Received hundreds of replies and posted em online

Unethical but not clear about legal or illegal

FISA background: 1978 foeign Intelligence Surveillance Act

Law enforcement access policy: is it preferable to have law enforcement or intelligence?

Stockpile software vulnerabilities, write targeted malware, and hack into targets when desired; mandate encryption backdoors or otherwise enable mass surveillance

Disclosure options for security flaws
- Develop fully weaponized malware and distribute on black market
- Tell no one
- Sell vulnerability to middleman and don't report to vendor
- Report to vendor only
- Report to vendor and receive bug bounty
- Report to vendor, wait for fix, report to public ("responsible disclosure")
- Report in full to public immediately ("full disclosure")

Process of reporting vulnerabilities
- Some vendors have sensible reporting process
  - E.g., Firefox and Chrome teams respond and react quickly, easy to work with on fixing bugs, etc.
- Some vendors less so
  - E.g., Send email through an intermediary, receive ACK, no real conversation.
  - E.g., Send email, poke individual folks for replies, no replies. Give up.
- Some vendors are playing catch up
  - E.g., Reported OOB write vulnerability, security "team" replied with "not a security bug." Later freaked out about public disclososure of OOB read vulnerability. Now there is a working group dedicated to security, slightly better definition of an attacker model, and reasonable reporting method: HackerOne.
- Some vendors are the worst: they will try to gag/sue you

Bug bounty programs: $$ for bugs