C.I.A: Confidentiality, Integrity, Availability
AES: 128/192/256 bit key, 128 bit block, 10 rounds for 128 bit key, 12 for 192, 14 for 256 (reduced round version broken)
ECB mode: direct encryption, cannot hide pattern
CBC mode: plaintext XOR IV/prev ciphertext, then encrypt; decrypt ciphertext then XOR with IV/prev ciphertext
CFB (Cipher Feedback) mode: plaintext XOR encrypted IV/prev ciphertext to encrypt; ciphertext XOR encrypted IV/prev ciphertext to decrypt
CTR mode: plaintext XOR encrypted nonce/counter to encrypt; ciphertext XOR encrypted nonce/counter to decrypt, parallel, precompute
Stream vs Block Cipher: stream enc one character (could be block of bits/bytes) at a time using key stream, while block groups the plaintext & enc
Stream Cipher Examples: RC4, A5/1 for GSM (64 bit key, A5/2 is weaker), SEAL (fast, optimized for 32bit arch and lots of RAM)
One-way function $f : X \to Y$: for all $x \in X$, $f(x)$ easy to find, but given $y \in Y$, find x such that $f(x) = y$ is hard
One-way trapdoor function: looks like one-way but trapdoor (knowledge making it easier to do inverse) exists
RSA: Public keys: n, e; Private keys: p, q, d; $n = pq, m = \phi(n) = (p-1)(q-1), e \in [1, m-1], \gcd(e, m) = 1, ed = 1 \bmod m$, trapdoor is $\phi(n)$, without it finding d is hard
Hash function: input arbitrary size message, output fixed size block, expected to be uniformly distributed, one bit different all bits likely to be different
Cryptographic hash function: one-way (pre-image resistant), second pre-image resistant (given m_1 hard to find m_2 such that $h(m_1) = h(m_2), m_1 \neq m_2$), collision resistant (hard to find 2 messages with same hash)
Hash examples: MD5 (128 bit digest, 512 bit msg block, 2^{64} operations for collision brute force), SHA1 (160 bit digest, similar to MD5, 2^{80} for brute force)
Message Authentication Code (MAC): $Mac(k, m) \to t, Ver(k, m', t) \to True/False$, k–key, m, m'–original & transmitted message, t–tag
CBC-MAC: use CBC to encrypt, last block of ciphertext is used as the MAC
HMAC: $h((k \oplus opad)||h((k \oplus ipad)||m))$, h–iterated hash function, k–key, $||$–concatenation, opad & ipad–one block long 0x5c and 0x36 constants
Digital Signature: ensures integrity & authenticity, easy to generate & verify, unforgeability, sign on message digest using private key, verify using public key
Key Establishment: make shared secret key available to 2 or more parties; methods: courier, trusted authority, public channel
Key Establishment Threats: disclosure, modification, replay, origin of keys (impersonation)
Key Transport: encrypt session key, replay attack issue, add timestamp (time) and identity
Key Agreement: all cooperatively agree on a shared result using function, no single party can predict/determine the shared key beforehand
Challenge-Response: authentication using nonces, also add identity, can include session key inside (agreed from both parties or just one)
Shamir's no-key protocol: $K^a \bmod p; (K^a)^b \bmod p; (K^{ab})^{a^{-1}} \bmod p$; K = session key, p is selected so prime discrete log is hard, start from A
Key Establishment with server: Trusted Third Party (TTP), Key Distribution Centre (KDC, supplies session key), Key Translation Centre (KTC, enables session key chosen by 1 user to be available to others), A & B have shared long term keys with TTP (KDC & KTC)
Needham-Schroeder Protocol: $A \to TS : A, B, N_a$
$TS \to A : E(K_{as}; K, B, N_a, E(K_{bs}; K, A))$
$A \to B : E(K_{bs}; K, A); B \to A : E(K : N_b) : A \to B : E(K; N_b - 1)$
PKC One-pass key transport: encrypt session key with public key, add identity, timestamp, sequence number to prevent replay attack, also IV
Needham-Schroeder Public Key Protocol: $E(K_b; K_1, A); E(K_a; k_1, k_2, B)$ $E(K_b; K_2)$; start from A, early version is missing B in step 2
Diffie-Hellman Key Agreement: agree on $g, p, p = 2q+1$, where both p and q are prime (so p is safe prime), private keys are $x_a, x_b, 1 \leq x_a, x_b \leq p$, public keys are $g^{x_a}, g^{x_b} \bmod p$, session key is $g^{x_a x_b} \bmod p$
Station-to-Station: $g^{x_a}; g^{x_b}, E(K; Sig(sk_a; g^{x_b}, g^{x_a}))$
$E(K; Sig(sk_b; g^{x_a}, g^{x_b}))$; start from A, DH with MITM protection
Perfect Forward Secrecy: if old session keys remain secure when the long term keys are compromised
Key Confirmation: one party is assured that all other parties actually have possession of a particular secret key
Implicit Key Authentication: one party is assured that only specifically identified parties can derive a particular key
Explicit Key Authentication: key confirmation + implicit key auth
Key Authentication (Public Key): bind identity of owner and public key
Certificate: publicly verifiable record which associates user and public key, generated by trusted authority (CA, using digital signature), can verify based on static info (offline) or dynamically with servers (online)
Certificate Authority: trusted by subs, maintains and issues certs, identifies entity who submitted Cert Signature Request, has public & private key pair
Certificate Revocation: not before & not after date, compromised private key (either CA or user), user no longer certified, use Certificate Revocation List or Online Certificate Status Protocol
Self-Signed Certificates: user signed (no CA), root CA (trusted by other subscribers), root widely trusted, user limited trust (no defense against MITM)
Registration Authority: Assess relationship between user identity and public key (verify user request for certificate), can be co-implemented with CA
Public Key Infrastructure: set of hardware, software, people, policies, procedures needed to create, manage, store, distribute, revoke public key certificates
PKI objective: Enable secure, convenient, efficient acquisition of public keys
PKI X.509 (PKIX): formal, generic model based on X.509, provides set of management functions (APIs), components: CA, RA, end entity (user), repositories (stores certs & CRLs), CRL issuer (optional component to publish CRLs)
PKIX functions: registration, initialisation, certification, key pair recovery, key pair update, revocation request, cross certification
Authentication: proving identification (asserted identity → often well known, predictable or guessable, ex: email, account name)
Authentication Mechanisms: password (store with hash + salt to hide duplicate passwords), token (active & passive), biometric (not exact match, hard to change, may not be unique), remote (usually use challenge-response), multi factor, SSO
Authentication via Hash Chain: user send $H_{i-1} = h^{i-1}(H_0)$, server check if $H_i = h(H_{i-1})$, vulnerable to desync (for anonymous), DoS, hash function must be secure
Anonymous Authentication: use server public key to encrypt identity, or use hash chain for symmetric (init chain with ID & seed, start from server; $a; h(v_i, a, b), b; h(v_{i+1}, a, b+1)$; session key can be $h(v_i, a, b+1)$, maybe use pseudonym as well)
Access Control: limit who can access what in what ways, only useful with authentication, has subjects (users, process) → access request/operation (rwx, copy) → reference monitor (enforce access control, always invoked, tamperproof, verifiable, usually part of OS) → objects (files, tables, programs, hardware)
Access Control Models: confidentiality based (protect against unauthorised reading), integrity based (protect against unauthorised writing)
Access Control Mechanisms: identification, authentication, authorization (give permission based on ACL), accountability (ensure all actions linked to authenticated identities, ex: system logs)
Mandatory Access Control: use data classification schemes to give users & data owners limited control over access to information resources
Access Control Matrix: each column is permissions per object (ACL), each row is permissions per subject (capability)
Access Control List: list of subjects and corresponding operations allowed on one object, good when users manage their own files, can set default access right for users, data-oriented protection, easy to change rights to object
Capability: list of objects and corresponding operations allowed for one subject, easy to delegate, easy to add/delete users, more difficult to implement
Discretionary Access Control: implemented at discretion/option of the data user, allows subjects to pass permissions to any other subjects
Nondiscretionary Access Control: strictly enforced MAC, managed by central authority in organization
Role Based Access Control: based on roles (collection of permissions) individual users have as part of organization, is nondiscretionary, has sessions (mapping of one user to possibly many roles), hierarchical RBAC can have roles associated with other roles
RBAC Separation of Duty: user cannot be authorised (static) or simultaneously activated (dynamic) for/in roles in which exist conflict of interest, to prevent fraud, dynamic restricts combinations in session
Advantages of RBAC: allows efficient security management (administrative roles, role hierarchy), principle of least privilege (minimize damage), prevent fraud (above), allows grouping of objects/users, encompasses DAC & MAC
Multilevel Security: subjects and objects correspond to security levels, clearance for subjects, classifications for objects, on access compare clearance & classification
Bell-LaPadula (BLP) Model: lattice with compartments & ACM, confidentiality based AC model, no read up/simple security condition (subject can read object if clearance $\geq$ classification and actually allowed to read), no write down/*-property (subject can write to object if clearance $\leq$ classification & actually allowed to write)
Basic Security Theorem: consider system with secure initial state (simple security & *-property hold) & set of state transitions, if all state transitions secure then system always secure
Biba Model: much of basis is same as BLP, integrity based AC model, no write up (subject can write to object if clearance $\geq$ classification & can actually write), no read down (subject can read object if clearance $\leq$ classification & can actually read, can trust higher classification)
OS Security: prevent unauthorised user from gaining access to system, authorising permissions for different subjects and objects for access control, securing shared resources (prevent users from accessing resources beyond its security level & interfering with other programs)
OS Security Techniques: authentication, authorization, auditing (record logs for what, when, where, how user behave in system for later analysis)
UNIX File System Security: use ACL and DAC; user has username, UID (can be associated to multiple usernames), group (has GID, can give entra permission to users); nobody user does not own files, no group, restricted privileges, can be used to access files with no permissions & network service daemon
UNIX Authentication: use /etc/passwd, actual passwords stored in /etc/shadow (owned by root, salt per user), Process Effective User ID (EUID) compared against file UID, compare GID first
UNIX File Permissions: use mode/permission bits, control which subject can access and how, setuid and setgid allows process to run with higher privilege than user/group that called it, chroot to restrict access
Kerberos: Authentication and Authorisation Infrastructure (AAI), has Authentication Server (AS, centralized & trusted, issues long lifetime tickets for whole system), Ticket Granting Server (TGS, issues short lifetime tickets, can be many), Service Server (S/V, provide different services)
Kerberos Reasons: user impersonation, network address impersonation, eavesdropping, replay, etc
Kerberos v4 Limitations: stuck with DES (v5 allows any), limited ticket lifetime (8 bits, v5 can specify start & end), cannot forward authentication (access on behalf of another), offline double encryption in steps 2 & 4, dictionary attack in step 2
Kerberos v4 Protocol: (step 0: user enter password to client machine)
$C \to AS : ID_c, ID_{tgs}, TS_1$; [AD: Address]
$AS \to C : E(K_c; K_{c.tgs}, ID_{tgs}, TS_2, Lifetime_2, Ticket_{tgs})$
$Ticket_{tgs} = E(K_{tgs}; K_{c.tgs}, ID_c, AD_c, ID_{tgs}, TS_2, Lifetime_2)$
$C \to TGS : ID_v, Ticket_{tgs}, Authenticator_c$
$Authenticator_c = E(K_{c.tgs}; ID_c, AD_c, TS_3)$
$TGS \to C : E(K_{c.tgs}; K_{c.v}, ID_v, TS_4, Ticket_v)$
$Ticket_v = E(K_v; K_{c.v}, ID_c, AD_c, ID_v, TS_4, Lifetime_4)$
$C \to V : Ticket_v, Authenticator_c$
$Authenticator_c = E(K_{c.v}; ID_c, AD_c, TS_5)$
$V \to C : E(K_{c.v}; TS_5 + 1)$
Kerberos Inter-Realm Protocol: (step 0–2 unchanged)
$C \to TGS : ID_{tgs_r}, Ticket_{tgs}, Authenticator_c$ (auth same as before)
$TGS \to C : E(K_{c.tgs}; K_{c.tgs_r}, ID_{tgs_r}, TS_4, Lifetime_4, Ticket_{tgs_r})$
$Ticket_{tgs_r} = E(K_{tgs.tgs_r}; K_{c.tgs_r}, ID_c, AD_c, ID_{tgs_r}, TS_4, Lifetime_4)$
$C \to TGS_r : ID_{v_r}, Ticket_{tgs_r}, Authenticator_c$
$Authenticator_c = E(K_{c.tgs_r}; ID_c, AD_c, TS_5)$
$TGS_r \to C : E(K_{c.tgs_r}; K_{c.v_r}, ID_{v_r}, TS_6, Ticket_{v_r})$
$Ticket_{v_r} = E(K_{v_r}; K_{c.v_r}, ID_c, AD_c, ID_{v_r}, TS_6, Lifetime_6)$
$C \to V_r = Ticket_{v_r}, Authenticator_c$
$Authenticator_c = E(K_{c.v_r}; ID_c, AD_c, TS_7)$
Kerberos v5 Protocol: (N: nonce, R: realm, options is a request for flags to be set in ticket, subkey is sub encryption key which defaults to session key,

Seq# to prevent replay)
$C \rightarrow AS : Options, ID_c, R_c, ID_{tgs}, Times, N_1$
$AS \rightarrow C : R_c, ID_c, Ticket_{tgs}, E(K_c; K_{c.tgs}, ID_{tgs}, Times, N_1)$
$Ticket_{tgs} = E(K_{tgs}; Flags, K_{c.tgs}, R_c, ID_c, AD_c, TS_2, Times)$
$C \rightarrow TGS : Options, ID_v, Times, N_2, Ticket_{tgs}, Authenticator_c$
$Authenticator_c = E(K_{c.tgs}; ID_c, R_c, TS_1)$
$TGS \rightarrow C : R_c, ID_c, Ticket_v, E(K_{c.tgs}; K_{c.v}, Times, N_2, R_v, ID_v)$
$Ticket_v = E(K_v; Flags, K_{c.v}, R_c, ID_c, AD_c, Times)$
$C \rightarrow V : Options, Ticket, Authenticator_c$
$Authenticator_c = E(K_{c.v}; ID_c, R_c, TS_2, Subkey, Seq\#)$
$V \rightarrow C : E(K_{c.v}; TS_2, Subkey, Seq\#)$

Web Service Security: flexible, designed to be used as the basis for construction of wide variety of security models including PKI, Kerberos, SSL
Simple Object Access Protocol (SOAP): lightweight protocol which supports structured message (XML based) in decentralised and distributed environment, does not define app semantics, but to provide mechanism/framework to express app message semantics
SOAP Structure: SOAP envelope (SOAP header (header blocks), SOAP body (message body))
Security Assertion Markup Language (SAML): XML based standard for exchanging security information between entities (enhance SOAP security), can be used for FIM and SSO
SAML Entities: Service Provider (SP, SAML enabled service), Subject (S, request log in to SP), Identity Provider (IdP, SAML enabled system that can authenticate S and make assertions about identity)
SAML Authentication: S request login to SP, SP sends S an auth request, S relay to IdP, IdP auth S & return response (SAML token, has assertions) to S, S relay response to SP, SP read response, if authorised apply privileges
SAML Assertions: claim, statement or declaration of fact made by SAML authority, to assert characteristics and attributes of S (bob is student of uni, email is e@m.c)
SAML Assertion Containment: SOAP body (SAML response (Response Header, SAML Assertion (statements)))
SAML Assertion Types: authentication (S was authenticated by particular means at particular time), attribute (subject is associated with attributes), authorization decision (request from S to access resource granted/denied), can also have custom statement
SAML Security: mutual authentication (impersonation), message integrity, confidentiality, replay, MITM, can encrypt and/or sign security assertions, can also use SSL/TLS
Single Sign On (SSO): single log in for multiple apps & systems, each app has agreement between them, authenticate in one place
Federated Identity Management (FIM): apps agree/trust on same identity provider to refer to single user (may be known by different name), FIM gives SSO but SSO may not result in FIM
OAuth: authorization standard rather than authentication, purpose is authorising 3rd party apps to access APIs on user's behalf, enforce least privilege
OAuth Roles: Resource Owner (user with password protected online account, like SAML subject), Resource Server (server with APIs), Client (app attempting to access APIs, like SAML SP), Authorisation Server (can authenticate resource owner and grant client access to resource server, like SAML IdP)
OAuth Authorisation Flow:
0. client register to auth server, auth server assigns client ID and client secret
1. client redirect RO user agent to auth endpoint & prepare auth params
2. auth server authenticates RO, RO accept/deny requested operations
3. auth server redirects user agent back to client with authentication code
4. client requests access token from auth server by including auth code
5. auth server authenticates client, validates auth code, respond with access token & optional refresh token
6. client uses access token to access protected resources, refresh token could still work until revoked

OAuth & SAML: SAML designed for web browsers, OAuth works with native apps, SAML may require encryption & signature for assertion content, OAuth uses TLS/SSL to provide confidentiality & integrity
Network Layers: OSI below, TCP/IP missing 1, 5, 6, and rename 3 to Internet

n	OSI	Protocols	Use
7	Application	HTTPS, SMTP, FTP	Provides functions needed by users
6	Presentation	SSL	Converts diff representations
5	Session		Manages task dialogs
4	Transport	TCP, UDP, SSL/TLS	Provides end-to-end delivery
3	Network	IPv4/6, IPSec, ARP	Sends packets over multiple links
2	Data link	Ethernet, 802.1x, L2TP/PPTP	Sends frames of info
1	Physical		Sends bits as signals

OSI Security Architecture: Security Attack (reasons, action that compromises security of info owned by org or individual), Security Mechanisms (tools, process designed to detect, prevent or recover from security attack, ex: encryption, signature), Security Services (goal, processing/communication service that enchances security of data processing and info transmission, ex: CIA)
IPSec (IP Security): has source authentication, message authentication & integrity check, data confidentiality, access control, implemented as extension headers in IP packet
IPSec Security Protocols: Authentication Header, Encapsulating Security Protocol

	AH	ESP (enc)	ESP (enc+auth)
Access Control	✓	✓	✓
Connectionless Integrity	✓		✓
Data origin auth	✓		✓
Anti-replay	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow config		✓	✓

IPSec Operation Modes: tunnel (2 IP headers, outer has IPSec processing destination, inner has source & ultimate destination), transport (1 IP header with apparent source & ultimate destination)
IPSec Security Association (SA): unidirectional, logical connection that provide security services to a traffic stream between 2 IP nodes, serves as contract between 2 or more entities & completely specifies how they use security services to communicate securely
IPSec SA Parameters: AH/ESP authentication algorithm, ESP encryption algorithm, keys, IVs, IPSec operation mode & lifetime (if PKI used this must not exceed cert validity)
IPSec Internet Key Exchange (IKE): phase 1 establishes SA to secure own traffic (mutual authentication & key establishment), phase 2 establishes another SA to provide security to app data
IPSec IKE Features: Cookies to prevent DoS (make target keep calculating keys, prevent by force cookie exchange first), params for DH can be negotiated, nonces to protect against replay
IPSec IKE Phase 1 Main: $list; algo; g^x, N_a; g^y, N_b; E(K_e; A, Sig_a(M_{ab}))$; $E(K_e; B, Sig_b(M_{ba}))$; Start from A (also for all IKE below), all steps include cookies (first C_a only, then both), $M_{ab} = MAC_{KM}(g^x \| g^y \| C_a \| C_b \| list \| A)$, for other one flip g and C, also change A to B, KM derived from $(N_a \| N_b, g^{xy})$, K_e derived from KM, all g is mod p
IPSec IKE Phase 1 Aggressive: $list, g^x, N_a, A; algo, g^y, N_b, B, Sig_b(M_{ba})$; $Sig_a(M_{ab})$; No identity protection
IPSec IKE Phase 2: share phase 1 SA with each other, $CP, SPI_a, N_a, g^a; CPA, SPI_b, N_b, g^b; ack$; After SA transfer always send X (phase 1 cookies) & Y (distinguish phase 2 session from others), CP is crypto proposal, CPA is CP accepted, ack is ready to accept now, SPI is sec param index (id SA for packet), all protected using phase 1 encryption & integrity
Secure Socket Layer/Transport Layer Security: OSI presentation & transport layers, has connection (P2P relationship, transient, associated with one session) & session (association between server & client, created by handshake protocols, defines set of crypto security params)
SSL Record Protocol: provides confidentiality (encrypt using key from handshake) & integrity (MAC using key) for SSL connections, all other SSL on top
SSL Change Cipher Spec Protocol: contains single byte with value 1, switch pending state to current state, so no need to renegotiate connection
SSL Alert Protocol: convey SSL related alerts, compressed & encrypted, contains 2 bytes (1st contains 1 if fatal, 2 if warning, 2nd contains status of certificate & other specific alerts)
SSL Handshake Protocol: allow server & client to authenticate each other, negotiate encryption & MAC algo, keys, used before app data transmission
SSL Handshake Procedure: client-hello; server-hello, server-cert*, server-key-exchange*, cert-request*, server-hello-done; client-cert*, client-key-exchange, cert-verify*, change-cipher, finished; change-cipher, finished; * optional, hellos have SSL version, timestamp, nonce, session ID, client has list of algo for encrypt & compress, server selects, server-key-exchange only for cases like anon DH, sign short ephemeral public key, cert-verify is for client cert, contains signed hash of preceding messages
SSL Key Exchange Methods: RSA, Fixed DH, Ephemeral DH (most secure), Anonymous DH (no authentication, suffer from MITM)
TLS: uses HMAC, SSLv3 concatenates key rather than XOR, supports all SSLv3 exchange teachniques except Fortezza, minor diff on cert types
HTTPS: HTTP over SSL, use port 443 instead of 80, encrypt URL of document, contents of document, forms & header, cookies, TLS handshake then HTTP request, when closing have close in HTTP record, TLS close-notify alert, close TCP connection, handle TCP close before alert sent/completed
Secure Shell (SSH): general secure channel for network apps, on top of TCP, user authentication & connection protocol (support multiple conections over single transport layer channel, reuse) on top of transport layer protocol (initial connection, server authentication, sets up secure channel by using fresh shared secret to create other keys)
SSH Local Port Forwarding: (SSH server as proxy) client sets up connection to web server, select unused local port 8080 & config SSH to accept traffic from 8080 destined for 80 on server, client informs SSH server to create connection to 80, SSH client takes bits sent to 8080 and send to SSH server, SSH server decrypt & send to web server 80, SSH server takes bits from 80 and send to client
SSH Remote Port Forwarding: (SSH client as proxy) from work, connect via SSH to home, config SSH server to listen to local port 22 & deliver data across SSH addressed to remote port, config SSH on home to accept traffic on 222, now have SSH tunnel for remote logon to work
Wired Equivalent Privacy (WEP): RC4 40 bit key + 24 bit IV, WEP2 uses 104 bit key + same IV, aim to have access control & privacy, use CRC32 (non crypto hash), ((M + ICV) XOR keystream) + IV
WEP weaknesses: Key management (same key for authentication & encryption), key size too short, only one way authentication, integrity
Extensible Authentication Protocol (EAP): authentication framework, supports multiple methods, operates over data link layer
IEEE 802.1X: encapsulate EAP over wired/wireless LAN, port based, parties are supplicant (S, entity wanting access), authentication server (AS, can be together with authenticator), authenticator (A, access point)
802.1X Setup: S authenticates via A to AS, AS confirm S credentials, AS directs authenticator to allow S access to services
Port-Based Access Control: controlled port accepts packets from authenticated devices, uncontrolled port only passes 802.1X packets
802.1X 4-way Handshake: S start, A request identity, S respond identity, A forward to AS, AS respond with challenge, A forward to S, S respond, A forward to AS, AS respond with OK, A forward, complete
802.1X Keys: Pairwise Master Key (unique to S-A pair), Groupwise Key (for all S connected to same A)
Wi-Fi Protected Access (WPA): basically 802.1X + TKIP (RC4), can also use PSK instead of 802.1X + EAP
Temporal Key Integrity Protocol (TKIP): quick fix for WEP, use existing device calculation abilities to encrypt, has temporal and Msg Integrity Check keys derived from PMK, 48 bit sequence counter
Pretty Good Privacy (PGP): provides authentication (create SHA-1 hash of message and sign with RSA), confidentiality (create random 128 bit session key, randomness from user keystrokes, encrypt message, encrypt session key with RSA/ElGamal & attach), compression (ZIP, order is sign, compress, encrypt), convert raw bits to Radix-64
PGP Key Rings: use least significant 64 bits of public key as identifier, each user has both private key ring (personal key pairs) and public key ring (other public keys)
PGP Trust Model: web of trust, each user can sign public keys of known users, trust levels (undefined, unknown, partial, always, ultimately for own keys), possible to trust public key without trusing user, public key can also be trusted if certified by at least 2 partially trusted users