INTRODUCTION TO COMPUTER SECURITY

Computer Security: prevent/detect unauth. actions by users on computer systems
sensible to Authoriz. & security policy (who? what?)
Information Security: general
Security: protection of **assets** from **threats** (abstract)
Owner wants to protect assets → **countermeasures** to reduce **vulnerabilities**
Threat Agents seek to abuse assets
Safety: different from Security physical protection / mitigating hazards effects
Security Properties: confidentiality, integrity, authentication, availability, accountability
Confidentiality is privacy if for an individual, secrecy if between orgs.
Countermeasures: prevention, detection, response

INTRODUCTION TO CRYPTOGRAPHY

Cryptography: technology that enables us to turn unsecure channels into secure ones
→ need to provide confidentiality, integrity and authentication (non-repudiation desirable)
→ Also **Steganography** exists (science of hiding messages in other messages)
Types of Cryptography: **substitution** & **transposition**
→ security depends on secrecy of key, not the encryption algorithm (P=NP is a threat)
Symmetric: enc/dec keys equals or derivable one from each other
→ **Key exchange problem**: need of secure channel to share key (dog biting his tail)
Asymmetric: different keys → publishing public key doesn't compromise the private one
→ development of **digital signature** to sign a document
Cryptanalyst: job of looking at cipher text and trying to reconstruct either plaintext or key
Unconditional Security: secure even versus infinite computing power (information theory)
Conditional Security: broken in principle but not realistically (complexity theory)
Brute force attack: try every possible key
Security by Obscurity: not revealing how things work (ex: alg)
not applicable (cryptanalysts may find vulnerabilities before serious damage)
Types of attack (difficult to easy for the attacker):
cipthertext only, known plaintext, chosen plaintext (CP), adaptive CP, chosen ciphertext
m → **[e]** → **c** (e ∈ Key-Space is a bijective function from msg-Space to Ciphertext-Space)
encryption schema: two sets *(Ee, Dd)* that 4each ***e*** there's unique *d* such that $Dd=Ee^{-1}$
→ balance between key space length & "key safety"
Chanches of approach in a system: 1) no sec but efficient 2) sec but inefficient
3) **sec and efficient** (client update triggers transf. of the cipher in server: no file transfer)

Types of symmetric encryption with **substitution**:
- **Caesar**: int key for all letters (if k=3 "A" becomes "D") → easy to break
- **Mono-alphabetic**(Caesar generaliz. with 26!): mapping between letters (A→C, B→Z, ..)
→ Statistical problems: **freq. analysis** (ex: natural lang) with letters or digrams (ex: "the")
- **Affine cipher** (ex of mono-alph): e(m) = (a*m+b)%|A| with a & |A| relatively prime
- **Homophonic**: random string in a chosen set (ex: a can be "01" or "10"), mitigate freq. an.
- **Polyalphabetic**: different transformation for every block (sequence of substitutions)
- **Vigenere (ex of poly-alph)**: f = sum (first x letters shifted by n, then other x shifted by m, ..)
- **One time pad** (Vernam): key long as plaintxt, change every time (Unconditional but utopia)
→ Malleability if ∃ F(x),G(x) | F(E(K,M))=E(K,G(M)) ∀ K,M and E(K,M) = K *xor* M
→ Corollary: u can turn c1 in a c2 corresponding to your msg without knowing the key (xor)

Transposition ciphers: set of all transformations like for block of length t and k: set {1,..t}
→ factorial in t (length of msg, instead of length of alphabet) ; contro: letters unchanged
Composite ciphers: subst. → tranpos. → subst. → …

SYMMETRIC CRYPTOGRAPHY

Stream cipher: bit/Byte at time (ES: SSL/TLS ← Approx of Vernam)

Block cipher: each block is en/de-crypted, like a subst. on a large alphabet, >=64bits

→ Ideal: huge subs. / would need a table of 2^n entries for a n-bit block

→ key size of n * 2^n → 2^n! possible transformations.

Feistel ciph. structure: approx the ideal BC with a **product cipher** (comb of other simpler) → dev. a BC with a key-len of k and block-len of n bits = 2^k possible transf. (rather than 2^n!)

Claude Shannon & Subs-Perm. ciphers: confusion & diffusion of msg & keys (S-P boxes)

Festiel in detail: based on invertible block product cipher (partitions input block into 2 halves → multiple rounds with: substitution on left part based on round *f* of right part & subkey), with *f* an S-P network or any cipher. The right part, after the application of *f* with subkey, is xored with the left part. This result will be the new right part, while the previous right part becomes the new left part. → cipher is a concat → decrypt. uses same subkeys but in reverse order

DES: (Plaintxt: 64b + Key: 56b → DES → Ciphertxt), done with 16 rounds of Feistel function, with Fesitel being 2 permutations & S-BOX subs and subkeys derived by key schedule alg.

- **Avalanche effect** (1b input changed→half out. changed) - 2^{56} possible key values
- Broken (Analytic attacks, Timing Attack) - 2DES (112b key) but **Meet-In-TheMiddle**

3DES: extension to overcome short key length → no known attacks (if not 2^{112} brute force)

Ptxt → E → D → E → Ctxt NB1: if k1 = k2 → compatible with DES

(k1) (k2) (k1)

Ptxt ← D ← E ← D ← Ctxt NB2: 3-key 3DES exists (Ek3(Dk2(Ek1(P))))

AES: block size of 128b / 3 keys of 128,192,256 len / fast on both hw&sw / S-P with Feistel

ELECTRONIC CODEBOOK (ECB MODE): msg split in m blocks, each encrypted with same key (→ same problem of subst.) at different times → Limits: Info leak & limited integrity.

CIPHER-BLOCK CHAINING (CBC MODE): At the beginning, InitVector xored with plaintxt, then encrypted with key → C1, also used in xor with P2 for the next phase.

→ Prop: Chaining dependency & Self-Syncro

STREAM CIPHER: to encrypt data flowing → same idea of OTP but with **pseudorandom** gen (seed as key so not as giant as OTP) / faster & easier to implement / keys can be reused / with proper design: as secure as block cipher with comparable key length (es: **RC4**)

PLACEMENT OF ENCRYPTION: **1) Link encr**. (OSI 1,2) & **2) E-to-E encr**. (OSI 3,4,6,7)

1) each link between user and server has to be protected → but nodes can read plaintext!

2) as we move up: more secure but more complex → e-to-e protect the entire path + **auth** header is in plain (network has to route), traffic flow not protected (but there's 1))

→ You don't have to trust nodes (es:Routers) and that's good, but you have to trust the final host (a specific process, not the whole system!) → higher level of encr (ports, socket, …).

Traffic Analysis: just observing the flow of encr. data the other can obtain info

Countermeasures: link encr to protect headers + padding traffic to confuse

Key distribution: (**1)** A physically deliver key to B **2)** third party deliver **3)** previous direct exchange **4)** both rely on third-party) → often problems due to key distrib.

→ Two types of key:

Session key (data between users for one logical session), ephimeral

Master key (to encrypt the Session key), shared by user and the KeyDistrib.Center

Needham-Schroeder Shared Key Proto. (goals: 1) secrecy of Ks 2) freshness 3) key auth)

Fst Step) A send in clear his ID, B's ID and a nounce N (to avoid **Replay Attack**)

Snd) KDC sends back E(Ka,[Ks|IDa|IDb|N1]|E(Kb,[Ks,IDa])) (A can open but not 2nd part)

Trd) A send the sh-key Ks and his ID to B (encrypted with Kb as he received it from KDC)

Fth) B sends a nounce encrypted with Ks to check the auth of A → A will use f to transform

Ffh) A sends f(N2) (f can be every function except identity)

Reply Attack: no freshness ((with no N) C subst. new *2Step* with an old one: old Ks reused)

MESSAGE AUTH. & DIGITAL SIGNATURE

Authenticator (A): value used to auth a message, generated by an auth function

1) **Msg encr**: (A) is the cipher of the entire msg E(Kab,M) → waste of bandwidth
2) crypto fun with I = msg & crypto key → produces (A) called MAC/crypto checksum
3) M → H → h(M) hash code

2) **Crypto fun** in detail:
- the fun is **many-to-one** (potentially >1 msgs with same MAC) but it's really difficult
- **Data auth alg** (to compute MAC) that uses DES (so no longer secure)
→ input split in 64 bit chunks (Time=1: DES(D1, K) produces o1, xored with D2 for next)
→ Attacker can find M' such that E(M', Kab) = E(M, Kab) ← inf card. VS finite card !
CAR problem: initially non-connected things in the real world → examples

3) **Hash** (hash code = msg digest, no key in input) in detail:
sending M, h(M) the intruder can trick B to believe that A sent M', so I send E(Kab,h(M)) aka the fingerprint of M, in order to avoid it. However this isn't 100% a sign of A, because Bob can generate itself Kab and so A can repudiate his own sign to trick Bob (limit of symm crypt). Note that, in order to check if A "is" the sender, B has to compute h(M), decrypt the crypted part and then check if they're equal. Note also that i can send h(M||S) instead.
Hash prop: **1)** H can be applied to a block of data of any sz **2)** produces a fixed-len output **3)** ez to compute (ideally linear) for a given x: not good in pwd context (for attacker is better)
4) **ONE WAY**: given y it's computationally infeasible (c.i) to find a x such that h(x) = y
→ important in msg auth tech with a secret values (A→B:M|H(M|S)) (attacker deconstruct S)
5) **WEAK COLLISION** (pre-img) **resistance**: given x, it's c.i to find y!=x such that h(y)=h(x)
→ important to prevent forgery when encr. hash code is used (A→B:M|E(K,h(M))), means that C can't "sign" a document of not his own.
& protect pwd file storing h(p), often combined with salt ((s,h(s|p)) to protect from dictionary attacks (attacker could download pwd file & then offline try to match pre-computed hashes)
6) **STRONG** C. (2nd pre-img) **res**.: it's c.i to find any pair (x,y) such that h(y)=h(x)
→ against birthday attack (known-plaintxt attack): h sz must be doubles if collision detection is important (with 2^m search-space h must be applied $2^{m/2}$ so that p_of_collision > 0.5)
The attacker can be A itself (A prepares M, good for B, and M' → A generates good msgs and bad ones, and she stops when h(xi) = h(yj) → B signs h(xi) but later on A uses sign for Yj (B has his bad msg, he wants to change words in the good one so that the changed good one (but with same semantics) has same hash of the bad one) → MD5 and SHA

DIGITAL SIGN: 3rd party arbitrer can address problems with 3 mods:
1) conventional encr. & msg visible for arbitrer (msg in plain + timestamp)
2) conventional encr. & msg NOT visible for arbitrer (encr. msg if needed + timestamp)
3) public key encr. & msg NOT visible for arbitrer (not so used)

PGP: openSource proj with: digital sig, msg encr., compression, email compat., segmentat.
1) **Auth only**: hash code of M encrypted by A with PRkey is concatenated with M itself
→ then the dest. B compares hashed M with decrypted M (with PUkey)
2) **Confidentiality only**: M compressed → encrypted with a brand new fresh Ks → results concat with Ks itself (but encrypted with public key of B, so that only B can recover Ks decrypting with his private key), then B decrypt with Ks → key exchange problem is no more
3) **both**: "(1) with (2) built in inside"

Data protection: in transit / at rest / in-use (f.e. in memory)
PGP structure has **private key ring** (sensitive data structure, protected by a private key, usually consisting of an hashed passphrase keeped by the user) and **public key ring** (his integrity is crucial, to avoid possibility of C substituting B public key with his own).

PUBLIC KEY: solution to both key distribution & signature problems
Contro: binding (of PUBk with his owner), that is another type of key distribution problem
Confident.: B: M→E(PUa)→(B to A) A: D(PRa) (B encr with PUa so that only A can read)

Auth.: B: M→E(PRb)→(B to A) A: D(PUb) (B encr with his PR so that A can know he's him)
Requisites: **1)** comput. ez for B to generate pair (PUb, PRb) **2)** comput. ez for sender A, knowing PUb and M, to generate C=E(PUb,M) **3)** comput. ez for receiver B to decrypt C using PRb to recover M **4)** comput. infeasible, for attacker, knowing PUb to determine PRb **5)** comput. infeasible, for an attacker, knowing PUb and C=E(PUb,M) to recover M **(!)** **6)** 2 keys applicable in 2 orders: D(PUb,E(PRb,M))=M=D(PRb,E(PUb,M)) (not alw. necess.)
→ We'll see **RSA** (En/Decr, Digital Sign, Key exchange) & **Diffie-Hellman**(only Key exch)
One-way fun (!= from hash f): ez to compute ∀x but f^{-1} is hard
f.e given p1, p2 primes, let n=p1*p2 & X={1,..n-1}. Def. f:X→N by f(x)=x^3modn → compute f is ez but inverting it is hard (given y and n, find x such that x^3 = y modn)
Trapdoor o-w f (parametric in k, f.e k=p1 & p2): x= f^{-1} is ez if both y & k are known
Public key cryptanalysis: **1)** brute force (counterm. is using large keys, but we need a tradeoff → in practice pubk confined to key managm. & digital sign) **2)** computing PR from PU (no proof that is infeas.) **3)** probable msg attack: short M is sent encrypted with PUa. The attacker computes all Yi=E(PUa,Xi) ∀ possible plaintxt Xi as soon as Yi=C → means that M=Xi → solution? attach random bits to M
Revise of Number Theory: <u>prime factorization, relatively primes, arithm mod & properities</u>
Euler Totient f: reduced set of residues is X={0,n-1}\{not relativ. primes to n} = {relat. prim.}
→ the Totient fun Φ is the cardinality of the residues set, with these properties:
1) Φ(p) = p-1 if p is prime **2)** Φ(pq) = Φ(p)Φ(q) = (p-1)(q-1) if p&q primes
Theor.: $a^{\Phi(n)}$ = 1 modn ∀ a,n such that gcd(a,n)=1 (aka rel. primes) f.e a=3,n=10: 3^4 = 1modn
RSA: security comes from difficulty of factoring large numbers → keys are f of a pair of >= 100 digits primes / n is known by A & B (not prime, but product of two primes) / used to digitally sign documents & sw, for PGP, SSL, …
1) txt split in blocks of floor(log2(n))b (each block is a number M (<n))
2) C=M^e modn ←→ M=C^d modn = M^{ed} modn, with e & d properly chosen: PU(e,n) & PR(d,n)
Req.: **1)** possible finding e,d,n such that M^{ed} modn = M ∀M<n **2)** easy to calculate C & M **3)** unfeasible to determine d given e,n
Alg: **1)** keypair gen
(1) gen. 2 large p&q (primes) **(2)** n = pq & Φ=(p-1)(q-1) **(3)** select e ∈ (1;Φ) rel prime to Φ **(4)** find d such that ed modΦ = 1 **(5)** publish (e,n), keep (d,n) priv., discard p&q
2) encr. with (e,n)
(1) break M into M1,... with Mi<n (2) Ci=Mi^e modn
3) decr. with (d,n): Mi=Ci^d modn
RSA security: **1)** compute d given (e,n) as difficult as factorization (no known polyn. time → but at least 1024b) **2)** computat. of Mi given Ci & (e,n): no proof of necess. to find d to fact. n
Malleability: problem, so RSA often combined with padding (add bits before encr. to detect transform.) ← attacker can obtain ciphers without knowing key → can so alter the cipher
ASYMM. ALG. for secret key distribution (f.e with RSA & browser/S SSL handshake):
B → send "hello" to S
send S(ID) || (e,n) to B←S (**Problem1**: how can B know S auth? We must ensure auth here)
{B guess K, then encrypt with PU provided by S} B → send E((e,n),k) to S
{S obtain K decrypting with (d,n) he has: now k is usable for communications
Problem2: PRk of S has to be protected at rest, but we can't ask everytime for a passphr. We can't solve it! Is the problem only for future communications? No, it's retroactive due to accessing logs with PRk C has discovered → Diffie-Hellman (**perfect forwarding secrecy**)
→ solutions to Problem2: 1) not storing Ks in the server, 2) Diffie Hellman
Diffie-Hellman
Security depends on difficulty on computing discrete logs (see file for explanation)
Alg: **1)** sharing of prime q & primeRoot a of q (both may be public), **2)** A&B randomly

generate Xa & Xb (both <q), **3)** A computes Ya=a^{Xa}mod(q) and B computes Yb=a^{Xb}mod(q), **4)** A&B exchange results **5)** A computes Ka=Yb^{Xa}mod(q) and B viceversa → Ka=Kb=Ks
Pro: better than RSA bcs Ks is created out of nothing & never transmitted: PFS (C can't recover prev communications even "knowing Ks").
Contro: no auth (suffers from MIDattack, see slides) → solution is using RSA combined with D-H in order to sign the exponents.
HSM (f.e smartcard): module with ez interface (hopefully) that decrypts E(Kb, Ks) with PRb
Applications…
→ smartcard helps with digital sign keeping PRkeys (f.e. h(M) goes to a SmartcardReader and is signed with the PR inside the SmartCard, then E(PR, h(M)) is sent back (previa auth.) There're multiple formats for digital sign (f.e embedded in pdf). Note **NFC** bcs IDcard can't be embedded in smartphone but can interact via NFC), and note the **RemoteSignature** scenario, with server that does 2-step auth providing a cell to call to be sure of A auth.
Digital Certificates (DC)
A Man in the Middle attack would be possible, so we use it to serve key distribution & auth with no repudiaton (key role in WebSec): verify auth of PUk by binding PUk to the owner name (stating owner, PUk, issuer (Certification Authority, often built-in in the browser (that shows SHA fingerprint instead of the long key)), …, signature). We write (B, PUb, CA)[PRca] as DC of B issued by CA → MID attack countered
The DC has a certain Life Cycle: B requests DC from CA and he provides it / CA provides a self signed DC to A / PUk exchange from B to A → exchange of msg from A to b
PKI Infrastructure(See slides)
CA is divided between an interface ("front-end") for accepting requests (**Registration Authority**, that is seen by who request) & the inside part for generating ("back-end").
We also have **CRL** (certificate revocation list): before validating certificates, the Browser check if it isn't in the CRL (or better, asks the **OCSP** responder that avoids overhead for browsers that should check the whole (often huge) list).
→ Domain name isn't always enough (f.e Webank.it/com) → **DC types**:
Domain **V**alidation: CA verifies if requester has control of the domain
Org/**E**xtended **V**alidation: CA also verifies real-world ID
→ with DC S auths to C, while with login C auths to S / channel is encrypted end-to-end (TLS to connect in order to "check" it)
Security Protocols (set of rules that determines the exchange of msgs = distributed alg with emphasis on communications) → use crypto mech. to achieve security objectives
How Bob knows that msg comes from Alice (auth + non-rep) & that she just said it (freshness against ReplayAttacks (C re-sending Alice's msg after some time)) → IPsec, SSH, PGP, SSL, Kerberos, … → for sensor nets, pairing of wireless devices, access controls for area wide structures, online payments, … → Best example: auth the carController to open it from remote
Remote Keyless System: CarOwner→KeyFob←→RadioLink←→Receiver→Actuator
Security goal(1): receiver sends unlock command to Actuator only if CarOwner *previously* pressed the button on the KeyFob. But an attacker C can send unlock request to RadioLink!
1) KFob → R: {unlock, SerialNumber (secret shared between KF & R)}
→ Not enough: C can overhear SN and then replay it (auth only of the 1st msg)
2) KFob → R: $\{unlock, SerialNumber\}_k$ (protect secrecy of SN with encrypting)
→ Not enough yet: the msg is auth but "old" (no freshness), because C can send the whole encrypted msg (SN is secret but for C this isn't relevant, he just replies the whole stuff back)
→ Security goal(1) is OK, so we need to write a new one: "*recently*" instead of "*previously*"
3) KFob → R: $\{unlock, T\}_k$ with T=timestamp (R compares local clock with decrypted T, with a ms tolerance → but there's the 2-clocks sync problem, f.e if A brings KF onto an airplane. This problem leads to 2 possible consequences: 1) time of KF is older: still secure against C

but owner can't unlock anymore, 2) time of car is older: more window for C to attack
4) nonces (!= everytime, last for few ms, increment or (better) randomly selected in a range)

1. KF → R : hello 2. R → KF: N 3. KF → R: $\{unlock, N\}_k$

Revise:**Assumptions & goals, types of Attackers, std Attacker model**: *slides 16-18*
Kinds of Attack: REPLAY, ManInTheMiddle, Reflection, Type Flaws
NSPublicKeyProtocol: GOAL: mutual (entity) auth / correctness arg (informal)
Nominal run: **1.** A→B:$\{A,Na\}_{Kb}$ **2.** B→A:$\{Na,Nb\}_{Ka}$ (Bob's auth) **3.** A→B:$\{Nb\}_{Kb}$ (Alice's auth)
→ recall principles can be involved in multiple runs. Goal should hold in all interleaved protocol runs → **MID attack on NSPK** with 2 concurrent runs:
A ← run1 → C ← run2 → B
| $\{A,Na\}_{Kc}$ → | $\{A,Na\}_{Kb}$ → |
| ←$\{Na,Nb\}_{Ka}$ |←$\{Na,Nb\}_{Ka}$ | → solutions: add ID here ({Na,Nb,B})
| $\{Nb\}_{Kc}$ → | $\{Nb\}_{Kb}$ → | → problem: auth flaw (B thinks he's speaking to A)
Type Flaw Attack: msg are bit strings without type info → B parses M "badly"
f.e. **Otway-Rees proto** (has key auth + fresh, but not entity auth & key confirmation → see slides for details of the proto) suffers Type Flaw + Reflection (your own msg defeats u):

1. TF+Refl: after step1 A is in wait, so C can replay s1 in s4 and, if |Kab|=|I,A,B|, A will see I,A,B as the key because they have the same length
2. NO key auth/secrecy: C play S's role (see slides for details)

→ solution: preserve typing info, assigning codes to types & (ap|pre)pending the code
Key exchange with CA:
A→S: A,B S→A: Ca, Cb A→B: Ca, Cb, $\{\{Ta, Kab\}_{ka(-1)}\}_{kb}$
→ MID attack (B believes that was sent by A, so Kac will be used instead of Kab → C can overhear) → countermeasure: be explicit about purpose: include A,B ID's in the last step
Prudent eng:*slides 37-9*
KERBEROS: auth for C/S (mutual), based on NSSK but without nested encr & T instead N
Aims: user pwd never travel over the net & never stored on client's machine (discard after usage) & never stored un-encr even on the server → **Single Sign-On**: user asked for pwd once x session (but not for every service) → auth info only in auth S, not in AppS
Req: secure, reliable, transparent, scalable
Architecture: KerberosAuthenticationServer & TicketGrantingServer (authorization)
[**msgs 1&2** "=" 1&2 in NSSK (C auths, once x session)]:
A logs & req. netw resources → KAS accesses DB and sends to A a session key Ka,tgs (that expire in hours) & an encrypted ticket AuthTicket [Ka,s is derived from user pwd (Ka,s = h(pwd(a)|A)] → both user & server keys must be registered in DB → A types pwd on client to decr. results. Tickets & session key are saved, pwd forgotten → When Ka,tgs expires: logout
[**msgs 3&4** "=" 3&4 in NSSK (C asks for service, once x type of service)]:
???
[**msgs 5&6** (C gets access, once x service session)]:
???
Federation: inter-realm auth with ticker for communication in another net
Limit. of Kerb IV: DOS (C can flood S), nested encr, relies on clocks
Buf Overflow, **IPSEC**, **WebSec**: 📦
Access Controls: Taken Confidentiality,Integrity, and Availability: For C & I, we have seen approaches using cryptography (not suited for operations, so we use **security policies** that define what is allowed) → access restrictions: relationships between subjects & objects.
A **security model** provides a formal representation of a security policy, while a **security mechanism** defines the low level sw/hw functions that implements the controls imposed by the policy and formally stated in the model. ACL can be: **DAC** (based on ID of requestor),

MAC (based on rules by a central authority), **RBAC** (based on roles that users play).
*DAC and RBAC are usually coupled with an administrative policy that defines who can specify rules governing AC
Every request passes through a trusted component called **reference monitor** (with these properties: tamper-proof (impossible altering it or at least with accountability), non-bypassable, security kernel (confined in a limited part of the sys), small).
States: A state of a system is the collection of current values of all its components.The subset addressing protection is the system protection state. Let P be the system state space and Q⊆P be the states in which system is authorized to reside in. A security policy characterizes Q. Hence a security policy partitions the states of the system into secure and unsecure states. A security mechanism prevents a system from entering P\Q. A secure system is a system that starts in an authorized state and can't enter an unauthorized state.
DAC: users own resources and control their access.Owner is able to change its permission.
Matrix Model: M(s, o) = {p ∈ Privileges | (s, o, p) ∈ AC} → Write(S,O,M) $|-_c$ (S',O',M') to denote a transition associated with the command c
The Harrison-Ruzzo-Ullman Model: defines authorization system formalizing how to change access rights or how to create and delete subjects and objects. State transitions described by cmds like: command c(x_1, . . . ,x_k): if r_1 in M(x_{s1},x_{o1}) and. . .r_m in M(x_{sm},x_{om}) then op_1;. . .op_n
Data structures: Access Matrix (S x O: P), ACL (O→(S,P),..), Capabilities List (S→(O,P),..)
Unix: Simple ACLs (actually triples with scheme owner/group/other) → chown/grp/mod
→ File Mode Bits: file permission bits & special mode bits (4 executable files & dirs)
Special bits: When a non-owner executes the file, the process will run with user and/or group permissions set upon it by its owner.
Trojan: Discretionary policies DP do not distinguish users from subjects.Once connected to the system, users originate processes (subjects) that execute on their behalf. DP ignores this distinction and evaluates all requests submitted by a process running on behalf of some user against the authorizations of the user. Since DP doesn't enforce any control on the flow of information, once this information is acquired by a procesd it is possible for processes to leak information to users not allowed to read it.
MAC: AC controlled by comparing security labels (indicating criticality of objs) with formal authoriz of subjects. Mandatory bcs subjects may not grant (> rigid than DAC but > secure). 2 principles to hold for confidentiality: Read down (a subject's clearance must dominate (>=) the security level of the obj being read, while a subject's clearance must be dominated (>=) by the security level of the obj being written → there's a lattice (L(sec levels), <=(partial ord)) → Example on slides
Bell-LaPadula Model: Models confidentiality aspects of multi-user systems (OS, DBMS, ..) BLP models confidentiality by combining aspects of DAC and MAC & its' a static model Multi-level security (**MLS**): mandatory policies prevent information flowing downwards from a high security level to a low one (there's the covert channel problem: the mere existence of the file (not only its content) is an info itself (1 bit of T/F, but still an info))
Biba Model: As Bell-Lapadula but models integrity
Chinese Wall Model: Models Confidentiality with this rule: "there must be no information-flow that causes a conflict of interest" (A subject can access any information as long as it has never accessed information from a different company in the same conflict class).
The Clark-Wilson Integrity Model: f.e. after a day, integrity of withdraw in a bank
RBAC: User Assignment with User-Role & then Permission Assignment with Role-Permission
Roles VS Groups: G define sets of users while roles define sets of privileges. R can be "activated" and "deactivated" by users at their discretion. Gmembership always applies, that is, users cannot enable and disable group memberships (and corresponding authorizations) at their will
→ hierarchies simplify / As DAC an admin is needed to manage static rules in times (adm. policies)
Adm. policies (modifying UA): UEmployee : {Student, !TA} => PTEmployee (a UEmp can become a PTEmp as long as he doesn't have another job yet) → so that means that this can be done